AADL Real-time design-pattern Automatic Recognition

Pierre Dissaux (pierre.dissaux@ellidiss.com), Jérôme Legrand
(Ellidiss Technologies)
Vincent Gaudel, Alain Plantec, Stéphane Rubini, Frank Singhoff
(Université de Bretagne Occidentale, LISyC, UEB)

Abstract

This article deals with performance verifications applied to architecture models of real-time embedded systems. We focus on models that can be 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 help designers to check that their architecture models are compliant with this theory. We show how to explicitly model the relationships between an architectural model and real-time scheduling analysis methods. From these models, we apply a model-based engineering process to generate a recognition tool that is able to detect from an architecture model which are the analysis methods that can be applied.

1) Introduction

The SAE Architecture Analysis and Design Language (AADL) is a textual and graphical language for model-based engineering of embedded real-time systems that has been approved and published as the SAE Standard AS-5506A [2,7]. AADL is used to design and analyze the software and the hardware architecture of embedded real-time systems. To perform verifications of AADL models, analysis tools based on the real-time scheduling theory [1,4,5] such as Cheddar [6] can be used. Real-time scheduling theory helps the system designer to predict the timing behavior of a set of real-time tasks with scheduling simulation and feasibility tests. Scheduling simulation requires, first to compute a scheduling on a given time interval and second, to look for timing properties in this computed scheduling. On the contrary, feasibility tests allow the designer to study a set of real-time tasks without computing scheduling. To apply a real-time scheduling tool, architecture designers need to check that their models are compliant with the assumptions of this theory. Unfortunately, this task is difficult since it requires a deep understanding of the real-time scheduling theory. In this article, we investigate a solution to automatically check whether a given AADL specification is compliant to this theory or not. 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, a model-based engineering process is proposed to generate a recognition tool which is able to decide which feasibility tests can be applied by analysis tools. For such a purpose, the recognition tool handles AADL design-patterns.

In the sequel, we first present some related works. Then, we introduce a set of AADL design-patterns and explain how they are used by our recognition tool to ensure feasibility tests applicability. A brief description of the recognition tool implementation is then given. Finally, we shortly explain how we expect to integrate our propositions in an industrial toolset called “AADL Inspector”.

2) Model analysis and related works

Model Driven Engineering (MDE) is more and more perceived as being the way to go to improve at the same time quality and productivity of software intensive systems developments. One of the most valuable awaited benefits of this approach is the ability to introduce early analysis in the development process, in order to support verification or automatic code generation activities. However, concrete realizations of such associations between modeling and analysis technologies face various issues:

- Issue number 1, soundness of the modeling languages: Modeling languages, and especially the more general purpose ones, usually suffer from a lack of strictness in the definition of their semantics. The user must then choose between significantly restricting the scope of the applicable verification techniques, and overloading the semantics of the modeling language. The latter often leads to loosing the benefit of using sharable standardized models.
- Issue number 2, adaptability of the analysis approaches: Most verification techniques have been developed independently and are based on specific data representations that may be quite far from end-user applicative models. Moreover, analysis techniques are usually available though pre-existing tools that do not necessarily comply with tool chain integration requirements.
- Issue number 3, compatibility between the characteristics of the applicative model and the scope of the analysis approach: Verification techniques usually have a restricted scope and can only address a subset of what can be legally specified with a modeling language.

The next paragraphs show three examples of concrete realizations, among many others, of early analysis approaches in Model Driven Engineering tool-chains and targeting an improvement of the development of critical real-time embedded software.

2.1) AADL model checking with TINA

A set of tools have been integrated in the context of the SPICES European research project (2006-2009, http://www.spices-itea.org) [9], in order to provide a complete behavioral verification chain at a model level. This tool-chain is composed of a graphical editor to build AADL models (ADELE), a model transformation engine using a pivot language (FIACRE) and a pre-existing verification tool based on time Petri nets analysis (TINA).

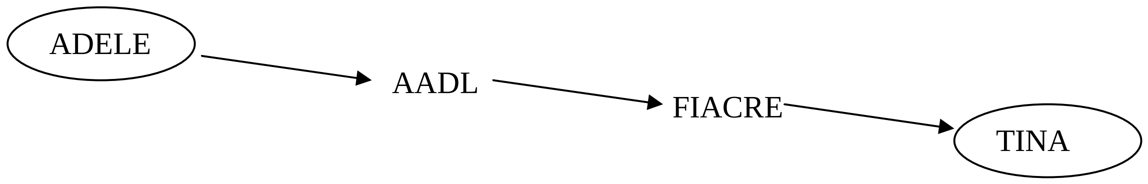


Figure 1- AADL model checking with TINA

With this approach, use of the TINA model checking tool has been made possible thanks to a dedicated model transformation that converts the AADL specification into the corresponding description in FIACRE pivot

language. FIACRE models are then compiled into a Time Transition System (TTS) which is the formal specification representation used by TINA.
This tool-chain is currently being improved in the context of the QUARTEFT project funded by the FRAE (French foundation for research in aerospace).

2.2) AADL automatic code generation with OCARINA

The TASTE toolset results from the ASSERT European research project (2004-2007, http://www.assert-project.net/) [10] and additional tool development supported by the European Space Agency. The rationale of TASTE is to enable modeling of correct by construction heterogeneous systems in order to automatically build homogeneous distributed software applications. This tool-chain is composed of a set of modeling and integration tools that produce a complete AADL specification of the system, where software data and function details remain expressed in specialized languages (Simulink, SDL, ASN.1, C, Ada), and the OCARINA tool that compiles all that together and generates the complete set of executable files.

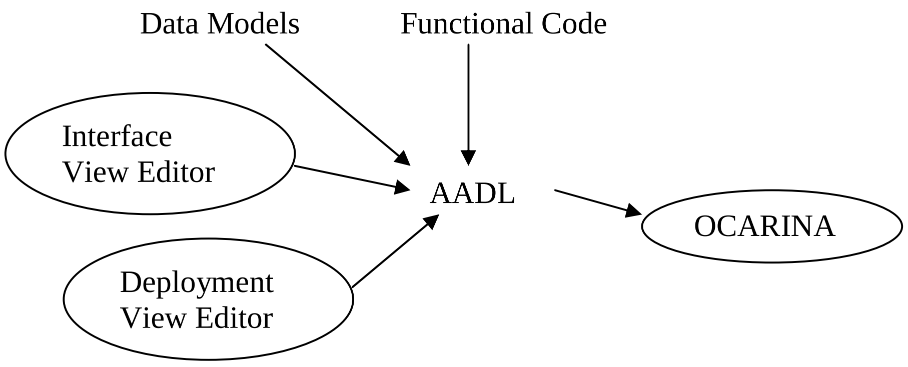


Figure 2 – Simplified view of the TASTE tool-chain

In this tool-chain [10], automatic code generation of the complete distributed system from a high level architecture is made feasible thanks to the early specification of the underlying execution model which is based on the Ravenscar Computation Model. The Interface View graphical editor is used to specify the logical interactions between the various functions and the Deployment View editor aims at describing the hardware architecture of the distributed system. A dedicated model transformation then creates a composite AADL representation of these various views while being compliant with the TASTE run-time specifications. Finally, this AADL model is processed by OCARINA.

2.3) AADL Real-Time analysis with Cheddar

This third example is the result of collaboration between the University of Brest and Ellidiss Technologies [11]. STOOD is a model driven design tool that allows for automatic generation of textual AADL specifications from a user friendly graphical editor. Cheddar is a real-time performance analysis tool that can read and interpret textual AADL files.

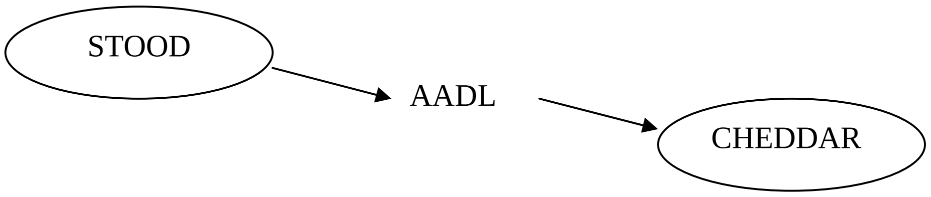


Figure 3 – AADL Real-Time analysis with Cheddar

Similarly to the two previous examples, AADL is used as a pivot language. Cheddar provides an implementation of the Real-Time Scheduling Theory, and can thus analyze the various concurrent architectures that can be described in AADL.

PRoposed Improvement

As shown with these examples of concrete tool-chains, emergence of well formalized Architecture Description Languages such as the AADL has significantly reduced the issue number 1 mentioned at the beginning of this article. Moreover, model transformation technologies are becoming more and more mature, which also reduces the effects of the issue number 2.

However, the issue number 3 remains a real restriction for the practical use of verification approaches in model driven engineering development processes. In the three cases presented above, each model that is used for the analysis activity (Time Transition Systems, Ravenscar Computation Model, Real-Time Scheduling Theory) addresses a precise subset of what can be described by an AADL architecture. Optimal use of such approaches would thus require the designer to perfectly master the semantic compliancy between the applicative model and the chosen analysis technique. This is not realistic, that’s why we propose below an approach that aims at improving the efficiency of model analysis activities.

In the following sections, only the third case will be considered (Real-Time Scheduling Theory), although a similar approach could be investigated for all the other analysis techniques.

3) A set of AADL Real-time design-patterns

In [6], we have proposed an approach based on design-patterns in order to ease usability of the real-time scheduling theory. To enforce issue number 3, we have defined four design-patterns called «Synchronous data flow», «Ravenscar», «Blackboard» and «Queued buffer». These design-patterns represent usual communication paradigms of multitasked real-time software. For each design-pattern, the set of feasibility tests that can be computed to perform the verification of the corresponding AADL architecture have been identified.
The four AADL design-patterns are:

- Synchronous data-flows design-pattern: this first design-pattern is the simplest one. The data sharing is achieved by a clock synchronization of the threads as Meta-H [2] proposed it. In this synchronization schema, thread dispatch is not affected by the inter-thread communications that are expressed by pure data-flows. Each thread reads its input data ports at dispatch time and writes its output data ports at complete time. This design-pattern does not require the use of a shared data component. In this simple case, the execution platform consists in one processor running a scheduler such as Rate Monotonic [1].
- Ravenscar design-pattern: main drawback of the previous 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 of Ada 2005 [3]. In Ravenscar, threads access shared data components asynchronously according to priority inheritance protocols.
- Blackboard design-pattern: Ravenscar allows a thread to allocate/release several shared resources (eg. AADL data). Real-time scheduling theory usually models such a shared resource as a semaphore, to represent, for example, a critical section. In classical operating systems, there exist many synchronization design-patterns such as critical section, barrier, readers-writers, private semaphore, and various producers-consumers [17]. 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 stored data, as opposed to the readers which are allowed to read the data simultaneously. The usual implementation of this protocol implies that readers and writers do not perform the same semaphore access that requires extra analysis.
- 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 executives provide communication features which allow storing all written messages in a memory unit. AADL also proposes such a feature with event data ports or shared data components.

For each pattern, an applicative test case was described under the form of an AADL model which has been formatted in purpose to highlight some of the possible performance analysis that Cheddar is able to automatically compute (thread worst case response time, bound on shared resource blocking time, memory footprint analysis, ...) [11].
However, this approach shows two weaknesses. First, it assumes that the designer is able to check that his AADL architecture is compliant with the set of pre-defined design-patterns. Second, for a given AADL design-pattern, several feasibility tests may be applicable. For example, in the case of the «Synchronous data flow» design-pattern, we have listed 126 possible variants for which several feasibility tests can be applied. This implies that only defining a set of design-patterns may not be enough to really help the designer.

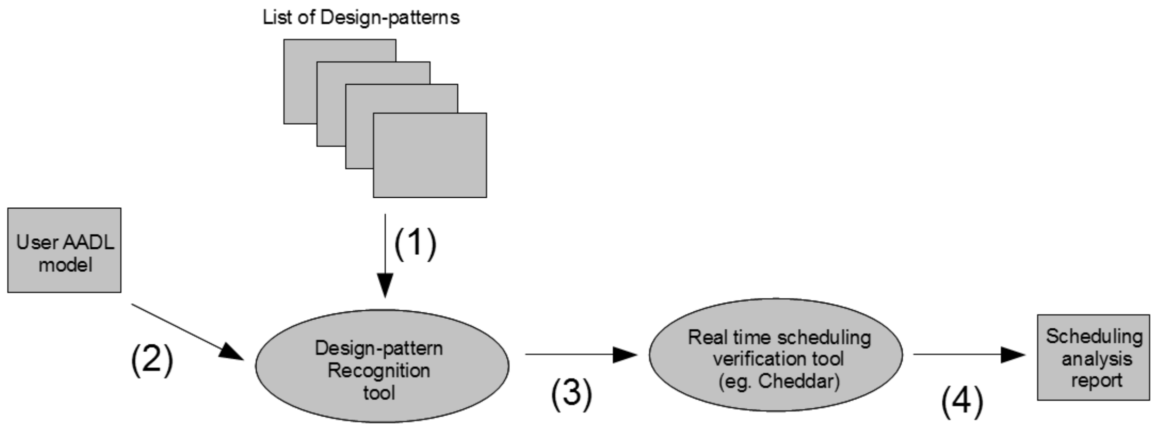

List of Design-patterns
(1)
User AADL model
(2)
Design-pattern Recognition tool
(3)
Real time scheduling verification tool (eg. Cheddar)
(4)
Scheduling analysis report

Figure 4- This Process proposed for the analysis of an AADL model

We then propose a process that allows the user to perform scheduling verification of his models. This process is depicted by figure 4:

1) A set of pre-defined design-patterns for which real-time scheduling analysis can be performed are identified (e.g. design-patterns defined above).
2) AADL models that are expected to be analyzed are specified by the end user.
3) Compliancy of the user AADL model with one or several design-patterns is checked by a design-pattern recognition tool.
4) Then, proper scheduling analysis can be performed on the compliant subset of the original specification.

In the sequel, we present how the design-patterns recognition tool is implemented.

4) About The implementation of the recognition tool

As the set of design-patterns may vary according to the application domain, a flexible model driven engineering process is being applied to automatically generate the recognition tool.
For such a purpose, we are using Platypus [8]. Platypus [http://cassoulet.univ-brest.fr/mme] is a software engineering tool which embeds a modeling environment based on the STEP standard [12,13]. First of all, Platypus is a STEP environment, allowing data modeling with the EXPRESS language [12,13] and the implementation of STEP exchange components automatically generated from EXPRESS models. Platypus includes an EXPRESS editor and checker as well as a STEP file reader, writer and checker.
In Platypus, a meta-model consists in a set of EXPRESS schemas. The main components of the meta-model are types and entities. From an EXPRESS schema and a data set made of instances of entities described by the EXPRESS schema, Platypus is able to check the data set conformity by evaluating the constraint rules specified in the EXPRESS model.

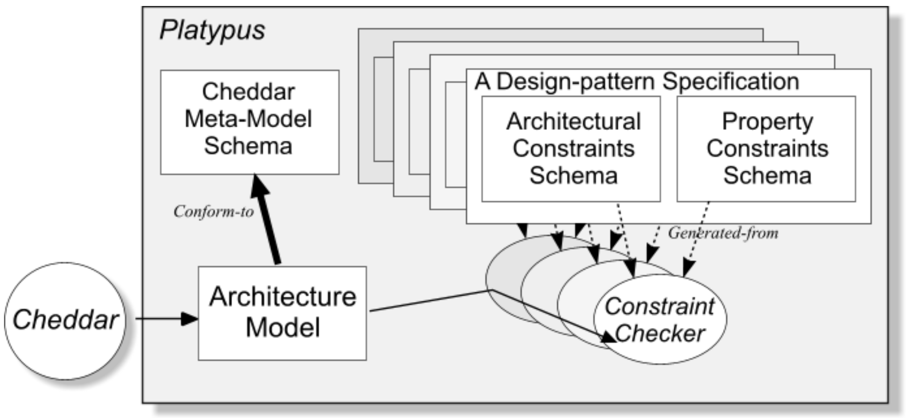

Platypus
Cheddar Meta-Model Schema
Conform-to
Architecture Model
Cheddar
A Design-pattern Specification
Architectural Constraints Schema
Property Constraints Schema
Generated-from
Constraint Checker

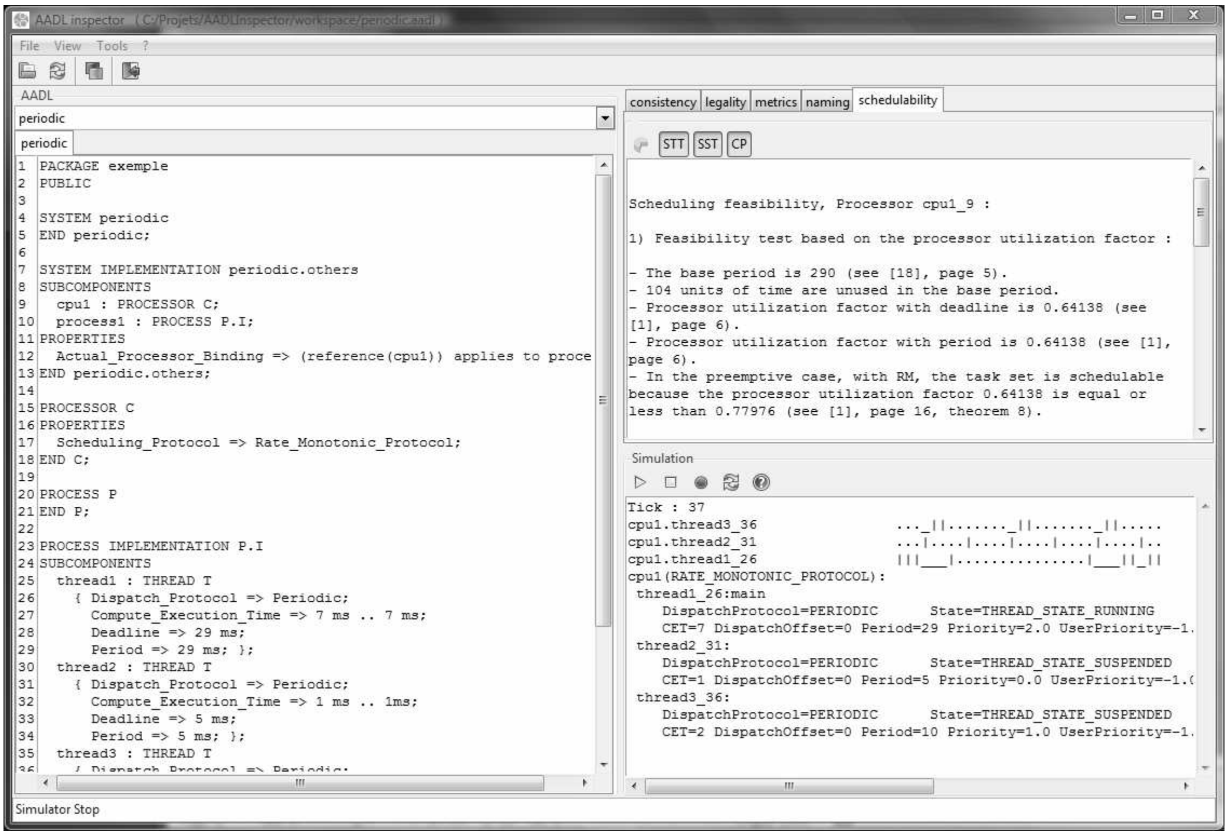
Figure 5- The recognition tool prototype within Platypus

Thus, given that the design-patterns are specified with EXPRESS, the decision tool prototype directly benefits from the Platypus STEP generic framework.
Figure 5 shows the prototype components and the data flow when an architecture model is analyzed. The prototype is first made of the shared meta-model named Cheddar meta-model schema. This meta-model specifies the internal Cheddar representation of a real-time architecture to verify. Then, each design-pattern is composed of two models which are defined in order to further constraint the Cheddar meta-model. These models correspond to two kinds of constraints: constraints on the entities of the architecture (called architectural constraints) and constraints on properties of entities (called property constraints). In the figure 5, they are specified by the architectural constraints and property constraints schemas. The constraints of these two models represent the applicability constraints of the feasibility tests assigned to the design-pattern.
Due to the current usage of the prototype, in order to be analyzed, an AADL architecture model must be encoded as an XML or a STEP data exchange file conforming to the Cheddar meta-model. Cheddar can be used for that purpose. Then, each design-pattern is evaluated separately. Evaluating a design-pattern consists in interpreting all rules specified in the architectural constraints and property constraints schemas.
We are currently elaborating EXPRESS models for the work presented above. It is very efficient to be able to directly test them from the Platypus modeling environment itself. But having to use Cheddar together with Platypus is not comfortable for end-users and Platypus, as a STEP based data modeling environment, is not user friendly enough. This is why we are currently re-implementing this analysis tool in Ada. This first version is hand-made but, for subsequent versions, we plan to use a model driven engineering process in order to automatically generate the recognition tool in Ada, the implementation language of Cheddar. Today, Cheddar is already partly automatically generated by Platypus. The Cheddar meta-model schema is used in order to produce the core components of Cheddar [14,15]. Implementing source code generation of the recognition tool only implies to update code generators of the core components of Cheddar.

5) Intended industrial use

Emergence of architecture languages like the AADL has greatly improved the strictness of the design phase of real-time systems and software and has led to the development of advanced modelling tools. At the same time, the real-time scheduling theory has also been subject to various implementation works. However, although several connections between design and analysis AADL tools are currently available, the experience shows a quite low level of analysability of these architectural models. A common solution to increase the level of analysability consists in restricting a priori the available set of modelling constructs. On the contrary, the approach presented in this paper can be applied *a posteriori* on non restricted real-time architectures. The pattern recognition process presented in this paper could be included into existing analysis tools such as AADL

Inspector developed by Ellidiss Technologies as an outcome of the long term investment of the company in the AADL standardization process.


AADL inspector - C:/Projets/AADLInspector/workspace/periodic.aadl
File View Tools ?
AADL
periodic
consistency legality metrics naming schedulability
STT SST CP
Simulation
Simulator Stop

```
PACKAGE exemple
PUBLIC

SYSTEM periodic
END periodic;

SYSTEM IMPLEMENTATION periodic.others
SUBCOMPONENTS
  cpu1 : PROCESSOR C;
  process1 : PROCESS P.I;
PROPERTIES
  Actual_Processor_Binding => (reference(cpu1)) applies to proce
END periodic.others;

PROCESSOR C
PROPERTIES
  Scheduling_Protocol => Rate_Monotonic_Protocol;
END C;

PROCESS P
END P;

PROCESS IMPLEMENTATION P.I
SUBCOMPONENTS
  thread1 : THREAD T
    { Dispatch_Protocol => Periodic;
      Compute_Execution_Time => 7 ms .. 7 ms;
      Deadline => 29 ms;
      Period => 29 ms; };
  thread2 : THREAD T
    { Dispatch_Protocol => Periodic;
      Compute_Execution_Time => 1 ms .. 1ms;
      Deadline => 5 ms;
      Period => 5 ms; };
  thread3 : THREAD T
    { Dispatch_Protocol => Periodic;
```

```
Scheduling feasibility, Processor cpu1_9 :

1) Feasibility test based on the processor utilization factor :

- The base period is 290 (see [18], page 5).
- 104 units of time are unused in the base period.
- Processor utilization factor with deadline is 0.64138 (see
[1], page 6).
- Processor utilization factor with period is 0.64138 (see [1],
page 6).
- In the preemptive case, with RM, the task set is schedulable
because the processor utilization factor 0.64138 is equal or
less than 0.77976 (see [1], page 16, theorem 8).
```

```
Tick : 37
cpu1.thread3_36                ..._||......._||......._||.....
cpu1.thread2_31                ...|....|....|....|....|....|..
cpu1.thread1_26                |||___|...............|___||_||
cpu1(RATE_MONOTONIC_PROTOCOL):
 thread1_26:main
    DispatchProtocol=PERIODIC      State=THREAD_STATE_RUNNING
    CET=7 DispatchOffset=0 Period=29 Priority=2.0 UserPriority=-1.
 thread2_31:
    DispatchProtocol=PERIODIC      State=THREAD_STATE_SUSPENDED
    CET=1 DispatchOffset=0 Period=5 Priority=0.0 UserPriority=-1.(
 thread3_36:
    DispatchProtocol=PERIODIC      State=THREAD_STATE_SUSPENDED
    CET=2 DispatchOffset=0 Period=10 Priority=1.0 UserPriority=-1.
```

Figure 6- AADL Inspector

AADL Inspector is a new lightweight and standalone tool that can analyse multi-files standard AADL projects. It comes with a set of analysis plug-ins that can be extended with additional rules checkers and bridgers for remote verification tools. AADL Inspector complies with the SAE AS-5506A standard (AADL v2) and SAE AS-5506/2 annex document (Behavior Annex). For upwards compatibility, AADL v1 files are also accepted. AADL Inspector also includes two Real-Time analysis tools (Cheddar and Marzhin) and their respective AADL input bridger.
A concrete application of the work presented in this paper would be the introduction of a “pre-analysis” layer in AADL inspector, in order to provide interactive assistance to AADL designers for a practical and efficient use of the Real-Time Scheduling Theory. A similar approach could then be considered for adding other analysis approaches, such as the ones presented at the beginning of this article.

6) Conclusions

Verification of real-time systems with the real-time scheduling theory may be difficult to be performed by system designers. In this article, we investigate how to increase real-time scheduling theory usability with an approach based on design-patterns. We have defined a set of design-patterns that model different task synchronization and communication paradigms. When a designer wants to perform a performance analysis of an AADL model, he must check that his model is compliant with one of these design-patterns. If the model is actually compliant with a design-pattern then he can call Cheddar to automatically perform schedulability analysis. However, checking compliance of his models to the design-patterns may be difficult to achieve, especially if the designer is not an expert on real-time scheduling theory. To automatically check compliance,

we have proposed a recognition tool which relies on the Platypus environment. We are currently implementing this recognition tool and it will be distributed with the next release of Cheddar.
The work presented in this article leads to numerous future works. A first future work is related to the list of design-patterns that we have studied. Indeed, we only have investigated how to check compliance with the Synchronous data flows design-pattern. In the next months, we will do the same work for the other design-patterns: Ravenscar, Queued Buffer and BlackBoard. There are also numerous other synchronization and communication paradigms that we have to investigate [17]. Second, in this approach, we expect to verify if an AADL model is fully compliant with a set of design-patterns. But in some cases, architectural AADL models of practitioners may be compliant with none of the proposed design-patterns. Then, we plan to investigate how the designers can eventually be helped with a set of metrics [16]. These metrics should allow the designers to compare their AADL models with our design-patterns and to improve their models in order to be compliant with the real-time scheduling theory.

7) Acknowledgments

We would like to thank Conseil Régional de Bretagne and Ellidiss Technologies for their support to this project.

8) References

1. C. L. Liu et J. W. Layland : Scheduling algorithms for multiprogramming in a hard real-time environnment ; Journal of the Association for Computing Machinery, vol. 20, n° 1, pp. 46- 61, January, 1973.
2. S. Vestal. Meta-H User's Manual, Version 1.27, 1998, download at http://www.htc.honeywell.com/metah/uguide.pdf
3. ISO, Ada 2005 International Standard ISO/IEC ISO/IEC 8652:1995/Amd 1:2007, March 2007, Ada Working Group (WG9)
4. Sha, R. Rajkumar and J.P. Lehoczky. Priority Inheritance Protocols: An Approach to Real-time Synchronization. IEEE Transactions on computers, 39(9):1175-1185. 1990.
5. F. Cottet, J. Delacroix, C. Kaiser, Z. Mammeri. Scheduling in Real-Time Systems, John Wiley & Sons, Ltd, 2003.
6. F. Singhoff, A. Plantec, P. Dissaux, J. Legrand. Investigating the usability of real-time scheduling theory with the Cheddar project. Journal of Real Time Systems. Volume 43, number 3, pages 259-295. November 2009. Springer Verlag.
7. SAE AS-2C. Architecture Analysis and Design Language (AADL) AS 5506A, Aerospace Standard, Version 2, January 2009.
8. A. Plantec et V. Ribaud. PLATYPUS : A STEP-based Integration Framework. 14th Interdisciplinary Information Management Talks (IDIMT-2006). Budweis, République Tchèque. pp. 261-274. September 2006.
9. B. Bertomieu, J.-P. Bodeveix, S. Dal Zilio, P. Dissaux, M. Filali, P. Gaufillet, S. Heim, F. Vernadat. Formal Verification of AADL models with Fiacre and Tina. Proceedings of the 5th International Congress Embedded Real Time Software and Systems (ERTS 2010). Toulouse, France, 19-21 May 2010.
10. M. Perrotin, E. Conquet, P. Dissaux, T. Tsiodras, J. Hugues. The TASTE Toolset: turning human designed heterogeneous systems into computer built homogeneous software. Proceedings of the 5th International Congress Embedded Real-Time Software and Systems (ERTS 2010). Toulouse, France, 19-21 May 2010.
11. P. Dissaux, F. Singhoff. Stood and Cheddar: AADL as a pivot language for Analysing Performances of Real-Time Architectures. Proceeding of the 4th ERTS conference, February 2008. Toulouse, France.
12. ISO 10303-1. Part 1: Overview and fundamental principles, 1994.
13. ISO 10303-11. Part 11: EXPRESS Language Reference Manual, 1994
14. F. Singhoff and A. Plantec. Towards User-Level extensibility of an Ada library : an experiment with Cheddar. Proceedings of the 12th International Conference on Reliable Software Technologies, Ada-Europe,

Lecture Notes on Computer Science/LNCS, Springer-Verlag, Volume 4498, pages 180-191, Geneva, June 2007.

15. A. Plantec and F. Singhoff. Refactoring of an Ada 95 Library with a Meta CASE Tool. ACM SIGAda Ada Letters, volume 26, number 3, pages 61-70. ACM Press, New York, USA, November 2006, ISSN:1094-3641.
16. M. Monperrus, JM Jezequel, J. Champeau, and B. Hoeltzener, A Model-driven Measurement Approach. Proceedings of the ACM/IEEE 11th International Conference on Model Driven Engineering Languages and Systems (MODELS'2008).
17. A. Tanenbaum. Modern Operating Systems. Prentice-Hall. 2001.