AADL Design-Patterns and Tools for Modelling and Performance Analysis of Real-Time systems

Pierre Dissaux, Jérôme Legrand[1]

Alain Plantec, Mickael Kerboeuf, Frank Singhoff[2]

1: Ellidiss Technologies, 24 quai de la douane, 29200 Brest, France
2: University of Brest/LISyC/UEB, 20 avenue Le Gorgeu, 29238 Brest Cedex 3, France

Abstract: This article deals with performance verifications of architecture models of real-time embedded systems. We focus on models expressed with the AADL language and verified with the real-time scheduling theory. To perform verifications with the real-time scheduling theory, the architecture designers must check that their models are compliant with the assumptions of this theory. Unfortunately, this task is difficult since it requires that designers have a deep understanding of the real-time scheduling theory. In this article, we investigate how to automatically check that an AADL architecture is compliant to this theory. We show how to explicitly model the relationships between an AADL architectural model and the analytical methods proposed by the real-time scheduling theory. From these models, we apply a model-based engineering process to generate a decision tool which is able to decide from an AADL architecture model what are the feasibility tests that the designer can apply.

Keywords: AADL, Real-Time, Performance analysis, Design-patterns

1. Introduction

In [6], we have proposed a set of architecture design-patterns that allows early performance verifications of architecture models. Architecture models are expressed with AADL, a textual and graphical language support for model-based engineering of embedded real-time systems that has been approved and published as SAE Standard AS-5506 [1].

Performance verifications of embedded real-time architectures can be performed with the real-time scheduling theory. Real-time scheduling theory provides analytical methods, called feasibility tests which make possible timing constraints verifications.
Real-time scheduling theory foundations were proposed in 1970 [2] and have led to extensive researches. However, it appears that in many practical cases no such analysis is performed with this theory although experience shows that it could be profitable.

Indeed this theory is not easy to understand and to apply for many engineers. Most of the known feasibility tests have been elaborated during the last 30 years. Feasibility tests provide a way to compute different performance criteria such as worst case thread response time. But each criterion requires that the target system fulfills a set of specific assumptions that are applicability constraints. Thus, due to the large number of feasibility tests and due to the large number of applicability constraints, it may be difficult for a designer to choose the relevant feasibility test for a given architecture to analyze.

In [6], we have proposed an approach based on design-patterns in order to ease usability of the real-time scheduling theory. We have defined four design-patterns called «Synchronous data flow», «Ravenscar», «Blackboard» and «Queued buffer». These design-patterns model usual communication paradigms of multitasked real-time software. For each design-pattern, we have identified which feasibility tests the designer can compute to perform the verification of his AADL architecture. This approach had two weaknesses. First, we have assumed that the designer is able to check that his AADL architecture is compliant with the design-pattern he has chosen. Second, for a given AADL design-pattern, many feasibility tests may exist. For example, in the case of the «Synchronous data flow» design-pattern, we have listed 126 possible cases in which several feasibility tests can be applied. It implies that only defining a set of design-patterns may not be enough to really help the designer.

In this article, we investigate how to automatically check that an AADL architecture is compliant to a design-pattern and a set of feasibility tests. We show how to explicitly model the relationships between an architectural design-pattern and the compliant feasibility tests. From these models, we apply a model-based engineering process to generate a

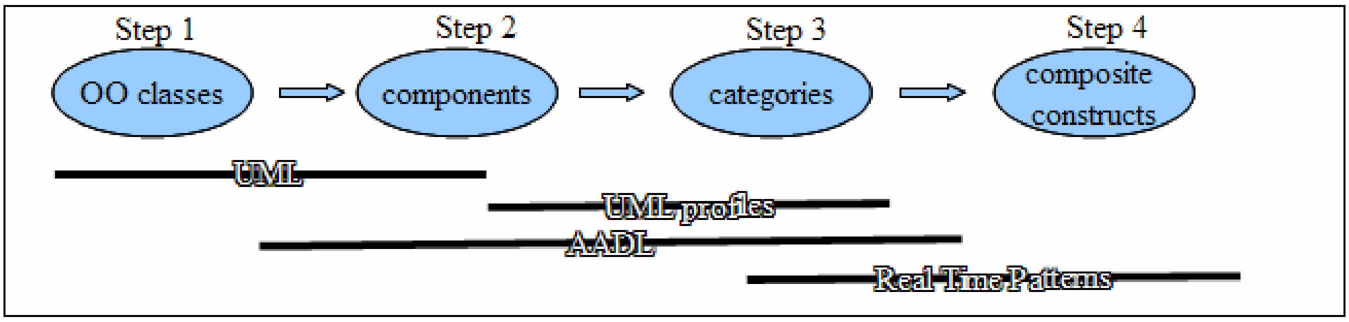


Fig 1. From object oriented modeling to design-patterns

decision tool which is able to identify, from an AADL architecture model, the feasibility tests the designer is allowed to compute. Then, this decision tool helps the designer to choose the feasibility tests that he is allowed to apply to his AADL architecture models.

This article is organized as follows. In section 2, we describe the set of design-patterns we consider. We also introduce AADL, the architecture language we promote for the modeling of both the architecture to analyze and our set of design-patterns. Section 3 presents Platypus, the model-based engineering tool we use to generate the decision tool. In section 4, we show an example of the use of this tool with one of our design-patterns: the «Synchronous data flow» design-pattern. Then, section 5 is devoted to related works and we conclude and present future works in section 6.

2. AADL Real Time design-patterns

During the last decades, a lot of emphasis has been given to software modeling techniques, in a continuous move from traditional coding activities to higher level of abstractions.

First step in this advancement has been the generalized usage of Object Oriented paradigms in modeling languages, especially through class diagrams. Such a representation is perfect for static data modeling and meta-modeling activities, but is not usually appropriate to highlight dynamic interactions of system and software architectures.

That's why components appeared in a second step, which extend the OO model with concepts of provided and required interfaces (black box view) and internal composition (white box view). With components, it becomes easier to describe functional interactions between well identified subsystems and to manage complex system and software architectures in a modular way.

However, as far as real-time systems are concerned, not only the applicative architecture must be described, but also its interaction with the underlying executive. This aspect is not supported by simple component based models, thus a third step can be identified by the availability of categorized components. This categorization aims at providing a stronger semantics to enrich the basic concept of component. As an example, a thread is a component which can be scheduled by the run-time executive. Several standardized languages such as the MARTE profile for UML [7] or the AADL provide a set of categorized components that are appropriate for real-time system and software modeling activities.

Nevertheless, although it becomes now easier to describe real-time architectures, their validation still remains a subject of investigation. For instance, the lack of a single property may sometimes prevent a "correct" real-time architecture from being properly processed by a schedulability analysis tool.

That is why, the next step in the improvement of the development process of real-time systems consists in providing to the end user a set of predefined composite constructs that match known real-time analysis solutions. The composite constructs we have studied correspond to the various inter-thread communication paradigms that can be applied in an AADL architecture and can be seen as real-time design-patterns.

AADL is used to design and analyse software and hardware architecture of embedded real-time systems. Many tools provide support for the modelling and the analysis of AADL models. Ocarina implements Ada and C code generators for distributed systems [16]. TOPCASED, OSATE and Stood provide AADL modelling features [22,19,20]. The Fremont toolset and Cheddar implement AADL performance analysis methods [21,13]. An updated list of supporting tools can be found on the official AADL web site: http://www.aadl.info.

We proposed four AADL architecture design-patterns called “Synchronous data flow”, “Ravenscar”, “BlackBoard” and “Queued Buffer”. A detailed description of them is given in [6]. The next section will give a short description of them.

2.1 Synchronous data flow design-pattern

With this design-pattern, threads are periodic and communication is achieved with AADL data ports. This architectural pattern is inherited from Meta-H [1].

In this synchronization schema, the thread dispatch is not affected by the inter-thread communications that are expressed by pure data flows. Each thread reads input data ports at dispatch time and writes output data ports at completion time. In this simple case, the execution platform consists in one processor running a scheduler such as Rate Monotonic [2].

2.2 Ravenscar design-pattern

The main drawback of the previous design-pattern is its lack of flexibility at run-time. Each thread will always execute, read and write data at pre-defined times, even if useless. In order to introduce more flexibility, asynchronous inter-thread communications can be proposed.

An example of such a run-time environment is given by the Ravenscar profile. Ravenscar is a part of the Ada 2005 standard [17]. It is a set of Ada program restrictions usually enforced at compilation time, which guarantee that the software architecture is real-time scheduling theory compliant. Ravenscar is an Ada subset where real-time applications are composed of a set of threads and shared data.

Ravenscar assumes that threads are scheduled with a fixed priority scheduler and that data components are accessed with ICPP (Inheritance Ceiling Priority Protocol) [18].

In this second design-pattern data component access may occur at any time.

2.3 Blackboard design-pattern

Ravenscar allows a thread to allocate/release several AADL data components. Real-time scheduling theory usually models such a shared resource as a semaphore to handle concurrent access. In classical operating systems, many synchronization design-patterns exist such as critical sections, barriers, readers-writers, private semaphores and various producers-consumers synchronizations [23].

The blackboard design-pattern implements a readers-writers synchronization protocol. At a given time, only one writer can get the access to the blackboard in order to update the data component, as opposed to the readers which are allowed to read the data component simultaneously. The usual implementation of this protocol implies that readers and writers do not perform the same semaphore access, thus, it requires extra analysis.

2.4 Queued buffer design-pattern

In the blackboard design-pattern, at any time, only the last written message is made available to the threads.

Some real-time execution platforms provide communication features which allow all written messages to be stored in a buffer. AADL also proposes such a feature with event data ports. The Queued buffer design-pattern models such a communication. For this design-pattern, an analysis tool should provide some means to perform buffer dimensioning verifications.

2.5 Pattern notation

Each design-pattern presented above is always composed of the same items, according to the design-pattern language we are using. These items are described as follow:

- **Name**: the design-pattern name is a unique and representative name; the only use of the name should immediately recall the what and the how covered by the design-pattern;
- **Synoptic**: gives a very general description of what is covered by the design-pattern and of how it is covered;
- **Context**: the context is one or several situations in which the design-pattern may apply. The context may include the kind of problem for which the design-pattern is supposed to give a well accepted and tested solution.
- **Keywords**: a list of representative words which may be used as representative keys to help determine the application of the design-pattern and help finding design-patterns that apply to a specific project, especially on-line.
- **Predecessors**: more general design-patterns.
- **Solution**: the description of the technical solution illustrated with an AADL model.
- **Successors**: may give some other design-pattern names which are applicable in a more specific context.
- **References**: A set of reference to other design-patterns or information relevant to the context and solution.

In this section, we have presented four AADL design-patterns that are compliant with the real-time scheduling theory. In the next section, we present the engineering environment that we use to model the design-patterns and the real-time scheduling theory feasibility tests: the Platypus environment and the STEP/EXPRESS framework.

3. The STEP/EXPRESS Data exchange Framework

ISO 10303 provides a neutral mechanism for describing product data throughout the life cycle of a product, independent of any particular computer-aided system. ISO 10303 is suitable for file exchange and for implementing, sharing, and archiving product databases.

3.1 Modelling with EXPRESS

For the building of a STEP data exchange component, the EXPRESS data modeling language is used in order to describe data which are to be exchanged. For such a purpose, data schemas are specified with entity descriptions and constraints. The possibility to add constraints allows the specification of domain rules. Constraints can be either local or global. From a dynamic point of view, a data set is considered as conform to an EXPRESS schema if all local and global constraints specified within the schema are satisfied.

As an example, consider the simple EXPRESS model given in figure 2. This model is made of two schemas. The first schema, named *Architecture,* specifies a periodic thread concept with a deadline and a period. Each instance of *Periodic_Task* is constrained to have a period greater than its deadline. The second schema, named *Deadline_On_Request_System,* contains a global rule which constraints further each instance of *Periodic_Task:* given that a system is modeled with a set of *Periodic_Task* instances, the constraint ensures that such a system is made of at least two threads and that for each thread, its period is equal to its deadline.

3.2 Working with the Platypus environment

Platypus [http://cassoulet.univ-brest.fr/mme] is a software engineering tool which embeds a modeling environment based on the STEP standard.

First of all, Platypus is a STEP environment, allowing data modeling with the EXPRESS language and the implementation of STEP exchange components automatically generated from EXPRESS models. From this point of view, Platypus is a typical STEP based tool with an EXPRESS editor and checker, and a STEP file reader, writer and checker.

```
SCHEMA Architecture;

 ENTITY Periodic_Task;
    Deadline : Natural_Type;
    Period : Natural_Type;
  WHERE
    wr1 : Deadline <= Period;
  END_ENTITY;
END_SCHEMA;

SCHEMA Deadline_On_Request_System;
  USE FROM Architecture;

  (* all tasks must have period = deadline *)
  RULE Period_Equal_Deadline_Rule
  FOR ( Periodic_Task );
  WHERE
    at_least_two_tasks :
      SIZEOF ( Periodic_Task ) > 1;
    period_equal_deadline :
      SIZEOF ( QUERY ( p <* Periodic_Task |
        p.Period <> p.Deadline ) ) = 0;
  END_RULE;
END_SCHEMA;
```

Fig 2. Modelling of a thread constraint

Platypus is also an object oriented development tool. It is implemented inside *Pharo* [http://www.pharo-project.org], a free Smalltalk environment. Thanks to *Pharo*, *Platypus* is an hybrid tool. On one hand, it allows very precise data specification and manipulation of statically typed objects. On the other hand, associated with code generators, it allows rapid system prototyping and efficient code maintenance. *Platypus* is developed to be a schema mapping tool allowing the specification of mapping rules between source and target schemas. Mapping rules are designed with EXPRESS and can be interpreted or translated to Smalltalk.

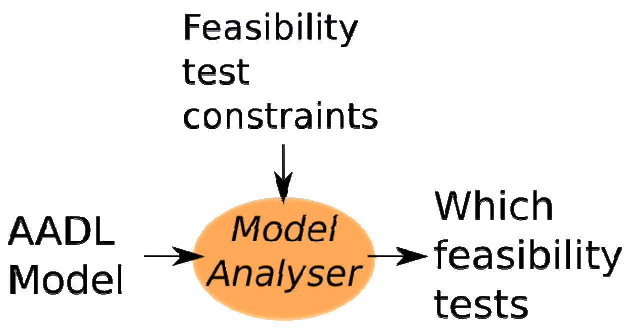


Fig 3. AADL model analyser overview

4. EXPRESS modeling of feasibility tests and architecture

Let see now how to model both feasibility tests and architectural design-pattern with EXPRESS. Given a feasibility test *FT*, it is possible to formally specify which applicability constraints the architecture model has to satisfy for the feasibility test *FT to be* applicable. This set of constraints can be specified in a *FT* specific meta-model.

Thus, a set of meta-models can be designed, one per feasibility test and used in order to help designers for the checking of their real-time system models. The figure 3 depicts a global view of our model analyzing tool which is using these meta-models in order to find which feasibility tests are to be performed for AADL model.

4.1 The design of the model analyzer

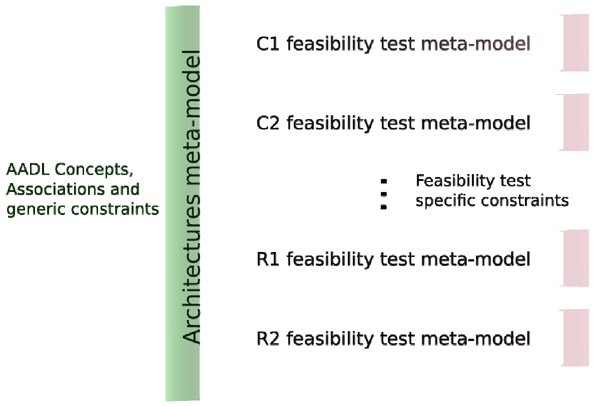

AADL Concepts, Associations and generic constraints
Architectures meta-model
C1 feasibility test meta-model
C2 feasibility test meta-model
Feasibility test specific constraints
R1 feasibility test meta-model
R2 feasibility test meta-model

Fig 4. The model analyzer conceptual components

As shown by the figure 4, the analyzer is designed from two main conceptual parts:

- The first part (left side of the figure 4) consists in an AADL meta-model named *Architectures* meta-model. This meta-model specifies the AADL concepts, their associations and constraints. The important point is that this meta-model specifies all the concepts needed in order to build a simplified AADL parser and to check AADL models. In other words, from an AADL model, it is possible to instantiate the *Architectures* meta-model and use this instance for an analysis.

- The second part (right side of the figure 4) is made of a set of meta-models. Each of them is a specialization of the *Architectures* meta-model and is specific to a particular feasibility test. Such a feasibility test meta-model specifies the constraints which are to be satisfied for the related feasibility test to be applicable. In other words, if an *Architectures* meta-model instance built from an AADL model satisfies all constraints specified by a feasibility test meta-model, it means that the related feasibility test is applicable to the AADL model.

4.2 The prototype implementation

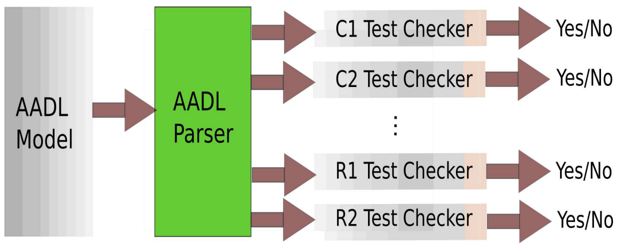


Fig 5. The model checker implementation

The prototype is made of an AADL parser and of feasibility test checkers. The AADL parser is classically implemented from an ADDL grammar and is made to build instances of the *Architecture* meta-model. This AADL parser is dedicated to our design-patterns: it is only able to parse AADL models that are composed of the AADL component categories of our design-patterns. Each test checker is automatically built from the corresponding feasibility test meta-model.

From a particular AADL model (see figure 4), an AADL meta-model instance is built by the parser, then, each test checker evaluates it. As an example, if all constraints of the C1 meta-model are satisfied, then, the C1 test checker result is true. It means that the designer can use the C1 feasibility test in order to evaluate the performance of its architecture model.

4.3 Example of the Synchronous data flow design-pattern

In the previous section, we have presented the overall approach which allows a designer to decide which feasibility tests he can apply on a given AADL model compliant to one of the design-patterns presented in section 2. In the sequel, we illustrate the approach with the simplest design-pattern: the Synchronous data flow design-pattern. First, we present an example of feasibility test that can be applied on the Synchronous data flow. Then, we present the EXPRESS models which allow Platypus to check an AADL model. Finally, we present a screenshot of the Platypus output.

4.3.1 Performance analysis of the Synchronous data flow design-pattern

From an AADL model compliant to the Synchronous data flow design-pattern, we can perform performance analysis based on real-time scheduling theory.

The thread components of the Synchronous data flow design-pattern are periodic threads [2] defined by three parameters: its deadline (*Di*), its period (*Pi*) and its capacity (*Ci*). *Pi* is a fixed delay between two release times of the thread *i*. Each time the thread *i* is released, it has to do a job whose execution time is bounded by *Ci* units of time. This job has to be ended before *Di* units of time after the thread wake up time.

Some algebraic methods can provide a proof that a model compliant to the Synchronous data flow design-pattern will meet its periodic thread performance requirements. Scheduling algorithms allow the designer to compute scheduling simulations of the architecture to analyze. Usually, simulations can not lead to a proof. However, in some cases (with deterministic schedulers and with periodic threads for example), scheduling simulation may lead to a schedulability proof if the designer is able to compute a scheduling during the base period [3].

Different kinds of feasibility tests exist for the Synchronous data flow design-pattern [4]: tests based on processor utilization factor (noted C1,C2,..,Cn tests in the figure 4) and tests based on worst case thread response time (noted R1, R2, …, Rn tests in the figure 4).

The worst case response time feasibility test consists in comparing the worst case response time of each thread with its deadline. Joseph, Pandia, Audsley *et al.* have proposed a way to compute the worst case response time of a thread with pre-emptive fixed priority scheduling by:

$$\boxed{R_i = C_i + \sum_{j \in hp(i)} \left(\frac{R_i}{P_j} \right) . C_j}$$

Eq. 1: compute worst case response time of a periodic thread i

Where R_i is the worst case response time of the thread *i*.

4.3.2 EXPRESS modeling of the Synchronous data flow design-pattern and its feasibility tests

Let see now the EXPRESS models for the Synchronous data flow design-pattern. Figures 6, 7 and 8 present the three EXPRESS models (schemas) that are required to produce the decision tool able to check if a given AADL model is compliant to the Synchronous data flow design-pattern.

```
SCHEMA Architecture;
  ENTITY Generic_Scheduler;
    Quantum : Natural_type;
    Preemptive_Type : BOOLEAN;
  END_ENTITY;

  ENTITY Rate_Monotonic_Protocol
    SUBTYPE OF (Generic_Scheduler);
  END_ENTITY;

 ENTITY Periodic_Task;
    Capacity : Natural_Type;
    Deadline : Natural_Type;
    Period : Natural_Type;
    Release_Time : Natural_Type;
    Priority : Priority_Type;
    Blocking_Time : Natural_Type;
  WHERE
    wr1 : Deadline <= Period;
  END_ENTITY;
END_SCHEMA;
```

Fig 6. EXPRESS modeling of the architecture

A first EXPRESS schema (figure 6), called `Architecture`, depicts the architecture point of view of the design-pattern. From section 2.1, we know that only one type of component is used in this design-pattern: AADL thread components. Schema `Architecture` defines all thread component attributes that are required by the feasibility tests (e.g. priority, deadline, period, …). The `Architecture` schema also defines the components that are part of the execution environment (e.g. scheduler) and that required for the analysis.

The third EXPRESS schema (figure 8), called `Feasibility_Tests`, specifies the different feasibility tests which can be applied to the Synchronous data flow design-pattern. This schema also includes a model of the applicability constraints of the feasibility tests. Remember that these constraints must be met by the AADL architecture to analyze. These feasibility test constraints are stored in separate schemas. For example, schema `Simultaneous_Release_Time_Constraint` and `Period_Equal_Deadline_Constraint` respectively specify that the threads of the Synchronous data flow design-pattern are released at the same time and that the thread deadlines are equal to their periods. Figure 7 shows a part of these schemas.

```
SCHEMA Simultaneous_Release_Time_Constraint;
  USE FROM Architectures;

  RULE Simultaneous_Release_Time
   FOR (Periodic_Task);
   LOCAL
     nbpt : INTEGER := SIZEOF (Periodic_Task);
     p1 : Periodic_Task := Periodic_Task [1];
   END_LOCAL;
   WHERE
   (* All tasks share the same release time *)
   r1 : ( nbpt < 2 ) OR
    (SIZEOF (QUERY(p <* Periodic_Task |
       p.Release_Time <> p1.Release_Time))= 0);
  END_RULE;
```

```
END_SCHEMA;

SCHEMA Period_Equal_Deadline_Constraint; …
```

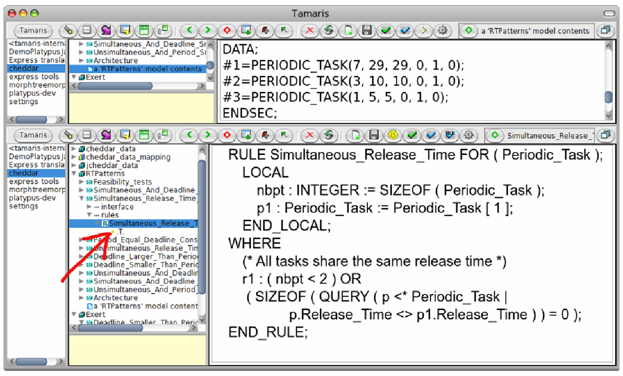
Fig 7. EXPRESS modeling of the feasibility tests constraints

```
SCHEMA Feasibility_tests;
 ENTITY Response_Time  …

SCHEMA Simultaneous_And_Deadline_Equal_Period;
 USE FROM Architecture;
 USE FROM Feasibility_Tests;
 USE FROM Simultaneous_Release_Time_Constraint;
 USE FROM Period_Equal_Deadline_Constraint;

 ENTITY Test_C1 SUBTYPE OF …
 END_ENTITY;

 ENTITY Test_C7 SUBTYPE OF …
 END_ENTITY;

 ENTITY Test_S1 SUBTYPE OF …
 END_ENTITY;

 ENTITY Test_R1 SUBTYPE OF …
 END_ENTITY;

 ENTITY Test_R2 SUBTYPE OF …
 END_ENTITY;

END_SCHEMA;
```

Fig 8. EXPRESS modeling of the feasibility tests

4.3.3 Example of use of the decision tool

The figure 9, shows the Platypus environment checking feasibility test applicability constraints of an architecture. Two opened panes are presented in this figure. The top pane shows the schema instance editor containing three periodic threads. These three instances are extracted from the AADL model and constitute the current architecture. Note that the current prototype does not handle AADL files: the architecture model is loaded from STEP files. The bottom pane shows the *Simultaneous_Release_Time* constraint and the result of its evaluation which is *true* (see the ***.T.*** pointed out by the arrow).


Tamaris
a 'RTPatterns' model contents
<tamaris-intern
DemoPlatypusJ
Express transla
cheddar
express tools
morphtreemorp
platypus-dev
settings
Simultaneous_And_Deadline_Sr
Unsimultaneous_And_Period_Sr
Architecture
Exert
cheddar_data
cheddar_data_mapping
jcheddar_data
RTPatterns
Feasibility_tests
Simultaneous_Release_Time
interface
rules
Simultaneous_Release_T
.T.
Period_Equal_Deadline_Cons
Unsimultaneous_Release_Tim
Deadline_Larger_Than_Perio
Deadline_Smaller_Than_Peric
Unsimultaneous_And_Deadlin
Simultaneous_And_Deadline
Unsimultaneous_And_Period
Deadline_Smaller_Than_Peric

```
DATA;
#1=PERIODIC_TASK(7, 29, 29, 0, 1, 0);
#2=PERIODIC_TASK(3, 10, 10, 0, 1, 0);
#3=PERIODIC_TASK(1, 5, 5, 0, 1, 0);
ENDSEC;
```

```
RULE Simultaneous_Release_Time FOR ( Periodic_Task );
  LOCAL
    nbpt : INTEGER := SIZEOF ( Periodic_Task );
    p1 : Periodic_Task := Periodic_Task [ 1 ];
  END_LOCAL;
WHERE
  (* All tasks share the same release time *)
  r1 : ( nbpt < 2 ) OR
   ( SIZEOF ( QUERY ( p <* Periodic_Task |
          p.Release_Time <> p1.Release_Time ) ) = 0 );
END_RULE;
```

Fig 9. A Platypus screenshot showing an *Architecture* schema instance together with a constraint and the result of its evaluation

5. Related works

This article has shown an approach to check that an architectural model of a real-time system is compliant to a set of constraints. Many other approaches also investigated how to perform such verifications.

UML together with its standard constraint language OCL could be used for the purpose of designing and building feasibility test checkers. But as far as we know, our approach has not been investigated with UML tools.

In [12], Gilles and al. have proposed a similar constraint language for AADL. The proposed language is called REAL (REAL stands for Requirement Enforcement Analysis Language). REAL is developed by Télécom-Paris-Tech and ISAE. It is an annex of the AADL standard. This language is then specifically designed for the modeling of real-time architectures. REAL allows to express various type of constraints on AADL architecture and their authors have shown that it can express some of the applicability constraints of the real-time scheduling theory.

Another example of a similar move towards more analyzable constructs built on top of a modelling language can be found in the history of the HOOD method [8]. The first versions of this modelling approach defined a quite basic concept of component (called HOOD objects) which aimed at representing more or less an Ada 83 package. In 1995, two specializations of HOOD were specified: HOOD 4 [9] which targets Object-Oriented programming languages and especially Ada 95, and HRT-HOOD [10] which goal is to comply with the Ada Ravenscar model (now included into Ada 2005). In both cases, the original concepts and principles of the HOOD methodology have been kept, and specific composite constructs have been identified in order to support properly Ada 95 tagged types or Ravenscar cyclic, sporadic and protected objects.

More recently, in the context of the IST-ASSERT project, Panunzio and al proposed to integrate some HRT-HOOD components with UML models [11]. For such a purpose, they have proposed an engineering process based on a meta-model called RCM (RCM stands for Ravenscar Computational Model). In this process, performance verifications are performed with the MAST framework [14].

6. Conclusion

In this article, we have presented an approach that allows an architecture designer to automatically check that his models are compliant with the assumptions of the real-time scheduling theory. This theory provides analytical methods, called feasibility tests, which allow designers to perform verifications on architecture models.

We showed how to explicitly model the relationships between an architectural model and the feasibility tests with EXPRESS. From these EXPRESS models, we apply a model-based engineering process to generate a decision tool which is able to identify the compliant feasibility tests the designer is allowed to compute.

The current decision tool is a prototype inside the Platypus environment. In the next months, we plan to write a new version of this decision tool that can be embed into Cheddar [13].

Cheddar is an Ada tool which aims at performance analysis of real-time architectures. It includes numerous feasibility tests and most of the most classical scheduling algorithms of the real-time scheduling theory. Cheddar is already able to perform verifications of AADL models but today, Cheddar's users have to choose which feasibility tests to apply to their AADL models. The integration of the decision tool proposed in this article will increase Cheddar's usability.

A second possible extension of the work presented in this article would address the type of analysis the decision tool is able to produce. Indeed, in the current approach, we only check that a given architecture model is conform to a given design-pattern. If the architectural model is conforming to the design-pattern, the tool is able to list the compliant feasibility tests. But if not, such a decision tool should provide model metrics [15] to designers in order to increase their model compliance to the real-time scheduling theory. In a second step, we plan to investigate the relevant metrics for our different AADL design-patterns.

7. References

[1] SAE, Architecture Analysis and Design Language (AADL) AS 5506 ; Technical report, The Engineering Society For Advancing Mobility Land Sea Air and Space, Aerospace Information Report, Version 1.0, November 2004.

[2] Scheduling algorithms for multiprogramming in a hard real-time environnment . C. L. Liu et J. W. Layland. Journal of the Association for Computing Machinery, vol. 20, n°1, pp. 46- 61, January, 1973 .

[3] On the Complexity of Fixed-Priority Scheduling of Periodic, Real-Time Tasks. J.Y.T. Leung and J. Whitehead. Performance Evaluation 2, 237-250 (1982).

[4] Pre-emptive and Non-Preemptive Real Time Uni-Proces-sor Scheduling. L. George, N. Rivierre and M. Spuri. INRIA Research Report number 2966. September 1996.

[5] ISO, Ada reference manual ISO/IEC 8652:1995(E) with Technical Corrigendum 1 and Amendment 1 (Draft 16).

[6] Stood and Cheddar : AADL as a pivot Language for Analysing Performances of Real Time Architectures. P. Dissaux and F. Singhoff. 4th European Congress ERTS Embedded Real Time Software, January 2008.

[7] UML Profile for MARTE: Modeling and Analysis of Real-Time Embedded Systems Version 1.0, OMG, 2009

[8] HOOD Reference Manual release 3.1, HOOD User Group, Masson & Prentice-Hall, 1993.

[9] HOOD Reference Manual release 4.0, HOOD User Group, 1995.

[10] HRT-HOOD: A Structured Design Method for Hard Real-Time Ada Systems. A. Burns, A. Wellings. Elsevier, 1995

[11] A Meta model-Driven Process Featuring Advanced Model-Based Timing Analysis. M. Panunzio, T. Vardanega. 2007, June, Proceedings of the 12th International Conference on Reliable Software Technologies, Ada-Europe. Geneva, LNCS springer-Verlag.

[12] Expressing and enforcing user-defined constraints of AADL models. O. Gilles, J. Hugues. 5th international workshop on AADL and UML. In the proceedings of the 15th IEEE International Conference on Engineering of Complex Computer Systems, pages 337-348, University of Oxford, UK, 22-26 March 2010.

[13] Investigating the usability of real-time scheduling theory with the Cheddar project. F. Singhoff, A. Plantec, P. Dissaux and J. Legrand. Journal of Real-Time Systems, volume 43, number 3, pages 259-295. November 2009. Springer Verlag. ISSN:0922-6443.

[14] MAST: Modeling and Analysis Suite for Real-Time Applications. M. González Harbour, J.J. Gutiérrez García, J.C. Palencia Gutiérrez, J.M. Drake Moyano. June 2001, pages 125--134, Proc. of the 13th Euromicro Conference on Real-Time Systems, Delft, The Netherlands.

[15] Model-driven Engineering Metrics for Real-Time Systems. M. Monperrus, J.M. Jezequel, J Champean, B. Hoeltzener. 4th European Congress ERTS Embedded Real Time Software, January 2008.

[16] Rapid Prototyping of Distributed Real-Time Embedded Systems Using the AADL and Ocarina. J. Hugues, B. Zalila, and L. Pautet. In 18th IEEE/IFIP International Workshop on Rapid System Prototyping (RSP'07), Porto Allegre, Brésil, June 2007.

[17] ISO, Ada reference manual ISO/IEC 8652:1995(E) with Technical Corrigendum 1 and Amendment 1 (Draft 16).

[18] Concurrent and Real Time programming in Ada. 2007. A. Burns and A. Wellings. Cambridge University Press.

[19] Using AADL for mission critical software development. P. Dissaux. 2nd European Congress ERTS (Embedded Real Time Software), 21-23 january 2004.
[20] SEI. OSATE : an extensible Source AADL tool environment. SEI AADL team technical report. December 2004.
[21] Schedulability analysis of AADL models. O. Sokolsky, I. Lee, D. Clake. Parallel and Distributed Processing Symposium, IPDPS, 25-29 April 2006.
[22] TOPCASED web site. http://www.topcased.org
[23] Modern Operating Systems. A. Tanenbaum. Prentice-Hall. 2001.