How architecture description languages help schedulability analysis: a return of experience from the Cheddar project

Frank Singhoff[1], Alain Plantec[1], Stéphane Rubini[1], Vincent Gaudel[1], Shuai Li[1], Christian Fotsing[1], Laurent Lemarchand[1], Pierre Dissaux[2], Jérôme Legrand[3]

January 8, 2019

[1]Lab-STICC, UMR CNRS 6285, Université de Bretagne Occidentale, UEB, 20, av Le Gorgeu, Brest, France.
[2]Ellidiss Technologie, 24, quai de la douane, 29200 Brest, France.

Abstract

Architecture Description Languages (ADL) are languages that allow people to specify the design of a system. An ADL provides a mean to capture the architecture of a system and to reason about it. As in the case of critical real-time systems, most of ADLs model both the software part and the hardware part of the applications and their execution environment, reasoning about scheduling with an ADL is natural. We believe that using an ADL may greatly help to perform scheduling analysis of real-time critical systems. This article discusses the different strengths and weaknesses of ADLs in the context of scheduling analysis of real-time critical systems, with ADLs that are dedicated to such analysis but also with others that are international standards. This discussion is illustrated by the work we have made in the context of the Cheddar scheduling analysis tool.

Keyword: Architecture Description Language, Schedulability analysis, MDE, AADL, MARTE/UML

1 Introduction

Real-time systems are systems on which timing constraints are specified and which have to meet them. They are usually concurrent systems, e.g. systems made of tasks constrained by the system timing requirements.

Architecture Description Languages (or ADL in the sequel) allow designers to specify, formally or not, the design of real-time systems. Usually, ADLs provide the abstraction of components, connections and deployments [1] and allow the designer to specify timing constraints. A component is an entity modeling a part of the system. Many ADLs allow the specification of hardware parts and software parts of the system with dedicated kinds of components. Connections usually model relationships between components and finally, deployments specify how software components are deployed on hardware components, i.e. how resources of the system are allocated.

Real-time systems intrinsically involve various functional and non-functional points of view [2]. Functional point of view mainly focuses on tasks specification. Non functional point of view mainly focuses on timing constraints and on resource consumptions. Non functional point of view involves not only software but also hardware aspects. This explains why ADLs usually provide means to model both functional and non functional view points.

Problem statement In this article, we focus on real-time systems that are critical. A critical system is a system that must run safely in any situation. Critical real-time systems are real-time systems that require that all their timing constraints must absolutely be met, otherwise dramatic issues may be raised on the users or the systems themself (e.g. destruction of the system, loose of live, bankrupt, ...).

Building a critical real-time system is known to be a very tedious task because of the high level of security that the system must ensure, but also because of the increasing complexity of software requirements or the increasing capabilities and complexity of the hardware. While it is possible to use formal methods to mathematically prove requirements of a real-time system, their use remains an exception. Indeed, the use of formal methods relies on a high theoretical skill level.

In many cases, the quality of a system is a question of trust. Trust and development cost are then two important issues. Trust is defined as the level of security and dependability achieved by an implementation [3]. The highest possible level of trust must be acquired together with a fast time to market.

Designers must use several tools and languages in order to cover all aspects of those systems (e.g. functional, non functional, software and hardware aspects, ...). Moreover, each system turns to be very specific and for each particular system a need to develop and integrate specific tools may arise. Then, the problems addressed in this article are of several kinds:

- How to design functional requirements and specify their timing constraints?

- More specifically, how to early check timing constraints before implementing and deploying the target system?
- How to design and relate the hardware and the software parts of the system to achieve timing constraint verifications?
- And finally, how a particular timing constraints verification tool can be designed, implemented and integrated in a specific tools chain?

Contributions of the article Making easier the design and the implementation of real-time critical systems is one the main goal of the Cheddar project. During the last decade, in the context of this project, we have used the two currently available main ADLs which are AADL and MARTE. We have also designed and implemented our own tool called Cheddar which is mainly dedicated to task timing constraints early verification. Using Model Driven Engineering, we have also built specific tools and Cheddar itself, and integrated them within various tools chains.

This article presents a return of experience regarding real-time system implementation and verification. The contributions of this article covered several aspects of scheduling analysis. First, we introduce Cheddar ADL, an ADL dedicated to scheduling analysis. This ADL only focuses on scheduling point of view, but we have shown, thanks to various experimentations, that models written with standard ADLs such as AADL and MARTE/UML can be successfully transformed to Cheddar ADL models and can lead to scheduling analysis. Furthermore, to ease automatic scheduling analysis and then reduce analysis skills required from engineers, we have proposed several architecture design-patterns specified with Cheddar ADL. They are modeling classical synchronization and communication patterns used in real-time critical systems. We believe that these design-patterns may greatly help toolset designers to enforce interoperability between their tools (e.g. between modeling tools and analysis tools). Finally, Cheddar ADL is provided with various tools generated from Cheddar ADL meta-models and a model-driven process. This allows Cheddar ADL designers and users to adapt their scheduling analysis tool to their needs.

In the next sections of this article, we illustrate the ability of ADLs to achieve scheduling analysis by a presentation of the works we have done in the context of the Cheddar project.

In this project, we have investigated schedulability analysis with different ADLs. Then, in section 2, two major standard ADLs are presented: AADL and MARTE. We expose their features and we also discuss their ability to achieve scheduling analysis.

Section 3 introduces Cheddar and its internal ADL: Cheddar ADL.

Section 4 shows how a dedicated ADL may help to produce and maintain scheduling analysis tools. In this section, we describe the various model-driven processes we have experimented in the context of Cheddar and in the context of scheduling analysis. They show how an ADL may allow tool designers to automatically produce scheduling analysis tools. We explain how to enforce

schedulability analysis of ADL models. We also discuss how to integrate analysis tools in software engineering tool-chains. We illustrate how an ADL may help to ensure tool interoperability.

Section 5 is devoted to related works.

Finally, in the last section, we discuss the different strengths and weaknesses of ADLs to perform scheduling analysis before concluding and presenting future works.

2 Architecture Description Languages

Architecture Description Languages usually provide the abstraction of components, connections and deployments [1]. A component is an entity modeling a part of the system. Many ADLs allow the specification of hardware parts and software parts of the system with dedicated kinds of components. Connections usually model relationships between components and finally, deployments specify how software components are deployed on hardware components, i.e. how resources of the system are allocated.

The two main purposes of ADLs are to provide a mean to capture the architecture elements of a system and to help designers to make various analysis about it, including scheduling analysis. Using an ADL may greatly help to perform scheduling analysis for various reasons.

Embedded systems, like avionic or automotive systems, are complex ones. Not only both the software and hardware architectures may be tricky during the initial design phase, but engineers must also compose with evolvable systems. Then, scheduling analysis that the ADL models support must include various aspects of the system to be accurate.

A first aspect is the representation of the different layers of the system, roughly speaking the application, the operating system and the hardware layers. Modeling hardware layer is mandatory because embedded systems inherently supply restricted resources, and sometimes dedicated to specific operations. Moreover, some systems rely on a large set of processors to achieve their computation needs; resource utilization conflicts may also appear at different points of the hardware architecture, and may impact the real execution time. The performance of an Operating System (OS) has a direct impact on the performance of the application. Generally an application is implemented with an API offered by the OS. The API facilitates programming but hides system operations that have a certain cost. For example, tasks may lock shared resources implemented with a certain API for resources, but the operation of locking a resource takes time by the OS and needs to be considered for fine-grain analysis. Furthermore performance of entities in the OS also has an impact on application performance. For instance, the time taken by the scheduler in the OS to switch memory contexts, when task preemption occurs, may be long enough to be considered. Coarse-grain OS clocks may also introduce clock jitters to the analysis.

A second aspect is the type of data an ADL allows to gather in the same

set of models. For example, an ADL may be useful to gather all data that scheduling analysis requires: data modeling of the application to verify and its timing parameters, but also data modeling of the timing characteristics of the execution environment (both hardware and software). Another example is the set of data that allow tool-chain implementation. Tool-chains are crucial in order to make scheduling analysis as automatic as possible. The ADL is then used as a pivot language in defining both the syntax and the semantic of the models shared by the different tools.

Finally, another aspect is the level of details of the models written with a given ADL. Depending on how the behavior of components and their interactions are described, scheduling analysis can be run with different analysis methods and can lead to various analysis result quality. For example, from the architecture of the system, attributes or properties may or must be appended to allow some kinds of analysis methods to be performed: non-functional data like execution time, execution flow or reference to program's codes may complete the component models, and may be useful as an entry for scheduling analysis tools.

As we have seen, the definition of a general ADL language is complex, because it needs to cover several requirements. Various ADLs have been proposed in the context of real-time systems. UML-MARTE [4], EAST ADL [5], Fractal [6] or AADL [7] are well-known.

Among these modeling languages, AADL and MARTE are considered as the two mainstream ADLs in the critical real-time domain. In the context of our experiments, we have mainly used AADL. We have also used UML MARTE but more especially for the operating system part. In the reminder of this section we briefly describe these two ADLs.

2.1 The Architecture Analysis & Design Language

AADL (namely Architecture Analysis Design Language) is a domain specific language for the design, the analysis and the integration of distributed real-time systems. AADL is a SAE standard (AS-5506), first published in 2004 [7]. This language supports the specification of systems as an assembly of hardware and software components. AADL models integrate functional component interfaces interconnected to non-functional aspects (response time, safety and certification properties, among many others). The interfaces are later refined as component implementations. Feiler et al. identify two ways to use AADL for analysis [8]: (1) lightweight analysis of architecture patterns to discover systemic problems and (2) full scale analysis based on theoretical frameworks like sensitivity analysis, schedulability or reliability analyses, code generation, etc.

An AADL model describes either textually or graphically, a system as a hierarchy of components with their interfaces and their connections. It is made of several categories of software and hardware components. (1) *Software component* categories are threads, processes and sub-programs. A thread component models a schedulable unit of concurrent execution. Process components represent virtual address spaces. A subprogram component models a piece of program

that is sequentially run. (2) *Hardware components* specify the execution platform. AADL defines processor, memory, device and bus categories. Hardware and software components can be stored in libraries or hierarchically organized in systems. An AADL system component specifies how software components are deployed on hardware components. The execution of a software task may be assigned to one or a set of components of the execution platform.

AADL components interact through connections. A connection models data or control flows between components. Connections can model exchange of events or queued messages, rendez-vous or accesses to shared data.

AADL model components can be enriched by properties. As an example, a particular deployment of a software application on an execution platform is specified through *binding* properties. A property makes explicit various concepts such as the periodicity of a thread, the name of the source code for a sub-program component, or the available bandwidth for a bus component. Properties are defined in property sets. The AADL standard includes a large set of normalized properties grouped in several standard property sets. Designers can also define their own properties. These kind of application or system specific properties may be added to extend the description with regard to the expected system analysis.

Any AADL model must be at least compliant with the core AADL language. In addition, a model may embed elements defined in annexes. An annex is a kind of extension. An annex can formulate particular design patterns for particular domains: data modeling, ARINC653 systems, ... An annex can also specify a companion language dedicated to a particular aspect. Currently two standard annexes are available: the error and the behavior annexes. Annex elements may be ignored by tools if they are outside of the considered analysis scope.

Model example To illustrate AADL, we present in this section an AADL model for an Attitude and Orbital Control System (AOCS) of a spacecraft case study [9, 10]. An AOCS maintains the spacecraft orbit and ensures the spacecraft is oriented to achieve the expected functionality. This subsystem consists of set of redundant sensors and actuators such as sun/star and earth sensors, gyroscopes, momentum wheels, reaction wheels, magnetic torquers, thrusters, and solar array and trim tab positioners.

Listing 1 is a simple AADL model for this system. It represents its software architecture and its hardware execution platform. This AOCS is composed of several tasks modeled by AADL threads:

- **Command Actuators Task** — applies a set of commands defined by the Control Law task to keep the spacecraft in its orientation and pointing.
- **Control Law Task** — implements the basic control laws and is therefore responsible for maintaining the spacecraft orientation to a defined point.
- **RW Data Task** — the Reaction Wheels (RW) actuator controls the movement of the spacecraft.

- **DSS Data Task** — the Digital Sun Sensor (DSS) keeps track of the spacecraft's orientation in relation to the position of the sun.
- **Gyro Data Task** — the Rate Gyro Sensor detects the rotation of the spacecraft; this sensor data is sporadic because normally it has a different clock rate so the data can contain some jitter.
- **IRES Data Task** — the InfraRed Earth Sensor (IRES) scans a large field of view and then detect signals at the Earth/Space transitions.

Listing 1 presents three of these threads (`RW_Data`, `DSS_Data` and `Gyro_Data`). The three threads are either periodic or sporadic. All threads are deployed on a multi-processor environment. Each thread has its own properties made of a dispatch protocol, a period and a computing execution time. The scheduling policy is a preemptive fixed priority scheduling.

```
1  system Product; end Product;
2  system implementation Product.impl
3  subcomponents
4     Hard : system Soc.Soc_Leon4;
5     Soft : process AOCS.impl;
6     ...
7  properties
8     Actual_Processor_Binding => (reference(Hard.Proc_Bus.Core1)
9           applied to Soft.RW_Data;
10    Actual_Processor_Binding => (reference(Hard.Proc_Bus.Core1)
11          applied to Soft.DSS_Data;
12    Actual_Processor_Binding => (reference(Hard.Proc_Bus.Core2)
13          applied to Soft.Gyro_Data;
14    Scheduling_Protocol => (POSIX_1003_Highest_Priority_First_Protocol)
15       applies to Hard.Core1, Hard.Core2;
16 end Product.impl;
17
18 process AOCS end AOCS;
19 process implementation AOCS.impl
20 subcomponents
21    RW_Data : thread periodic_aocs {
22       Dispatch_Protocol => Periodic;
23       Period => 200 ms;
24       Compute_Execution_Time => 1 ms .. 2 ms;
25       Deadline => 200 ms;
26       Priority => 10;
27    };
28    Gyro_Data : thread sporadic_aocs { ... };
29    DSS_Data : thread periodic_aocs { ... };
30 end AOCS.impl;
```

Figure 1: Modeling a system with AADL.

Details about the execution environment are issued from the AADL model of the sub-system instance `hard`. The execution platform is a System on Chip

(SoC) from Aeroflex Gaisler [11]. Three buses structure the SoC: (1) an ARM AHB (Advanced High-performance Bus) acting as a processor bus, (2) an ARM AHB that links fast IO devices, and (3) an ARM APB (Advanced Peripheral Bus) that connects other "slow" peripheral devices. Here, we only focus on the processor bus sub-system; the Figure 2 models the components sharing this bus.

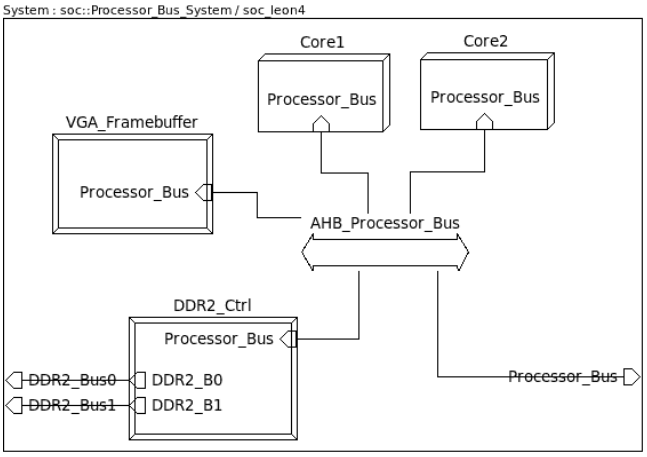


Figure 2: The processor bus sub-system, represented by the AADL graphical notation. Two LEON4 cores and a video frame buffer device share the bus to access memories through a DDR2 memory controller device.

The next AADL textual model gives details about the implementation of a processing core in the SoC. A property set extends the AADL standard properties to precisely describe the cache memory features.

Usually, the AADL models do not explicitly represent the operating system services; the AADL "hardware components" are abstraction of an execution platform implementing a service by the mean of hardware and software parts. For instance, the AADL standard defines a processor component like "an abstraction of hardware and software that is responsible for scheduling and executing threads". This semantic explains why, in the figure 1 (line 14), the specification of the scheduling protocol is applied on the processor instances.

2.2 UML MARTE

In the previous sections, we have shown how to model both the hardware parts and the software parts of a system with AADL in the perspective of scheduling analysis. Many concurrent real-time softwares access to the hardware resource through a Real-Time Operating System (RTOS). More specifically the software's development is based on the RTOS's Application Programming Interfaces (API). With its hardware components, AADL defines an execution platform that hides RTOS layer. In this section, we address the modeling of a

```
with cache_property_set;
processor Processor_Core
features
    Processor_Bus : requires bus access;
end Processor_Core;
processor implementation Processor_Core.Leon4
subcomponents
    L1_Inst_Cache : memory Cache.impl { ... };
    L1_Data_Cache : memory Cache.impl {
      Byte_Count => 4 KB;
      cache_property_set::Line_Size => 32;
      cache_property_set::Cache_Type => Data_Cache;
      cache_property_set::Associativity => Set_Associative;
      cache_property_set::Set_Size => 2;
    };
end Processor_Core.Leon4;
```

Figure 3: Modeling a cache with AADL.

real-time system in the perspective of scheduling analysis with MARTE-UML and we also show how an ADL as MARTE-UML allows an explicit modeling of the RTOS layer.

Modeling and Analysis of Real-Time Embedded systems (MARTE) [12] is a standard UML profile promoted by the Object Management Group [13]. The profile adds capabilities to UML for Model-Driven Development [14] of real-time systems. MARTE thus provides support to specify and to design such a system but also to annotate the model for different kinds of analysis. The MARTE package structure is based on UML version 2 [15]. More specifically, we focus on the "stereotype" and "tagged value" concepts in UML profiles. To connect the profiles with ADLs, we will assume that "stereotypes" are entities in an ADL and "tagged values" are an entity's attributes.

MARTE was designed to cover a large area of real-time systems, including avionic and automotive. For example, the profile was designed so that all AADL concepts can be modeled in MARTE (see [12], Annex A, section 2.3). The profile was also designed to support tools dedicated to real-time systems (e.g. modeler, code generator, functional analyzer, simulator).

The MARTE profile is composed of several sub-profiles to represent different concepts of a real-time system. For instance, the Hardware Resource Modeling (HRM) sub-profile is used to model hardware entities, the High Level Application Model (HLAM) to represent a set of functional entities...

Here, we focuses on the GRM and SRM sub-profiles that allow us to model an operating system's entities and services. For generic platform modeling, MARTE packages the General Resource Modeling (GRM) profile. This profile provides the foundations needed for a more refined modeling (e.g. specific software model). The Software Resource Modeling (SRM) profile offers stereotypes to model the entities provided by a RTOS API and their properties. SRM has

been built upon observing entities present in several standard or proprietary RTOS APIs such as POSIX, OSEK/VDX, ARINC 653, VxWorks, RTAI or QNX.

Model example Figure 4 shows the model of the AOCS case-study in MARTE: the three tasks called *RW_Data*, *DSS_Data*, and *Gyro_Data* are allocated on the *AOCS* address space. The address space is modeled as a UML component stereotyped <<MemoryPartition>>, while tasks are modeled as properties of the component stereotyped <<SwSchedulableResource>>. Since tasks are modeled as properties of the AOCS component stereotyped <<MemoryPartition>>, this means they are allocated on the AOCS address space. Tasks are scheduled by a fixed priority scheduler.

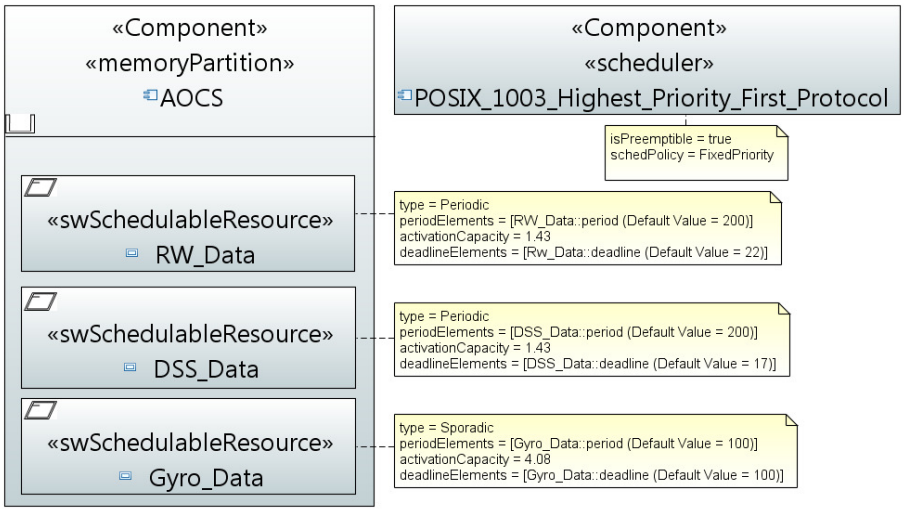


Figure 4: Tasks and Address Space Modeled in MARTE: Comments show stereotype tagged values. Dashed lines show links between comments and UML element.

During their execution, the tasks ask to lock a mutex to access some critical sections. The mutex is implemented using a mechanism offered by the RTOS API. In the case of POSIX, the mutex mechanism would be modeled with the class $Mutex\_t$ shown in Figure 5.

To model the actual mutex that is locked by tasks, a UML property is typed by the $Mutex\_t$ class. This UML property is also stereotyped <<SwMutualExclusionResource>>. This is shown in Figure 6 by the SHM_Mutex property, typed $Mutex\_t$, of the AOCS component. Note that this means that shared resource protected by SHM_Mutex is allocated on the AOCS address space.

To model a task locking SHM_Mutex (and thus accessing a critical section) a usage in UML is specified between the task and SHM_Mutex. The usage

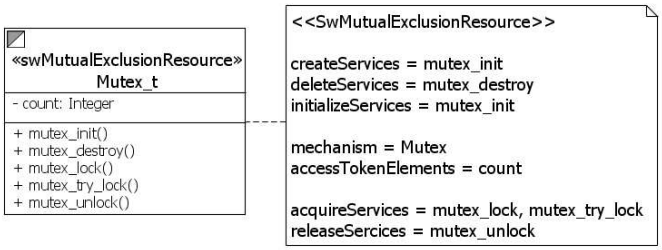


Figure 5: POSIX Mutex Modeled in MARTE: Comments show stereotype tagged values. Dashed lines show links between comments and UML element.

is stereotyped <<ResourceUsage>> in MARTE. The *execTime* attribute of <<ResourceUsage>> defines the length of the critical section. The start of the critical section in a task's execution is modeled by a UML constraint that constrains the usage. The constraint is stereotyped <<TimedConstraint>> in MARTE. The interpretation attribute of <<TimedConstraint>> is set to instant and the specification of the constraint is a LiteralReal in UML, whose value defines the start of the critical section in a task's execution. For example in Figure 6, task RW_Data locks SHM_Mutex at 0.2 of its execution time, and locks it for the next 0.13 of its execution time.

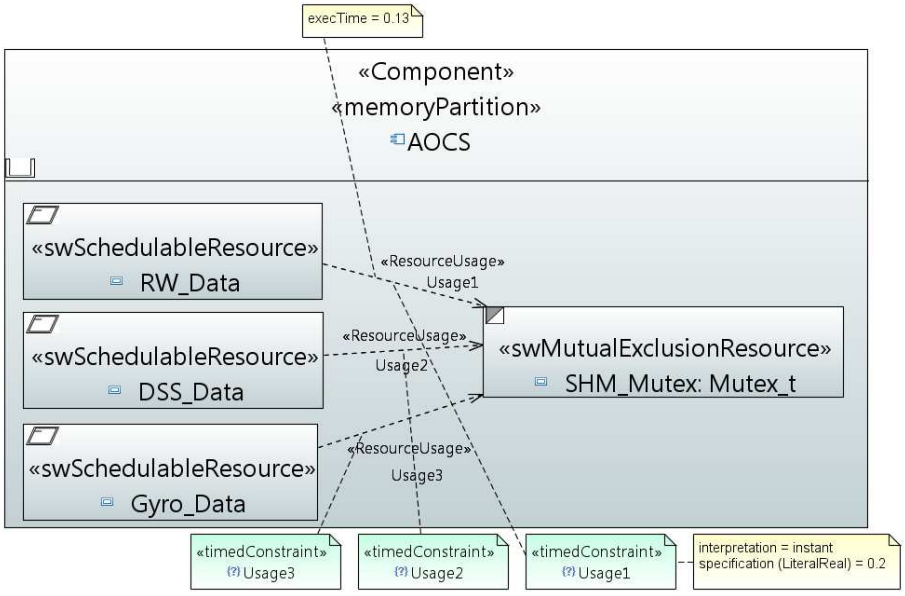


Figure 6: Mutex to Protect SHM, Modeled in MARTE: Comments show stereotype tagged values. Dashed lines show links between comments and UML element.

2.3 Tools support with AADL and MARTE

Several AADL and MARTE scheduling analysis tools exist. Designers usually need to tailor tool-chains for the construction of critical real-time systems by integrating scheduling analysis tools. In these tool-chains, the ADL may be used as a pivot language to store all architectural artifacts. The goal of this section is to compare the two different approaches of AADL and MARTE regarding tools support in general and also regarding scheduling analysis tools in particular.

AADL AADL have been designed to cover as many aspects of embedded real-time distributed systems as possible. For example, this language provides features for most of the concurrency mechanisms found in real-time systems (e.g. asynchronous or synchronous communication/synchronization, event or time driven communication/synchronization, ...). Each element of the language is described by the standard with its own semantic and its own set of validity rules. In this respect, among the available annexes, the error annex constitutes one the most fundamental part of the standard. The stability of the language and the well defined semantic bring to the community the opportunity to share a precise common knowledge and especially to design a set of common tools.

AADL models can be presented either graphically or textually. Thus, several levels of abstraction can be managed by tools. Typically, high level architectures can be designed graphically and later refined at a lower level of abstraction by using the textual syntax. As an example, the standard behavior annex can be used to precise some dynamic aspects of an architecture.

At the tool-chain level, interoperability issues may arise especially because specific aspects can be added through annexes. Fortunately, the goal of the standard is precisely to minimize this kind of issue.

However, being a stable and a general purpose standard modeling language has counter parts. We consider the following weak points as important issues regarding tools support for AADL:

- *Complexity of tools implementation:* AADL tools have to deal with a lot of concepts and with numerous interdependent aspects. As an illustration, the full AADL meta-model has more than 100 core elements. The AADL syntax has 185 rules and the core language standard describes around 250 legacy rules and more than 500 semantic rules. Furthermore, AADL designers have to cope with several levels of specifications (e.g. types and instances of components, interfaces and implementations of components).

- *Evolutivity and adaptability of tools:* As an international SAE standard, AADL benefits from a normalization process, organized around international meetings during which each decision to evolve the standard results from a consensus between official committee members. The counterpart is that the standard can not evolve rapidly. As hardware physical and logical characteristics evolve and as functional requirements complexity continuously increases too, the stability of AADL can be an issue when designers

have to describe systems that involve new and emerging technologies or specific functional requirements.

MARTE The MARTE and AADL languages are based on quite different philosophies. MARTE constitutes a common basis for a family of design languages. With MARTE, each design model potentially uses its own design language (i.e. UML profiles). Regarding tools support, because MARTE is UML based, users benefit from existing design environments. For a specific architecture, the idea behind MARTE is to make it possible for the user to design its own profiles and tools. Again, for that purpose, powerful UML based development environments exist, some of them are freely available and used by an important and active community. This can greatly facilitate the design of specific profile elements and the implementation of related tools as editors, checkers or code generators. In fact, being UML based gives *de facto* to this language a high level of popularity.

Designing an architecture with MARTE also implies to make choices between the available elements of the language or even to specify and implement its own elements with new profiles. Then, designers and tools developers have to deal with a small and known set of concepts that are very specifically useful for a particular real-time architecture.

Thus, the flexibility of MARTE makes it adaptable and usable to build tools for many architecture kinds comprising emerging or very specific systems. However, from the semantic of profile elements and the composition of profiles point of view, its flexibility can turn as a major weak point regarding tools support:

- *Tools interoperability:* by its nature, a profile is made to be customized by additional elements. An important drawback of this approach is that the semantic of new elements can be let implicit and implementation dependent. Then several tools can use the same profile but with different semantics. This can make difficult to integrate several tools within the same tool-chain.

- *Semantic coherence:* Defining a coherent semantic of a single element and more, of a set of related and reusable language elements can be a difficult task. In this case, the problem for tools is that they cannot rely on clearly defined building blocks.

- *Profiles reusability and composability:* As several profiles can be specified independently in different contexts, it may be difficult to compose and reuse profile elements. This problem can push designers to always implement new specific profiles instead of reusing and composing elements of existing profiles. Thus, this problem may increase the tools interoperability issue. This may also increase tool implementation costs.

2.4 Tools interoperability issue with AADL and MARTE

AADL can be considered as a monolithic and stable ADL with a semantic which is very precisely defined. The benefit is a standard ADL which favors tools interoperability at the price of less flexibility. However, AADL supports multiple modeling styles and allows the use of specific property sets. The behavior part of a system can be specified either with the help of properties, with connections between elements or with binding relations allocating execution resources to components or also with annexes. Thus, a particular AADL tool or tool-chains can impose to the designer the use of a very specific modeling style, its own set of properties and actually, its own subset of the AADL language. The consequence is that compatibility issues can be raised between AADL models and tools.

As an UML profile, MARTE favors language extensions with application specific elements. As it is explained in section 2.3, the counter part of MARTE flexibility is an increased difficulty regarding tools interoperability.

2.5 Can we use MARTE or AADL models to drive scheduling analysis?

The information contained in an ADL model may drive different kinds of analysis or transformation processes. Especially, the Cheddar project focuses on the assessment of system's timing constraints by applying several methods from the scheduling analysis domain. MARTE and AADL languages are both able to express the required timing properties and the structural data to parametrize the scheduling analysis.

MARTE The Generic Quantitative Analysis Modeling (GQAM) profile lets the user create an analysis model. The analysis model contains any extra annotations needed for analysis (e.g. the correctness criterion). However, the MARTE approach is not to create new analysis methods, but to support existing ones instead. Separating the analysis model from the design model makes it possible to aggregate the latter with more information, without modifying it. On the other hand MARTE does not make this separation mandatory in order to not add modeling efforts when a model is already analyzable. GQAM provides a framework that can be used to create new profiles for a specific kind of analysis. Schedulability Analysis Modeling (SAM) is an example of a profile based on GQAM for timing analysis.

AADL The AADL standard includes pre-defined property sets as the property set *timings* that defines the useful parameters needed to perform a scheduling analysis. Some values of these properties are left outside of the standard. As an example, the potential scheduling algorithms available for a given target are specified by the user in the property set `AADL_project`. One of the property of `AADL_project` enumerates the list of supported scheduling algorithms (called a "protocol" in AADL) on the target. Once again, the ADL allows the designer to create an analysis model but does not specify directly how the analysis is done.

Beyond the timing properties, the model must describe the execution platform in order to:

- Identify all the computing resources which may support *a priori* the parallel execution of the software tasks;
- Identify the access contentions of the shared resources which limit the effective task parallelism that the system can accept.

We already have shown with the example of the section 2.1 that AADL enables detailed modeling of hardware components, and hence satisfies the above requirements. Detailed hardware modeling with MARTE UML can be found in [16].

Do we need a specific ADL for scheduling analysis? As stated before, both AADL and MARTE aspire to cover a broad range of features on the real-time domain. However, when focusing on a particular type of analysis, like scheduling analysis, using ADLs with broad range of features may rise some issues for the designers.

A first issue is the abstraction level chosen for describing the system. According to the goal assigned to the model usage, the level of details or the properties to define are not the same. For instance, if we plan to generate a part of program's codes, attention should be paid on the code structures and function interfaces, where as only data about the program execution time are necessary to drive scheduling analysis. In the opposite, a precise model of the hardware architecture, including "secondary" parts like communication buses or memory sub-systems, might be meaningless for code generation, but are mandatory in scheduling analysis. The architecture designers must have high level skills in the scheduling domain to select the relevant information to express in the model.

Another issue is the evolving capabilities that we have already mentioned in section 2.3. As the scheduling community continuously produces new task models and feasibility tests, scheduling analysis tools may be updated frequently too. However, the applicability of these new features are often related on strong assumptions about the system to analyze, and the evolving of the analysis tool can be complex to achieve when it is based on general ADLs. For example, with the systems investigated in [17], new feasibility tests were required to be implemented to perform the analysis following an extension of the transaction task model. Those new feasibility tests require to update the set of structural entities and properties used by the analysis tool, which requires a deep knowledge of the analysis tool. This experiment is a good example which reminds us that even if an ADL and its scheduling analysis associating tools are available for architecture designers, architecture designers need skills on scheduling analysis methods and tools to build models that are compliant with scheduling analysis methods/tools.

More generally, when using a flexible modeling language like MARTE, the coherence of the tool-chains must be maintained. We have shown that MARTE's design packages are sufficient for simple scheduling analysis methods [18, 19, 17]

and an architecture designed in MARTE can be directly used for the analysis. However, the profile's analysis packages also let the designer add information required by specific scheduling analysis methods. Users can extend the packages through the GQAM profile for any specific types of analysis. And then, it is a challenge to decide what and how to model a system with MARTE and to be sure that the model will be compliant with scheduling analysis tools.

From the previous points, we concluded that there is a need to define dedicated ADL to control the scheduling analysis tools. We have defined Cheddar ADL. Cheddar ADL encompasses only the useful concepts for such activity, and the tool evolution is made easier thank to the narrowest set of entities to deal with. The next section presents in details the goals and the entities of this ADL.

3 The Cheddar ADL: A specific ADL for scheduling analysis

The mainstream ADLs such as AADL and MARTE are very powerful to describe real-time systems. However, for the purpose of scheduling validation, additional tools and tool-chains must be used. In fact, scheduling verifications involve only a subset of those mainstream ADLs modeling capabilities but also require specific information or computations.

Scheduling analysis can be achieved on high-level models by using either with analytical feasibility tests or with scheduling simulation on the feasibility interval [20]. A possible solution to achieve scheduling analysis is to implement a specific ADL and a set of companion tools.

The goal of the Cheddar project is to facilitate the scheduling analysis of real-time architectures. Our approach consists in providing extensible tools based on precisely defined architectural concepts. Our approach can be seen as a mix between AADL and MARTE :

- We base our tools on a precisely defined and stable set of architectural concepts manipulated through a specific ADL : Cheddar ADL. In fact, the underlying theoretical model is the real-time scheduling theory. It brings a mathematical proven foundation and then, a clearly defined semantic to our ADL elements.

- Cheddar ADL and related tools are domain specific. Their modeling and their implementation are the result of a Model Driven Engineering (MDE) process tooled with a specific infrastructure. Whereas the use of the MDE facilitates the building of new releases of Cheddar ADL and of our existing tools, it also facilitates the building of new tools.

- We designed the Cheddar ADL language as a subset of AADL such that our tools can be interoperable with others AADL tools. However, we do not restrict to AADL. Facilitating tools interoperability for various ADL is also an important issue for us. We specifically address it with the help of the MDE.

In this section, we present the ADL that we have designed to model software and hardware architectures in the specific perspective of scheduling analysis. This ADL illustrates what an ADL may provide to model a real-time application on which designers expect to perform scheduling analysis with the real-time scheduling theory. We first define the concepts introduced by this ADL. Then, we describe scheduling analysis that are expected to be run on models expressed with this ADL. Finally, we summarize the strengths and weaknesses of the approach.

3.1 The core structural elements of the Cheddar ADL

Cheddar [21] is a GPL open-source schedulability tool composed of a graphical editor and a library of schedulability analysis modules. The library of analysis modules implements various analysis methods based on the real-time scheduling theory.

Basically, the core structural concepts manipulated through the Cheddar ADL come from the real-time scheduling theory. Applications are then modeled as a set of tasks defined with various parameters such as a deadline (or D_i in the sequel), a period (or P_i), or a capacity (or C_i, also called WCET[1]). The set of task parameters that an architecture designer must provide depends on the type of analysis he expects to run. For example, architecture designers must provide task control flow graph if they expect to run cache analysis during scheduling analysis [22].

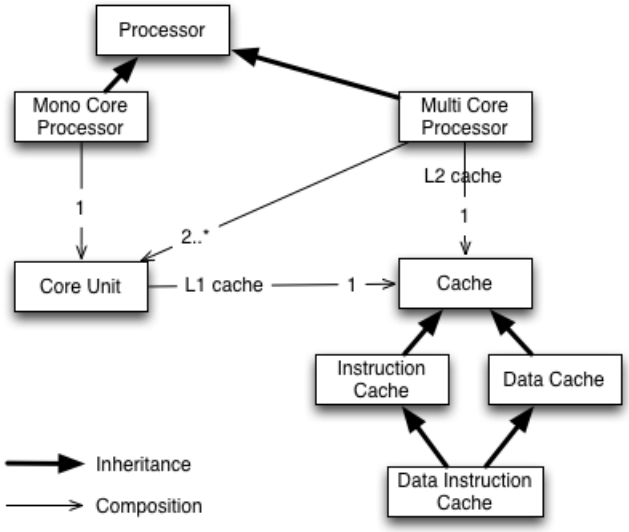


Figure 7: Cheddar main ADL hardware components.

[1]Worst Case Execution Time.

To support these real-time scheduling theory core concepts, the Cheddar ADL implements two types of entities: hardware components and software components. Hardware components represent the resources provided by the environment. Software components are deployed onto hardware components.

Cheddar provides limited capabilities to model hardware components. Indeed, real-time scheduling theory usually assumes simple models of hardware. On the contrary, software parts of a system are modeled in a more detailed way.

As shown in Figure 7, hardware components can be of three kinds. (1) *Core components* model entities providing a resource to sequentially run flows of control. (2) *Cache components* model memory caches related to one or several cores. (3) *Processors components* are composed of sets of cores and caches.

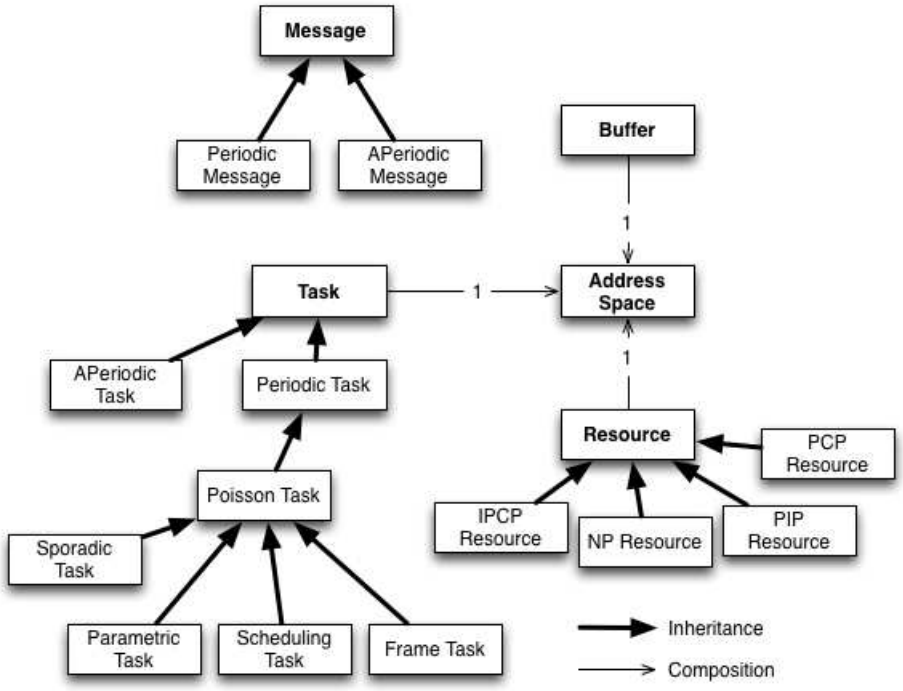


Figure 8: Cheddar main ADL software components.

Software components can be deployed on either core or processor components. Those deployments model two kinds of component connections that allow designers to express either global scheduling or partitioning scheduling[23]. The design of the software part of a real-time system can be specified with five component types. These component types are depicted by Figure 8. (1) *Address space components* model logical units of memory. They may be associated to an address protection mechanism. (2) *Task components* model flows of control. They are statically connected to address space components. (3) *Resource com-*

ponents may model any data structure, shared by tasks or not, synchronized or not. They may be accessed through classical priority inheritance protocols such as PCP [24]. They may model asynchronous communications between tasks of the same address space. Resource components are statically connected to address space components. (4) *Buffer components* model queued asynchronous data exchanges between tasks of the same address space. (5) *Message components* model queued asynchronous data exchanges between tasks located in different address spaces. Buffer, resource and message components specify types of connection between components.

```
<core_unit id=" 16">
  <name>core1</name>
    <scheduler_type>POSIX_1003_HIGHEST_PRIORITY_FIRST_PROTOCOL
      </scheduler_type>
    <preemptive_type>PREEMPTIVE</preemptive_type>...
<multi_core_processor id=" 17">
  <name>Soc_Leon4</name>
  <core ref=" 16"/>...
<periodic_task id=" 19">
  <name>RW_Data</name>
  <cpu_name>Soc_Leon4</cpu_name>
  <capacity>2</capacity>
  <deadline>200</deadline>
  <period>200</period> ...
<sporadic_task id=" 20">
  <name>Gyro_Data</name> ...
<periodic_task id=" 21">
  <name>DSS_Data</name> ...
```

Figure 9: Example of a Cheddar ADL model for the AOCS system.

Figure 9 shows a simple model of a real-time system specified with the Cheddar ADL. This is a model for the scheduling analysis of the AOCS system introduced in section 2.1. The software part of this example is composed of two periodic tasks (tasks RW_Data and DSS_Data) and a sporadic task (task $Gyro\_Data$) defined by their respective period, capacity and deadline. The hardware part only models a processor (called Soc_Leon4) including two cores (called $core1$ and $core2$).

From this simplified model of task, real-time scheduling theory provides various ways to perform schedulability analysis: analytic feasibility tests or scheduling simulations on the feasibility interval. The reminder of this section describes these two possibilities.

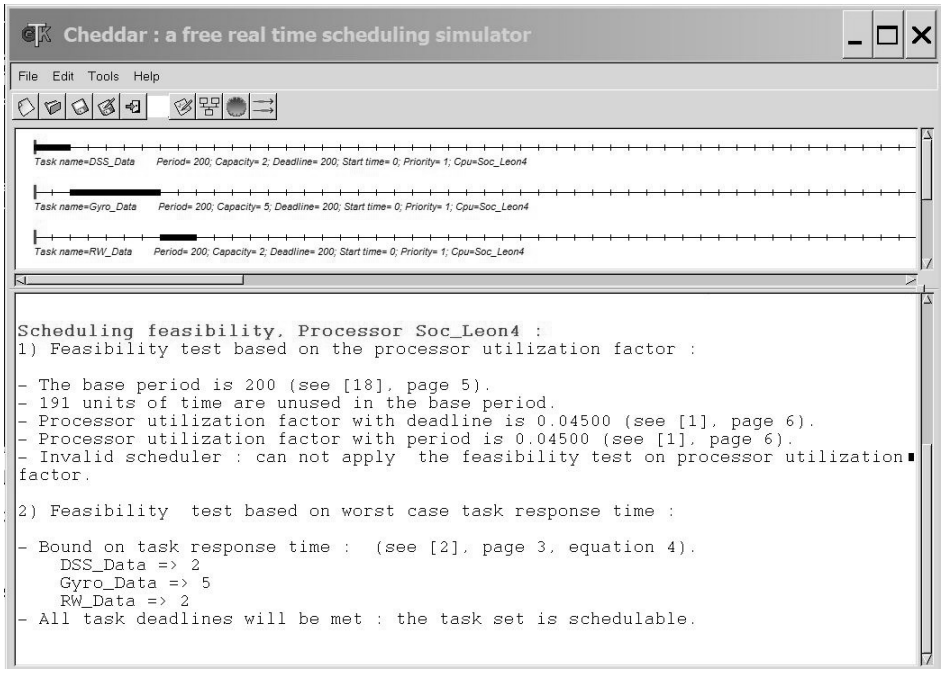


Figure 10: Scheduling analysis of the AOCS system by Cheddar.

3.2 Scheduling analysis with analytic feasibility tests

Formal verification [25] is achieved by analysis complying with analytic feasibility tests providing mathematically proven results on analyzed models. For example, Liu and Layland proposed feasibility tests for uni-processor real-time systems [26] based on Equation (1).

$$U = \sum_{i=1}^{n} \frac{C_i}{P_i} \tag{1}$$

This equation computes the processor utilization factor U. In the context of a fixed priority preemptive scheduling policy with priorities assigned according to Rate Monotonic, if $U \leq n(2^{\frac{1}{n}} - 1)$ then the system, under the Liu and Layland conditions, is schedulable, i.e. all tasks will meet their deadline. For this type of architecture, this feasibility test is a sufficient schedulability but not necessary condition. The Cheddar ADL supporting tool implements many other feasibility tests. Thus, by using the Cheddar ADL, for real-time architecture compliant with one or several feasibility tests implemented into Cheddar, it is possible to prove the validity of a given real-time architecture.

As an example, the AOCS system described in Figure 9 is mathematically proven to be compliant with the feasibility test presented in [27]. Applying this feasibility test leads to the result presented in Figure 10.

Unfortunately, it is not possible to analyze any kind of systems by this mean, and some theoretical results are often known as being too pessimistic. That's why additional techniques such as simulation can help to increase the confidence the designer has on his design.

3.3 Scheduling analysis with simulations

The Cheddar ADL allows the design and the analysis of specific schedulers. Such specific scheduler can be run on a given architecture. Using the Cheddar tool, simulations manually run by the architecture designer. The result consists of a graphical animation that the architecture designer observes. However, with the help of a simulation, a scheduling cannot be formally proven.

$$Priority(i) = Value\ at\ time(i)/Computation\ time \tag{2}$$

Cheddar provides a set of real-time schedulers and their analysis tools. Schedulers which are already implemented into Cheddar are mostly met in real-time applications and they can be used to perform analysis of many different types of real-time applications.

Specific schedulers or task models can be also specified with the help of the Cheddar ADL. Those specific schedulers allow users to extend the schedulability analysis capability without a deep understanding of Cheddar design and implementation. This feature allows users to quickly adapt the scheduling verification tool to their needs (i.e. implementing a scheduling method which does not exist yet into Cheddar).

```
start_section :
   aocs_dvs_priority : array (tasks_range) of integer;
   i : integer;
end section;

priority_section :
   for i in tasks_range loop
      aocs_dvs_priority(i):= tasks.value(i)/tasks.rest_of_capacity(i);
   end loop;
end section;

election_section :
   return max_to_index(aocs_dvs_priority);
end section;
```

Figure 11: Example of a Cheddar program modeling a Density value scheduler.

As an example, equation 2 specifies a Density value scheduler [28] which is not implemented as a built-in one in Cheddar. Figure 11 shows the implementation of a Density value scheduler with the Cheddar ADL and how such user-defined scheduler can be used to define a new processor type with our AOCS example. It this architecture model, algorithm of figure 11 is supposed to be stored in a separate file called *aocs_dvs.sc*.

3.4 Discussion

Cheddar ADL and its associated tools can be used to perform scheduling analysis. A significant part of the research effort consists in trying to simplify real-time system validation. However, using the Cheddar ADL alone remains difficult for architecture designers. Indeed, Cheddar ADL concepts are very close to real-time scheduling theory which are not so usual concepts for architecture designers. Furthermore, as the real-time scheduling theory, the ability of Cheddar ADL to model hardware and operating system parts is limited.

In practice, it is expected that architecture designers perform the modeling activity with separate systems or software engineering tools using standard modeling language such as AADL or MARTE. The Cheddar ADL used together with mainstream languages can offer extended modeling and validation capabilities.

Previous works have been done to use MARTE coupled with Cheddar for scheduling analysis. In [29] a MARTE to Cheddar model transformation has been proposed. This transformation was modified and experimented in [17]. These experiments were run on an industrial software radio system case-study. They have shown that simple modeling of Cheddar entities in MARTE with existing feasibility tests of Cheddar may give pessimistic scheduling analysis results. With the systems investigated in [17, 30], new feasibility tests have

to be implemented into Cheddar. Those new feasibility tests lead to extend Cheddar ADL with new attributes and entities.

Cheddar can be embedded into AADL or MARTE toolsets such as STOOD, AADLInspector[2] or Ocarina [31]. AADLInspector 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 rule checkers and bridgers for remote verification tools. AADLInspector includes two scheduling analysis tools : Cheddar and Marzhin [32]. AADLInspector performs prior analysis before calling scheduling analysis tools, in order to provide interactive assistance to AADL designers for a practical and efficient use of the real-time scheduling theory.

4 Model-driven process and ADLs

To handle very particular hardware or functional requirements, specific languages and tools may need to be implemented. Model-Driven Engineering (MDE) [14] aims at facilitating the specification and the implementation of specific languages and tools through the use of models.

Models that can be specified not only permit precise system documentation but also serve as input for automatic or semi-automatic production of the system and of the validation of tool-chains. Moreover, an ADL serves as a pivot language between several tools. Then, ADL models can ensure interoperability between different tool-chains.

In the context of the Cheddar project we have developed a specific ADL and a set of tools to help designers on verifications of the scheduling of real-time systems. For this purpose, the MDE is intensively used. Indeed, a significant part of our tools are automatically generated. Our tools can be used standalone or integrated within tool-chains. Integrating our tools within existing tool-chains raised interoperability issues. In this section we first present how the MDE is used to implement our ADL and its specific tool. Finally, we discuss about the interoperability issues and show how the MDE helped us to facilitate tools integration within tool-chains.

4.1 About processes to build ADL specific tools

With respect to our concerns regarding critical real-time systems, the big challenge for MDE is to actually provide means for early validation [33] of systems through the use of ADL models. As an example the Cheddar project mainly focuses on early scheduling verification of an execution model provided by ADL specifications.

Depending on the target execution model (e.g. multi-tasked, synchronous, ...), implementation options, and platforms (e.g. memory model, caches configuration, ...), the same set of ADL models can be implemented in many different

[2]STOOD and AADLInspector are products of Ellidiss Technologies (http://www.ellidiss.com)

ways. Moreover, new hardware elements and products continuously arise and have to be taken into account for the validation of systems. As a consequence, ADL based tools that are used for the early validation of real-time systems are (1) often specific tools and (2) often subject to changes. In this section, we first describe how a particular tool can be automatically produced from an ADL. Then we describe in more general terms, the MDE process and the set of MDE tools that are involved.

4.1.1 Building a specific scheduler with the help of the MDE

As we briefly describe in section 3.3, in order to run simulations, Cheddar provides a set of real-time schedulers and their analysis tools. Some schedulers are already implemented into Cheddar. Cheddar being implemented in Ada, those schedulers are also implemented in Ada.

Specific schedulers or task models can be also implemented. A specific scheduler can be implemented either in Ada or with the Cheddar ADL. Manually implementing a specific scheduler in Ada is a tedious and error prone task. Using the Cheddar ADL eases the implementation work because the designer has only to deal with high-level and simpler concepts. As it is depicted in Figure 12, with the Cheddar ADL, the process of implementing a specific scheduler is made of three steps. (1) The scheduler automaton is specified by using the Cheddar ADL. Figure 11 shows such a specification. (2) An interpreter of such automaton is made available as a Cheddar tool. This interpreter is useful to validate the specific scheduler. However, running specific schedulers this way can be very time consuming. (3) In order to speed up the running of a scheduler, the corresponding Ada code can be automatically produced from the specification of the scheduler written in the Cheddar ADL (the scheduler automaton). The generated Ada code is integrated and a new version of the Cheddar tools including the new specific scheduler is then produced [34].

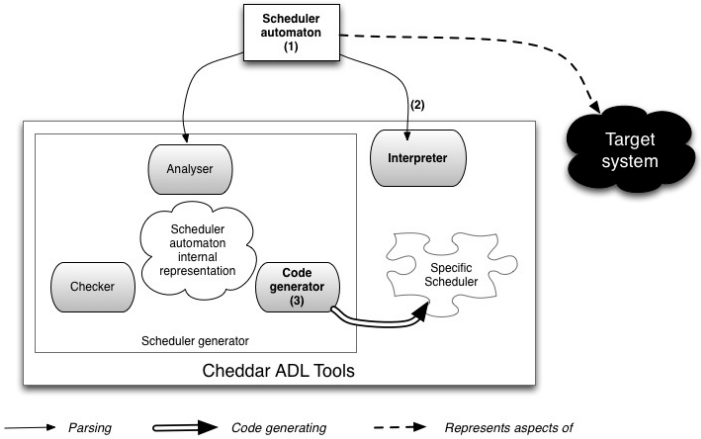


Figure 12: The MDE process for the building of a specific scheduler.

The Cheddar ADL meta-model is specified with a general purpose modeling language and the code generator is implemented within a meta-workbench. The next section describes in more general terms the different languages and tools that are involved to implement specialized tools such as these provided by Cheddar.

4.1.2 The MDE process and tools

As it is depicted in Figure 13, a particular ADL tool (e.g. Cheddar) is also a target system that must be specifically built, verified and validated through the use of a meta-tool. Figure 13 shows two dashed ellipsis which represent two iterations:

1. The inner iteration ellipsis is for the early validation of a particular aspect of the target system through the use of an ADL tool. As an example, Cheddar is used to validate the timing constraints.

2. The outer iteration ellipsis is for the specification, the verification and the validation of a particular ADL tool. Here, the process is twofold. First, the verification and the validation of the meta-specifications must be achieved [35]. Then late validation is processed through the use of the ADL tool itself (point 1).

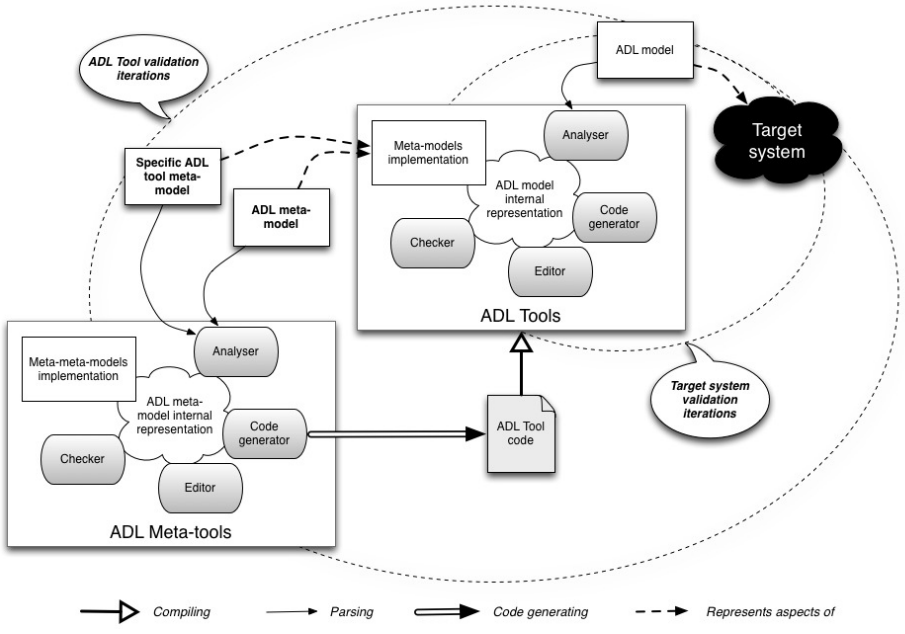


Figure 13: The MDE process for the building of ADL tools.

Thanks to the MDE, a part of an ADL tool can be automatically generated from its meta-model. First, the implementation of the meta-model elements is generated. This implementation generally includes the meta-model entities implementation (i.e. corresponding classes), a repository which is used to store and manage ADL models internal representations and an ADL model editor. Second, a set of ADL model checkers can also be automatically or semi-automatically generated. Checkers generation depends on specific meta-models (see specific ADL tool meta-model in Figure 13). As an example, an ADL can allow the specification of domain rules. These domain rules can be taken into account in order to generate specific checkers that are able to check if an ADL model is valid according to the domain rules. In that particular case, the specific ADL tool meta-model (see Figure 13) is the Cheddar ADL meta-model.

In the context of the Cheddar project, the meta-tools are implemented with the meta-workbench Platypus [36]. All our meta-models, domain rules and code generators are specified with the general purpose data modeling language EXPRESS [37]. As explained in a previous section, with this workbench, we can generate application specific schedulers. We can also generate infrastructure tools based on a set of architecture design patterns [38, 34]. In the next section, we focus on how we model and produce these infrastructure tools.

4.2 How an ADL can lead to a better tools interoperability

As shown in the previous section, it is possible to automatically generate a part of analysis tools based on their ADLs and we have shown the example of Cheddar specific schedulers. This section discusses how ADLs can also enable interoperability between model editors and analysis tools.

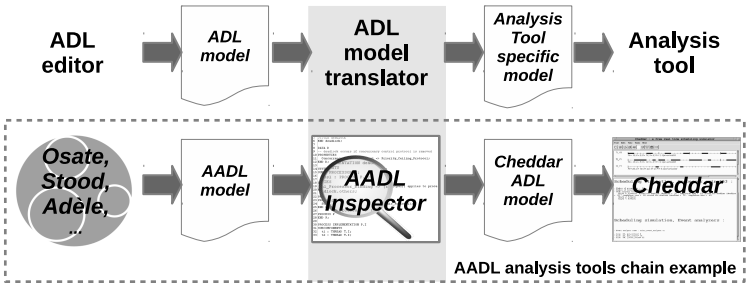
It is expected that architecture designers use ADLs that are adapted to their respective domains (e.g. EAST-ADL for automotive systems, AADL for avionic systems, ...). Analysis tools may also require dedicated ADLs to be used. However, usually ADLs dedicated to analysis tools are difficult to handle for architecture designer. In several tool-chains, analysis ADL models are hidden to the architecture designers.

As an example, Cheddar ADL introduces scheduling analysis concepts. As it is depicted by Figure 14, automatic model transformation can be used to hide Cheddar dedicated ADL's particularities. Thus, transformation from one ADL to another enables interoperability between model editors and analysis tools.

4.2.1 Issues raised by ADL model transformations

Using ADLs as pivot language to enforce interoperability between the different tools raises two kinds of issues:

Completeness issue First, information required to proceed analysis varies from one tool to another. For example, Cheddar requires concepts from the



A specific architecture analysis requires at least two kinds of tools: an editor which produces an input model for an analyzer. If the analyzer requires additional data or if the ADL used by the editor is different from the ADL used by the analyzer then a translator must be used. As an example, in the case of scheduling analysis of AADL models using Cheddar, *AADLInspector* can be used to translate an AADL model to a corresponding Cheddar ADL model.

Figure 14: ADL analysis tool-chains

real-time scheduling theory (e.g. priority, shared resource protocols, ...). Furthermore, depending on the analysis tool, abstraction levels may differ. Several possible abstract levels make selection of relevant information for the model analysis more difficult. For instance, a shared data can be modeled by several AADL communication paradigms: with data components, with AADL behavior annex subclauses, etc [39]. Deciding how each of these communication paradigms must be translated to Cheddar ADL is a complex issue.

Correctness issue Second, for an analysis to be relevant, the analyzed architecture needs to meet a set of assumptions. For example, the feasibility test of Liu and Layland based on equation (1) assumes that the system meets eight assumptions upon the software and hardware parts of the architecture: the architecture must be uniprocessor, tasks must be periodic and must be independently scheduled with a preemptive fixed priority scheduler and Rate Monotonic priority assignment, ...

As a consequence, the model of Figure 9 can not be analyzable with this test as it requires that all tasks are periodic. In this case, the architecture designer will have either to select another feasibility test or assume that task *Gyro_Data* is periodic.

4.2.2 Architecture design patterns to ensure tools interoperability

The previous section explains that completeness and correctness of architecture models must be enforced before applying a feasibility test on a model.

For such purpose, we proposed architecture design patterns [40, 38]. In this

context, an architecture design pattern is a set of constraints to be met by architecture models. By construction, each design pattern is compliant with a set of feasibility test assumptions. Then, an architecture model proposed by a designer which is compliant with a given design pattern is also compliant with the related feasibility tests.

Interoperability between model editor and scheduling analysis tools is ensured if a model edited by the designer is proved to be compliant with one or several architecture design patterns handled by the scheduling analysis tool.

4.2.3 Examples of architecture design patterns

To illustrate this approach, we present some examples of design patterns handled by both Cheddar (e.g. design patterns expressed with Cheddar ADL) and AADLInspector.

Each Cheddar ADL design pattern models communication and synchronization protocols between tasks. Constraints of a design pattern concern both the execution platform (the hardware platform and deployments) and the software part of the architecture.

For example, the architecture design pattern specific to time-triggered communications [41, 42] is composed of five constraints on the execution platform (R1 to R5) and four constraints on the software components (R6 to R9):

- R1: The scheduling protocol can be either earliest deadline first protocol, or rate monotonic protocol, or deadline monotonic protocol or any fixed priority scheduling.
- R2: The scheduler is preemptive.
- R3: The scheduler has no quantum.
- R4: The architecture must be uniprocessor.
- R5: There is no hierarchical scheduling.
- R6: Tasks must be periodic.
- R7: There is no shared data.
- R8: There is no buffer.
- R9: There is at least one time-triggered communication between tasks.

Four other design patterns have been defined for uniprocessor architectures: *Ravenscar*, *Blackboard*, *Queued buffer* and *Unplugged* [38]. Those design patterns model usual communication or synchronization protocols. *Ravenscar* is related to the Ada 2005 Ravenscar profile [43]. *Blackboard* and *Queued buffer* model communication services that exist in ARINC 653 [44].

4.3 Discussion

Supporting MDE with a comprehensive tool environment is crucial, as many of the techniques promoted as necessary in MDE strongly depend on proper tool support [45]. Having a well defined ADL helps in the building of a trusted real-time system because it can serve as the core element for the tools that are implemented in order to verify it. However, having an ADL is just not enough. A complete and working MDE environment must be used for the building of verification tools. Moreover, reliable code generators must be specified and interoperability between tools must be ensured.

The major difficulties lay in the specification of the specific meta-models and at the right abstraction level. Moreover, defining reliable specific meta-models is a long and costly task because of the verification and the validation process that implies tools building and adapting (specification of code generators, refactoring of the analysis tools, ...).

An ADL can serve as a pivot language between different kinds of tools. However, the interoperability between tools may be difficult to achieve by architecture designers. Enforcing interoperability by explicitly expressing the rules that govern this interoperability can help designers. This is one of the goals of our architecture design pattern approach. Architecture design patterns enable to restrict the utilization of an ADL.

This approach permits to explicitly express the interoperability rules and the analysis assumptions upon architecture model characteristics, but it can be hard to handle the different levels of abstraction between source and target ADL models. Moreover, most of transformations are not bijective. Thus, if a given analysis implies an update of the analysis model to meet schedulability requirements (e.g. Cheddar ADL model), finding its corresponding update to the original ADL (e.g. AADL model) is not trivial.

5 Related works

Numerous projects have investigated scheduling analysis and ADLs. In the first part, we present various analysis tools and methods that are able to handle standard ADLs such as AADL or MARTE/UML. Finally, we present some examples of projects that have focused on tools interoperability.

5.1 ADL analysis tools

Several scheduling analysis tools have been proposed for different ADLs. In this section, we focus on some of them that are AADL oriented.

In [46], the authors describes an AADL [7] based approach with OSATE [47]. A model transformation of AADL models to ACSR real-time process algebra [48] and its use by the VERSA tool [49] is proposed. In this approach, scheduling analysis is performed by model-checking techniques. Many other tools based on formal methods provide scheduling analysis on AADL models by model-checking. As an example, tools for Petri nets, automata or synchronous

languages have been adapted to AADL. This is the case of Tina [50], the Signal toolset Polychrony [51] or UPPAAL [52, 53]. Comparing to Cheddar, all those analysis tools above access schedulability through model state space exploration although Cheddar investigates schedulability analytically or by simulation.

ASSIST (Application Specific I/O Integration Support Tool) [54] is an other AADL scheduling analysis tool based on analytic methods. It computes communication delay bounds through a given I/O bus structure. AADL is used to model this structure and the hardware communication flows. The tool allows, during the scheduling analysis of real-time application, to take I/O wait time and cache I/O interferences into account. Contrary to the current Cheddar ADL implementation, ASSIST handles very detailed hardware models and uses network calculus theory to analyze bus delays and perform schedulability analysis.

Deubzer and al. also provide detailed hardware AADL models in the context of real-time multicore systems [55]. They propose extensions of AADL, in order to model heterogeneous cores with variable processing speed for example. To allow the modeling of such processor architecture, these extensions focus on the properties of the standard AADL component processor. Analysis is performed with a Monte-Carlo and scheduling simulation mixed approach.

5.2 ADL Tools interoperability

Other projects have focused on tools interoperability or mixed ADL approaches.

In [56], the authors combine verification, optimization and adaptation of safety critical embedded systems, in an engineering process with AADL. The process is design patterns based. It allows to adapt the workflow of rule-based refinement according to the user input models, to the targeted execution platform and to the analysis performed on intermediate transformed models. This engineering process is model-driven and composed of a three steps: first, identifying the refinement steps of a code generation process; second, decomposing each of these steps into a set of design patterns and finally, implementing these patterns using a hybrid domain specific language for model transformation. The approach of [56] may concern numerous analysis tools and code generators in the same time, which may raise complex interoperability issues, contrary to Cheddar design-patterns on which we focus on point to point interoperability (i.e. interoperability between two tools such as a model editor and a schedulability tool).

The works of [33] investigates schedulability of real-time systems at earliest design phases. Their approach is based on the combination of two modeling languages for architecture design (EAST-ADL2 and MARTE) and the integration of an open source toolset for scheduling analysis: MAST [57].

On one hand, EAST-ADL2 is an architecture description language defined as a domain specific language for the development of automotive electronic systems. On the other hand, MARTE is known by its rich expressive power to model various real-time system properties and constraints. The methodology proposed

by [33] has the objective of completing EAST-ADL models with MARTE entities to enable scheduling analysis at the design level.

Evensen and al. is another example of ADL mixed approach [58] . The authors show how AADL can extent a UML model to formally analyze architectures. With a case-study, the authors illustrate how qualitative analysis achieves detection of task deadlock from a model.

In the Cheddar project, we do not address direct analysis of architecture models combining several ADLs: we usually assume that architecture models written with any ADL will be transformed to Cheddar ADL models prior to analysis.

Finally, Ouhammou and al. proposed a sensitivity analysis based framework called MoSaRT (Modeling oriented Scheduling analysis of Real-Time systems) [59]. In this project, MoSaRT helps the designers to choose the feasibility tests corresponding to their designs. MoSaRT is also the name of the ADL proposed by the authors to handle the architecture models to verify. It is based on four complementary pillars: platform, behavior, analysis and functional model. Using MoSaRT ADL, the authors propose the different kinds of structural rules that must be met in order to verify compliance of a real-time system model to a set of scheduling analysis features. Cheddar explores a similar approach in order to enable interoperability between model editors and analysis tools [39, 40]. One of the main difference is how we model feasibility tests and the targeted architecture: for flexibility issues, we do not use decision trees as MoSaRT but entity associations.

We have presented several approaches mixing ADLs and schedubility analysis. Most of them propose model-driven strategies.

Whatever the considered approach, the authors rely on architecture description languages which prove that ADLs provide key concepts for scheduling analysis. Further reading about model-driven approaches and schedulability analysis of real-time systems can be found in [60].

In the next section we conclude this article with a list of opened issues related to ADLs and scheduling analysis.

6 Conclusion

In this article, we have presented several experiments that investigate how to perform scheduling analysis with several ADLs. We have investigated scheduling analysis with standard ADLs such as AADL or UML/MARTE and also with ADLs that are specific to scheduling analysis (e.g. Cheddar ADL). We have seen that the use of ADLs may contribute to build tool-chains: an ADL can be used as a pivot language between different tools and can also be used in model-driven engineering process to generate parts of the scheduling analysis tool.

However, experiments presented in this article, about Cheddar and also in the related works, raise many remaining issues and interesting challenges.

Even if the use of ADL helps scheduling analysis, it may stay difficult to apply as the designer must choose the right level of abstraction. A high level

of abstraction leads to simple models that are easy to analyze, but which imply pessimist and sometimes erroneous scheduling analysis results. On the contrary, models with a detailed description of the software and hardware parts make the analysis complex as few scheduling analysis methods may exist. The example of hardware modeling of section 2 is a good illustration: today, very few scheduling analysis methods are able to take care of memory caches. A similar issue is raised by UML/MARTE due to its flexibility.

A second example is related to the behavior of the functions of the system (e.g. the behavior of the tasks/threads and their associated source code). ADLs focus on architecture and some of them do not address how to model the behavior of the functions of the system. If architecture models do not contain detailed functions' behavior, scheduling analysis can produce pessimistic or erroneous results. Unfortunately, if ADLs provide means to model function behavior (e.g. formal languages such as automata or Petri nets), it makes the verification more complex.

A similar issue exists when architecture designers investigate various properties on the same architecture model (e.g. scheduling issues, fault tolerance issues, power consumption issues, etc). Each investigated property will imply the use of a formal model and its associated tool. It is then complex to enforce that the model includes all data required by all the expected analysis, but also, that the model does not contain any element that make analysis impossible or difficult to perform.

We must also notice that the use of an ADL to perform scheduling analysis requires a lot of skills from architecture designers to model both software and hardware parts and also to understand classical scheduling concepts and sometimes formal models if some of them are used to achieve analysis.

Finally, an important issue is related to engineering processes. Schedulability analysis is supposed to be performed prior to the full implementation of the software parts. However, this type of analysis also requires data extracted from partial system implementation ; this is the case of task WCET for example. As a result, architecture designers are sometimes lost when they try to apply those practices. In this context, ADLs may play a key role, thanks to their ability to be used as pivot languages.

Of course, many other issues could be presented here.

To conclude, we have presented various tools and projects that have shown how ADLs may be useful to perform scheduling analysis. However, we believe that the higher ADL weakness is related to the lack of feedback we have from architecture designers, and especially about the way architecture designers succeeded or failed to integrate scheduling analysis tools in their system/software engineering processes.

We then believe that investigating and experimenting engineering processes including scheduling analysis is one of the challenge that the community must address if we expect to improve impact of scheduling analysis in the design and the implementation of critical real-time systems.

7 Acknowledgments

Cheddar is supported by Ellidiss Technologies, Conseil régional de Bretagne, Conseil général du Finistère, BMO, EGIDE/Campus France PESSOA number 27380SA and Thales TCS.

References

[1] N. Medvidovic and R. Taylor, "A classification and comparison framework for software architecture description languages," *IEEE Transaction on Software Engineering*, vol. 26, no. 1, pp. 70–93, 2009.

[2] F. Terrier and S. Gérard, "MDE benefits for distributed, real-time and embedded systems," in *From Model-Driven Design to Resource Management for Distributed Embedded Systems (DIPES)*, pp. 15–24, 2006.

[3] C. Jouvray, M. Sall, and A. Kung, "Enforcing trust in embedded systems using models," in *Proceedings of the International Workshop on Security and Dependability for Resource Constrained Embedded Systems*, (New York, NY, USA), pp. 1:1–1:8, ACM, 2010.

[4] T. Frédéric, S. Gérard, and J. Delatour, "Towards an UML 2.0 profile for real-time execution platform modeling," Proceedings of the 18th Euromicro Conference on Real-Time Systems (ECRTS 06), Work in progress session, July 2006.

[5] V. Debruyne, F. Simonot-Lion, and Y. Trinquet, "EAST-ADL - An architecture description language," pp. 181–195, Book on Architecture Description Languages, IFIP International Federation for Information Processing, Springer Verlag, volume 176, 2005.

[6] P. Merle and J.-B. Stefani, "A formal specification of the Fractal component model in Alloy," INRIA Research Report 6721., November 2008.

[7] P. Feiler, B. Lewis, and S. Vestal, "The SAE AADL standard: A basis for model-based architecture-driven embedded systems engineering," in *Workshop on Model-Driven Embedded Systems*, May 2003.

[8] P. Feiler, D. Gluch, J. Hudak, and B. Lewis, "Embedded system architecture analysis using SAE AADL CMU/SEI-2004-TN-005," tech. rep., Software Engineering Institute, Carnegie Mellon University, Pittsburgh, june 2004.

[9] K. Almeida, J. Craveiro, R. Pinto, and J. Rufino, "Spaceborne software: typical spacecraft use-case and preliminary analysis of its timing requirements," in *Technical Report READAPT Project TR-13-01*, (Lisbon, Portugal), 2013.

[10] V. Gaudel, "Applicabilité des méthodes d'analyse et interopérabilité des outils de développement pour systèmes embarqués temps-réel critiques," Décembre 2014.

[11] Aeroflex Gaisler, *Dual core LEON4 SPARC V8 Processor LEON4-ASIC-DEMO, Data Sheet and User's Manual*, May 2011.

[12] Object Management Group, "MARTE specification," 2005.

[13] Object Management Group, "OMG website," 2013.

[14] D. C. Schmidt, "Model-driven engineering," *IEEE Computer*, vol. 39, February 2006.

[15] Object Management Group, "UML specification," 2011.

[16] S. Taha, A. Radermacher, S. Gerard, and J.-L. Dekeyser, "An open framework for detailed hardware modeling," in *Industrial Embedded Systems, 2007. SIES'07. International Symposium on*, pp. 118–125, IEEE, 2007.

[17] S. Li, F. Singhoff, S. Rubini, and M. Bourdellès, "Applicability of real-time schedulability analysis on a software radio protocol," in *Proceedings of the 2012 ACM conference on High integrity language technology*, (New York, USA), pp. 81–94, December 2012.

[18] M.-A. Peraldi-Frati and Y. Sorel, "From high-level modelling of time in marte to real-time scheduling analysis," in *Int. Workshop on Model Based Architecting and Construction of Embedded Systems*, pp. 129–144, 2008.

[19] S. Anssi, S. Tucci-Pergiovanni, C. Mraidha, A. Albinet, F. Terrier, and S. Gérard, "Completing east-adl2 with marte for enabling scheduling analysis for automotive applications," in *ERTS Conference, Toulouse*, 2010.

[20] E. Grolleau, J. Goossens, and L. Cucu-Grosjean, "On the periodic behavior of real-time schedulers on identical multiprocessor platforms," *arXiv preprint arXiv:1305.3849*, 2013.

[21] F. Singhoff, J. Legrand, L. Nana, and L. Marcé, "Cheddar : a Flexible Real-Time Scheduling Framework," *ACM SIGAda Ada Letters, ACM Press, New York, USA*, vol. 24, pp. 1–8, Dec. 2004.

[22] H. N. Tran, F. Singhoff, S. Rubini, and J. Boukhobza, "Instruction cache in hard real-time systems: modeling and integration in scheduling analysis tools with aadl," in *Proceedings of the 12th IEEE/IFIP International Conference on Embedded and Ubiquitous Computing (EUC 14)*, (Milan, Italy), August 2014.

[23] R. I. Davis and A. Burns, "A survey of hard real-time scheduling for multiprocessor systems," *ACM Comput. Surv.*, vol. 43, pp. 35:1–35:44, Oct. 2011.

[24] L. Sha, R. Rajkumar, and J. P. Lehoczky, "Priority inheritance protocols: An approach to real-time synchronization," *Computers, IEEE Transactions on*, vol. 39, no. 9, pp. 1175–1185, 1990.

[25] F. Cottet, J. Delacroix, C. Kaiser, and Z. Mammeri, *Scheduling in real-time systems* . John Wiley and Sons, octobre 2002.

[26] C. L. Liu and J. W. Layland, "Scheduling algorithms for multiprogramming in a hard real-time environnment," *Journal of the Association for Computing Machinery*, vol. 20, pp. 46–61, January 1973.

[27] M. Joseph and P. Pandya, "Finding response times in a real-time system," *The Computer Journal*, vol. 29, no. 5, pp. 390–395, 1986.

[28] S. A. Aldarmi and A. Burns, "Dynamic value-density for scheduling real-time systems," in *In Proceedings 11th Euromicro Conference on Real-Time Systems*, pp. 270–277, 1999.

[29] E. Maes and N. Vienne, "MARTE to cheddar transformation using ATL," technical report, THALES Research & Technologies, 2007.

[30] S. Li, S. Rubini, F. Singhoff, and M. Bourdellès, "Applying holistic schedulability tests to industrial systems: Experience and lessons learned," in *Proceedings of the 5th International Workshop on Analysis Tools and Methodologies for Embedded and Real-time System*, (Madrid, Spain), July 2014.

[31] J. Hugues, B. Zalila, L. Pautet, and F. Kordon, "From the prototype to the final embedded system using the ocarina aadl tool suite," *ACM Trans. Embed. Comput. Syst.*, vol. 7, pp. 42:1–42:25, Aug. 2008.

[32] P. Dissaux, O. Marc, S. Rubini, C. Fotsing, V. Gaudel, F. Singhoff, and H. N. Tran, "The SMART project: Multi-agent scheduling simulation of real-time architectures," in *Proceedings of the 7th European Congress Embedded Real Time Software and System (ERTSS)*, (Toulouse, France), Feb. 2014.

[33] S. Anssi, S. Gérard, A. Albinet, and F. Terrier, "Requirements and solutions for timing analysis of automotive systems," in *System Analysis and Modeling: About Models* (F. Kraemer and P. Herrmann, eds.), vol. 6598 of *Lecture Notes in Computer Science*, pp. 209–220, Springer Berlin / Heidelberg, 2011.

[34] F. Singhoff and A. Plantec, "Towards user-level extensibility of an Ada library: an experiment with cheddar," in *Proceedings of the 12th international conference on Reliable software technologies*, Ada-Europe'07, (Berlin, Heidelberg), pp. 180–191, Springer-Verlag, 2007.

[35] A. Plantec, "Faciliter la vérification et la validation de méta modèles dans le cadre de l'ingénierie dirigée par les modèles : une approche agile outillée et orientée données," November 2012.

[36] A. Plantec and V. Ribaud, "PLATYPUS: A STEP-based Integration Framework," in *14th Interdisciplinary Information Management Talks (IDIMT-2006)*, pp. 261–274, September 2006.

[37] I. T. N. WD, *EXPRESS Language Reference Manual*, 1997.

[38] V. Gaudel, F. Singhoff, A. Plantec, S. Rubini, P. Dissaux, and J. Legrand, "An Ada design pattern recognition tool for AADL performance analysis," *Ada Letters*, vol. 31, pp. 61–68, November 2011.

[39] P. Dissaux and F. Singhoff, "Stood and Cheddar : AADL as a pivot language for analysing performances of real time architectures," Proceedings of the European Real Time System conference. Toulouse, France, January 2008.

[40] A. Plantec, F. Singhoff, P. Dissaux, and J. Legrand, "Enforcing applicability of real-time scheduling theory feasibility tests with the use of design-patterns," in *Proceedings of the 4th international conference on Leveraging applications of formal methods, verification, and validation-Volume Part I*, pp. 4–17, Springer-Verlag, 2010.

[41] H. Kopetz and G. Bauer, "The time-triggered architecture," *Proceedings of the IEEE*, vol. 91, no. 1, pp. 112–126, 2003.

[42] S. Vestal, "Meta-H User's Manual, Version 1.27," tech. rep., 1998.

[43] A. Burns, B. Dobbing, and G. Romanski, "The Ravenscar tasking profile for high integrity real-time programs," in *Reliable Software Technologies Ada-Europe* (L. Asplund, ed.), vol. 1411 of *Lecture Notes in Computer Science*, pp. 263–275, Springer Netherlands, 1998.

[44] Arinc, *Avionics Application Software Standard Interface*. The ARINC Committee, January 1997.

[45] P. Mohagheghi and V. Dehlen, "Where is the proof? - a review of experiences from applying mde in industry," in *Proceedings of the 4th European conference on Model Driven Architecture: Foundations and Applications*, ECMDA-FA '08, (Berlin, Heidelberg), pp. 432–443, Springer-Verlag, 2008.

[46] O. Sokolsky, I. Lee, and D. Clarke, "Schedulability analysis of aadl models," in *IPDPS 2006, 20th International Parallel and Distributed Processing Symposium* (U. . IEEE Computer Society Washington, DC, ed.), (Rhodes Island, Greece), pp. 179–187, April 2006. ISBN: 1-4244-0054-6.

[47] SEI Carnegie Mellon University, "OSATE website," 2013.

[48] I. Lee, P. Brémond-Grégoire, and R. Gerber, "A process algebraic approach to the specification and analysis of resource-bound real-time systems," *Proceedings of the IEEE*, vol. 82, no. 1, pp. 158–171, 1994.

[49] D. Clarke, I. Lee, and H. L. Xie, "VERSA: a tool for the specification and analysis of resource-bound real-time systems," technical report, University of Pennsylvania, 1993.

[50] B. Berthomieu, J. P. Bodeveix, S. D. Zilio, P. Dissaux, M. Filali, P. Gaufillet, S. Heim, and F. Vernadat, "Formal verification of aadl models with fiacre and tina," in *ERTSS 2010 5th International Congress and Exhibition on Embedded Real-Time Software and Systems*, 2010.

[51] Y. Ma, H. Yu, T. Gautier, P. L. Guernic, J.-P. Talpin, L. Besnard, and M. Heitz, "Toward Polychronous Analysis and Validation for Timed Software Architectures in AADL," in *The Design, Automation, and Test in Europe (DATE) conference*, (Grenoble, France), p. 6, Mar. 2013.

[52] A. Johnsen, "Fixed-priority preemptive scheduling semantics of aadl in uppaal timed automata," tech. rep., Mälardalen University, School of Innovation, Design and Engineering, 2012.

[53] M. Mikučionis, K. G. Larsen, J. I. Rasmussen, B. Nielsen, A. Skou, S. U. Palm, J. S. Pedersen, and P. Hougaard, "Schedulability analysis using uppaal: Herschel-planck case study," in *Proceedings of the 4th international conference on Leveraging applications of formal methods, verification, and validation - Volume Part II*, ISoLA'10, (Berlin, Heidelberg), pp. 175–190, Springer-Verlag, 2010.

[54] M.-Y. Nam, R. Pellizzoni, L. Sha, and R. M. Bradford, "ASIIST: Aplication specific I/O integration support tool for real-time bus architecture designs," in *Proceedings of the 14th IEEE International Conference on Engineering of Complex Computer System*, pp. 11–22, June 2009.

[55] M. Deubzer, M. Hobelsberger, J. Mottok, F. Schiller, R. Dumke, M. Siegle, U. Margull, M. Niemetz, and G. Wirrer, "Modeling and simulation of embedded real-time multicore systems," in *Proceedings of the 3rd Embedded Software Engineering Congress*, pp. 228–241, 2010.

[56] F. Cadoret, E. Borde, S. Gardoll, and L. Pautet, "Design patterns for rule-based refinement of safety critical embedded systems models," in *Engineering of Complex Computer Systems (ICECCS), 2012 17th International Conference on*, pp. 67–76, IEEE, 2012.

[57] M. G. Harbour, J. G. Garcia, J. Palencia, and J. Drake Moyano, "MAST: modeling and analysis suite for real-time applications," in *Proceedings of the 13th Euromicro Conference on Real-Time Systems*, pp. 125–134, IEEE Comput. Soc, 2001.

[58] K. D. Evensen, "Reducing uncertainty in architectural decisions with AADL," in *Proceedings of the 44th Hawaii International Conference on System sciences*, (Kauai, HI), pp. 1–9, 2011. ISSN : 1530-1605.

[59] Y. Ouhammou, E. Grolleau, M. Richard, and P. Richard, "From model-based design to real-time analysis," in *The Fourth International Conference on Advances in System Testing and Validation Lifecycle (VALID)*, pp. 45–50, 2012.

[60] M. Bordin, M. Panunzio, and T. Vardanega, "Fitting schedulability analysis theory into model-driven-engineering," in *ECRTS'08: Euromicro Conference on Real-Time Systems*, (Prague), pp. 135–144, 2008. ISBN: 978-0-7695-3298-1.