Teaching Real-Time Scheduling Analysis with Cheddar

Frank Singhoff [1], Alain Plantec [1], Stéphane Rubini [1], Hai-Nam Tran [1], Vincent Gaudel [1], Jalil Boukhobza [1], Laurent Lemarchand [1], Shuai Li [1], Etienne Borde [2], Laurent Pautet [2], Jérôme Hugues [3], Pierre Dissaux [4], Jérôme Legrand [4], Christian Fotsing [5], Blandine Djika [5]

[1]Lab-STICC CNRS UMR 6285, UBO, UEB, 20 av Le Gorgeu, 29200 Brest, France
Email: firstname.lastname@univ-brest.fr
[2]Institut Mines-Telecom, TELECOM ParisTech, LTCI CNRS UMR 5141, 46 rue Barrault, 75013 Paris, France
Email:firstname.lastname@telecom-paristech.fr
[3]Université de Toulouse, ISAE, 10 Av. E. Belin, 31055 Toulouse Cedex 4, France
Email: jerome.hugues@isae.fr
[4]Ellidiss Technologies, 24 Quai de la Douane, 29200 Brest, France
Email:firstname.lastname@ellidiss.com
[5]Institut Universitaire de la Côte, BP 3001, Douala, Cameroun
Email:firstname.lastname@myiuc.com

***Abstract*—This article is a presentation of the Cheddar toolset. Cheddar is a GPL open-source scheduling analysis tool. It has been designed and distributed to allow students to understand the main concepts of the real-time scheduling theory. The tool is built around a simplified ADL (Architecture Description Language) devoted to real-time scheduling theory. Students can directly build their real-time systems models with this ADL and its associated editor, however, it is expected that they use modeling tools to illustrate how scheduling analysis fits in an engineering process. In this article, we introduce the Cheddar ADL and the scheduling analysis features of Cheddar. We also present how Cheddar is implemented and how it can be adapted to specific requirements. Two examples of use of Cheddar are then described. Finally, in the annex of this article, teachers may find a sample of hand-outs that may be used to illustrate real-time scheduling theory with their students.**

I. Introduction

This article is a presentation of the Cheddar toolset, a GPL open-source scheduling analysis tool [1]. This tool addresses the verification of real-time critical systems.

Real-time critical systems have to meet hard timing constraints implied by their environment. Safety failures, including violation of timing constraints, could lead to life losses or environmental damages for this kind of systems [2]. Real-time scheduling theory provides algebraic methods and algorithms in order to perform timing constraint verifications [3]. Real-time scheduling theory foundations were proposed in the 1970s [4] and have led to extensive researches [5].

This theory provides several ways to perform scheduling analysis. Most of the time, scheduling analysis is achieved either with feasibility tests or with scheduling simulations.

A feasibility test is an analytical method which usually allows designers to compute performance criteria in order to assess if task deadlines will be met.

Scheduling simulation to assess schedulability consists in applying scheduling algorithms during a period of time in order to compute the schedule of the system. A designer can then check during this period of time if no deadline is missed. It is usually expected to compute such a schedule for a period of time large enough in order to capture all possible states of the system. In this case, we call such period of time a feasibility interval [6].

Several tools implement those scheduling analysis methods. MAST [7], Rapid-RMA [8], SymTa/S [9] and Cheddar [1] are examples of them. Cheddar has been more specifically designed to be devoted to students, in order to present them the main concepts of the real-time scheduling theory. In this article, we give a short overview of the analysis features currently implemented into Cheddar.

All those scheduling analysis tools handle models of the real-time systems to verify. Architecture Description Languages (or ADL in the sequel) can be used to express such models. ADL are languages that allow designers to specify, formally or not, the design of a system. Various ADLs have been proposed in the context of critical real-time systems. UML-MARTE [10], EAST ADL [11], Fractal [12] or AADL [13] are some of them.

Usually, those ADLs provide the abstraction of components, connections and deployments [14]. A component is an entity modeling a part of the system. Many ADLs allow the specification of both 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 the resources of the system are shared.

Cheddar is built around a simple ADL devoted to real-time scheduling theory: Cheddar ADL. Students can directly build their real-time system models with this ADL and its associated editor. However, it is expected that they use specific modeling tools and their related ADLs to illustrate how scheduling analysis fits in an engineering process. In this article, we introduce Cheddar ADL and we present its main concepts.

Model-Driven Engineering (MDE) [15] aims at facilitating the specification and the implementation of specific languages, systems 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 its verification. In the context of the Cheddar project, the MDE is intensively used. Indeed, a significant part of Cheddar tools is automatically generated. These tools can be used standalone or integrated within tool-chains. Integrating our tools within existing tool-chains also raises interoperability issues that may be controled thanks to MDE processes. In this article, we explain how MDE has been applied when implementing Cheddar.

Finally, having a scheduling analysis tool and a modeling language are not enough to achieve scheduling analysis. Putting scheduling analysis tools in the engineering process is also a tedious task. In this article, we illustrate how scheduling analysis tools can take place in a software engineering process with two examples of use of Cheddar.

Then, in the rest of this article, we first introduce Cheddar ADL. In section 3, a brief summary of the scheduling analysis features of Cheddar is also presented. In section 4, we shortly explain how the Cheddar tools can be automatically produced thanks to a MDE process. Finally, two typical examples of use of the Cheddar tools are exposed in section 5.

II. Cheddar ADL: A specific ADL for scheduling analysis

In this section, we present Cheddar ADL, the ADL that we have designed to model software and hardware architectures in the specific perspective of scheduling analysis. The mainstream ADLs such as AADL [16] and MARTE [17] are very powerful to describe real-time systems and some experiments to apply Cheddar on such standard ADLs have been done [18], [19].

However, for the purpose of scheduling validation, additional tools and tool-chains may be used. In fact, scheduling verifications involve not only a subset of those mainstream ADLs modeling capabilities but also require specific information or computations.

Cheddar ADL and related tools are domain specific. Their modeling and their implementation are the results 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.

In the sequel, we first define the concepts introduced by this ADL. Then, we describe scheduling analysis that are expected to be run on models expressed with Cheddar ADL.

Basically, the main concepts manipulated through Cheddar ADL come from the real-time scheduling theory. The only purpose of Cheddar ADL is to model the concepts required to perform scheduling analysis techniques that we usually present to students. Real-time systems are then modeled as a set of entities with various attributes. For example, the concept of task is one of the main concepts and is defined with classical attributes such as deadline, period and capacity. An exhaustive list of both entities and attributes is given in the Cheddar ADL user guide [20].

To support 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 model the resource requirements : they are deployed onto hardware components.

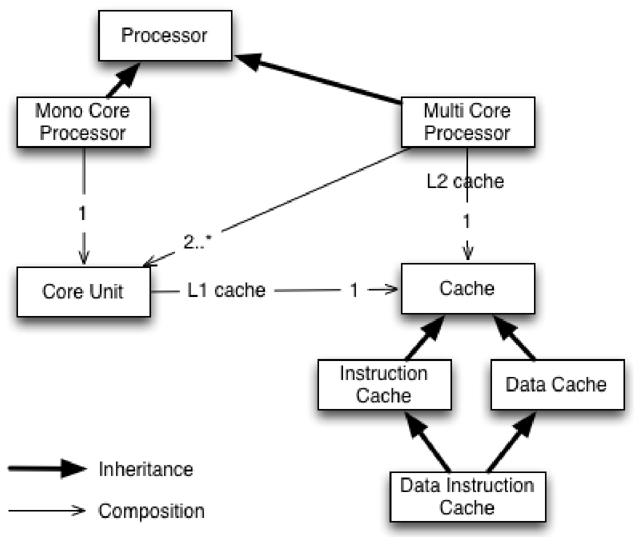


Fig. 1. Cheddar ADL main hardware components.

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 1, 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.

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 partitioned scheduling [21]. 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 2. (1) *Address space components* model a group of resources that can be accessed. 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 components* 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 [22]. They may model asynchronous

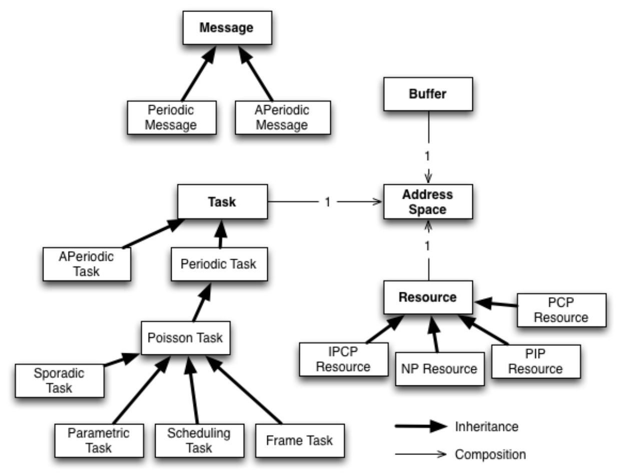


Fig. 2. Cheddar ADL main software components.

communications between tasks located in the same address space. Resource components are statically connected to address space components. (4) *Buffer components* model queued asynchronous data exchanges between tasks located in 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, i.e types of dependencies between tasks.

```
<core_unit id="␣16">
  <name>core1 </name>
     <scheduler_type>POSIX_1003_HIGHEST_
                 PRIORITY_FIRST_PROTOCOL
        </scheduler_type>
     <preemptive_type>
                 PREEMPTIVE
      </preemptive_type >...
<mono_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> ...
```

Fig. 3. Example of a Cheddar ADL model for the AOCS system.

Figure 3 shows a simple model of a real-time system specified with the Cheddar ADL. This is a model for the scheduling analysis of an Attitude and Orbital Control System (AOCS) of a spacecraft case study [23], [24]. An AOCS maintains the spacecraft orbit and ensures the spacecraft is oriented to achieve the expected functionality. This subsystem consists of a 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.

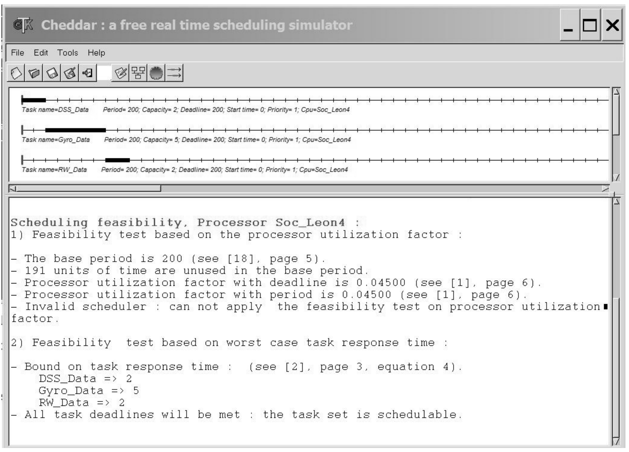
This AOCS is composed of several tasks:

- **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.
- **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.
- **IRES Data Task**: the InfraRed Earth Sensor (IRES) scans a large field of view and then detects signals at the Earth/Space transitions.

The software part of this example is composed of several periodic and sporadic tasks. For example, tasks RW_Data and DSS_Data are periodic while $Gyro\_Data$ is sporadic. They are all defined by their respective period, capacity and deadline. The hardware part only models a processor (called Soc_Leon4) including two cores (called $core1$ and $core2$).

III. Analysis features provided by Cheddar

From a Cheddar ADL model, real-time scheduling theory provides various ways to perform scheduling analysis: verifications can be performed either with feasibility tests or with scheduling simulations on the feasibility interval. Cheddar implements classical methods of both verification techniques in order to illustrate to the student how scheduling analysis can be driven. In this section, we first introduce feasibility tests implemented into Cheddar and then, we present its scheduling simulation features.



```
Task name=DSS_Data     Period= 200; Capacity= 2; Deadline= 200; Start time= 0; Priority= 1; Cpu=Soc_Leon4
Task name=Gyro_Data    Period= 200; Capacity= 5; Deadline= 200; Start time= 0; Priority= 1; Cpu=Soc_Leon4
Task name=RW_Data      Period= 200; Capacity= 2; Deadline= 200; Start time= 0; Priority= 1; Cpu=Soc_Leon4
```

```
Scheduling feasibility, Processor Soc_Leon4 :
1) Feasibility test based on the processor utilization factor :

- The base period is 200 (see [18], page 5).
- 191 units of time are unused in the base period.
- Processor utilization factor with deadline is 0.04500 (see [1], page 6).
- Processor utilization factor with period is 0.04500 (see [1], page 6).
- Invalid scheduler : can not apply  the feasibility test on processor utilization
factor.

2) Feasibility  test based on worst case task response time :

- Bound on task response time :  (see [2], page 3, equation 4).
    DSS_Data => 2
    Gyro_Data => 5
    RW_Data => 2
- All task deadlines will be met : the task set is schedulable.
```

Fig. 4. Scheduling analysis of the AOCS system by Cheddar.

A. Scheduling analysis with feasibility tests

One of the very first feasibility tests usually presented to students is the Liu and Layland feasibility test for uniprocessor real-time systems [4] 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 but not necessary schedulability condition.

The Cheddar tool implements various feasibility tests. Processor utilization feasibility tests can be applied on other scheduling policies. Furthermore, worst case response time can be computed on periodic tasks, organized or not in linear/graph transactions [25]. Those worst case response time can integrate delays related to shared resources (i.e. shared resource blocking time [22]). Finally, few feasibility tests for hierarchical architectures have also been implemented in order to explain to students the main concepts of compositional analysis [26].

As an example, the AOCS system described in Figure 3 is compliant with the feasibility test presented in [27]. Applying this feasibility test leads to the result presented in Figure 4.

It is not possible to analyze all systems by feasibility tests, 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/her design. We discuss this issue in the sequel.

B. Scheduling analysis with simulations

Cheddar implements several classical scheduling algorithms. Students may experiment classical schedulers such as Rate Monotonic, Deadline Monotonic, EDF, LLF, MUF, POSIX 1003 policies, both preemptive and non preemptive. Those algorithms have been implemented in the context of uniprocessor scheduling and also in the context of global multiprocessor scheduling. Global scheduling is also illustrated to the student by a specific global multiprocessor scheduling (e.g. Proportionate Fair [28]) and two task migration policies. The tool also implements non real-time policies (i.e. round-robin and time sharing policies) allowing the students to compare and understand specificities of real-time scheduling policies. Furthermore, hierarchical scheduling is illustrated by an implementation of the ARINC653 scheduling policies and several classical aperiodic servers (deferable, sporadic, priority exchange).

Scheduling simulations can be run for usual task models such as periodic, aperiodic, sporadic, GMF tasks or task released according to a Poisson process. Tasks can be constrained by dependencies related to shared resources, precedence or communication task relationships. Tasks may also be organized in linear or graph transactions. Again, classical policies have been implemented for the students. For example, a student can design architecture models with PIP, ICPP, PCP or FIFO shared resource protocols. During the design phase, he can also experiment various priority assignment algorithms such as Rate Monotonic [4], Deadline Monotonic [29], the Audsley's algorithm [30] or one devoted to architectures with caches [31].

From an architecture model, various performance criteria can be extracted from scheduling simulation: worst/best/average response time, probability distribution of response time, worst/best/average shared resource blocking time, number of context switch or preemption, deadlock, priority inversion or specific properties defined with a domain specific language.

Furthermore, specific schedulers or task models can be also specified with the help of the Cheddar ADL. Those specific schedulers allow users to extend the scheduling 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 in Cheddar).

In the sequel, we show how to implement such specific schedulers with Cheddar. Equation 2 specifies a Density value scheduler [32] which does not exist in the current implementation of Cheddar.

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

Figure 5 shows the implementation of this scheduler with the Cheddar ADL and how such a user-defined scheduler can be used to define a new processor type with our AOCS example. With this architecture model, algorithm of Figure 5 is supposed to be stored in a separate file called *aocs_dvs.sc*.

Basically, the *priority_section* expresses how priorities have to be computed at each unit of time during simulation and *election_section* describes how to select the task to run when priorities have been computed: here, we select the task with the highest *aocs_dvs_priority* priority.

```
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;
```

Fig. 5. Example of a Cheddar program modeling a Density value scheduler. Those statements are activated by an automaton during scheduling simulations.

C. Conclusion

In this section, we have presented the main analysis features implemented into Cheddar. Scheduling analysis can be performed either with feasibility tests or scheduling simulation. Some of the features of Cheddar have not been described here. The tool also provides various methods to assign priorities or to perform partitioning. All those features are described in the Cheddar user-guide [20].

However, using the Cheddar ADL alone remains difficult for architecture designers. Indeed, Cheddar ADL concepts are very close to real-time scheduling theory which may be not 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. We discuss this issue in the next sections.

IV. Model-driven process of Cheddar

New hardware elements and products continuously arise and have to be taken into account during the verification of real-time systems. As a consequence, ADL based tools that are used for the early verification of real-time systems are often specific tools and are often subject to changes. This is typically the case of Cheddar ADL.

To help keep Cheddar up-to-date according to use-cases, a model driven process is used to automatically build a part of the tool. In this section, we describe how some parts of Cheddar are automatically produced from its meta-models. We first describe how the infrastructure of Cheddar is specified, validated and automatically generated through the use of a meta-workbench. Then we focus on a more specific generating process regarding the automatic implementation of specific schedulers.

A. The Cheddar building process

As it is depicted in Figure 6, Cheddar is also a target system that must be specifically specified, validated and built through the use of a meta-workbench. Figure 6 shows two dashed ellipsis which represent two iterations:

1) The inner iteration ellipsis is for the early validation of timing constraints of the target system through the use of Cheddar.
2) The outer iteration ellipsis is for the specification, the validation and the generation of Cheddar itself. Here, the process is twofold. First, the validation of the meta-specifications must be achieved [33]. The meta-specifications are made of the Cheddar ADL meta-model (see Figures 1 and 2), of the Cheddar language meta-model and of code generators. Then late validation is processed through the use of Cheddar as explained in point 1.

Thanks to the MDE, a part of Cheddar is automatically generated from its meta-models. The generated part includes the meta-models entities implementation (i.e. corresponding classes), a repository which is used to store and manage Cheddar ADL models internal representations and a Cheddar ADL model exchange component.

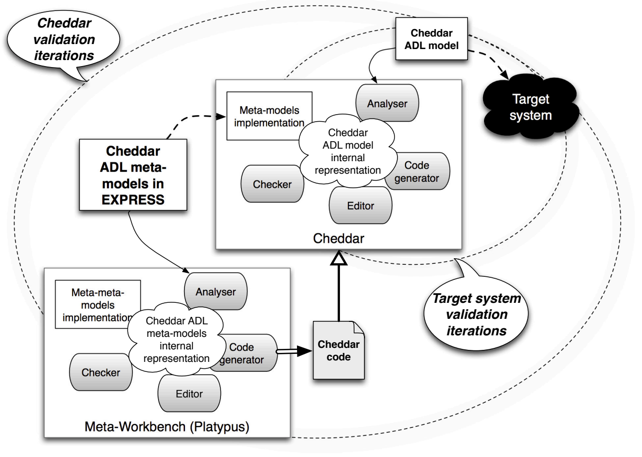

Cheddar validation iterations
Cheddar ADL model
Target system
Meta-models implementation
Analyser
Cheddar ADL model internal representation
Code generator
Checker
Editor
Cheddar
Cheddar ADL meta-models in EXPRESS
Meta-meta-models implementation
Analyser
Cheddar ADL meta-models internal representation
Code generator
Checker
Editor
Meta-Workbench (Platypus)
Cheddar code
Target system validation iterations
Compiling
Parsing
Code generating
Represents aspects of

Fig. 6. The MDE process for the building of Cheddar.

The meta-workbench Platypus [34] is used to implement the code generators. All our meta-models, domain rules and code generators are specified with the general purpose data modeling language EXPRESS [35].

B. Building a specific scheduler

As we briefly describe in section III-B, in order to run simulations, Cheddar provides a set of real-time schedulers and their analysis tools. Some schedulers are already present 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 written either in Ada or with the Cheddar ADL (an example of a specific scheduler is shown in Figure 5). Manually implementing a specific scheduler in Ada is a tedious and error prone task. Using the Cheddar ADL helps the designer as he has only to deal with high-level and simpler concepts. As it is depicted in Figure 7, with the Cheddar ADL, the process of implementing a specific scheduler is made of three steps. (1) The scheduler is specified by an automaton with the Cheddar ADL. Figure 5 shows a part of such 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 [36].

C. Conclusion

Supporting MDE with a comprehensive tool environment is crucial, as many of the techniques promoted as necessary in

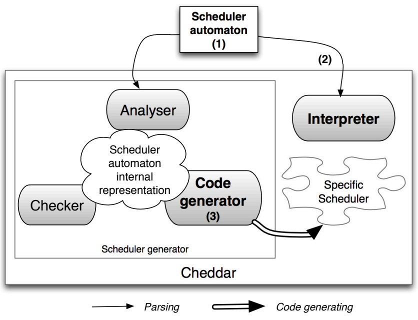


Fig. 7. The MDE process for the building of a specific scheduler.

MDE strongly depend on proper tool support [37]. 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 well defined 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.

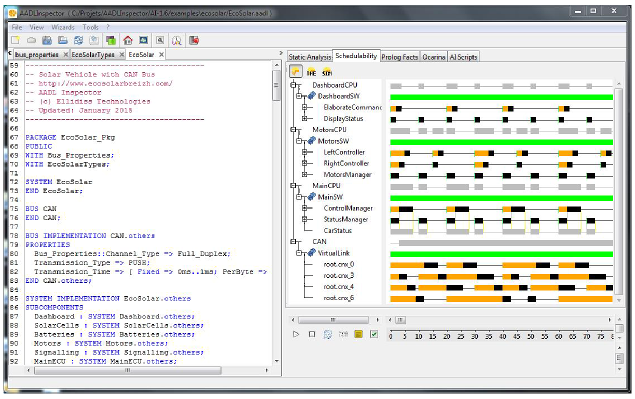
The major difficulty lays in the specification of the meta-models at the right abstraction level. Moreover, defining reliable specific meta-models is a long and costly task because of the verification and the validation process that implies tools building and adapting (specification of code generators, refactoring of the analysis tools, ...). Another difficulty is the integration of the generated code when Cheddar is embedded in a tool chain. Indeed, it may be difficult to anticipate what are the needs of the other tools. In the next section, we present two examples of tool chains embedding Cheddar where integration is achieved thanks to AADL.

V. Integration of Cheddar in AADL Inspector and the TASTE tool-chain

As it has been explained in the previous sections, Cheddar has been initially designed to be used as a standalone tool for an academic usage. In that respect, it includes its own Graphical User Interface to enter real-time models and display the scheduling analysis results. Nevertheless, such a tool can also be highly profitable for industry oriented environments and modeling languages. This however requires a more modular usage of the tool and the development of additional components to interface it to this new context. This is what has been done to integrate Cheddar as an analysis plug-in of the AADL Inspector tool, and of the Concurrency View editor of the TASTE tool-chain (TASTE-CV).

In order to be able to use Cheddar as an analysis backend while minimizing the interfacing issues, the Graphical User Interface and some model input features have been separated from the main scheduling analysis module. The result of this engineering operation is a subset of Cheddar called CheddarKernel that can easily be integrated within a system or software development environment. The containing tool must thus provide new control and data interfaces with the user.

AADL Inspector (see Figure 8) is a model processing framework that embeds a set of generic features to load real-time models and convert them into the appropriate format to let them be properly processed by various analysis or production tools. Although AADL Inspector uses the AADL standard as a base reference for its input models, it can also process UML profiles such as SysML and MARTE or be coupled to graphical editors such as the Stood tool. In the case of Cheddar, the input model is converted into a corresponding representation in the Cheddar ADL that has been presented before. Then, the results of the analysis, either coming from the feasibility tests or the simulation, are processed by the AADL Inspector Graphical User Interface.



```
-----------------------------------------
-- Solar Vehicle with CAN Bus
-- http://www.ecosolarbreizh.com/
-- AADL Inspector
-- (c) Ellidiss Technologies
-- Updated: January 2015
-----------------------------------------

PACKAGE EcoSolar_Pkg
PUBLIC
WITH Bus_Properties;
WITH EcoSolarTypes;

SYSTEM EcoSolar
END EcoSolar;

BUS CAN
END CAN;

BUS IMPLEMENTATION CAN.others
PROPERTIES
  Bus_Properties::Channel_Type => Full_Duplex;
  Transmission_Type => PUSH;
  Transmission_Time => [ Fixed => 0ms..1ms; PerByte =>
END CAN.others;

SYSTEM IMPLEMENTATION EcoSolar.others
SUBCOMPONENTS
  Dashboard : SYSTEM Dashboard.others;
  SolarCells : SYSTEM SolarCells.others;
  Batteries : SYSTEM Batteries.others;
  Motors : SYSTEM Motors.others;
  Signalling : SYSTEM Signalling.others;
  MainECU : SYSTEM MainECU.others;
```

Fig. 8. AADLInspector embedding Cheddar.

AADL Inspector embeds other analysis plug-ins, and in particular the Marzhin event-based simulation engine that complements the timing analysis by adding a capability to observe the behavior of non periodic systems. During the SMART collaborative project [38], Cheddar has been used as a reference tool to calibrate the Marzhin simulator. For a set of input model test cases where the two tools could provide a significant result, an automatic comparison of the two simulation traces has been implemented for that purpose.

TASTE is a software development tool-chain that is developed by the European Space Agency and a team of subcontractors [39]. TASTE supports a well defined design process for embedded applications in the space domain. This process includes modeling, verification, generation and testing phases that are supported by dedicated tools. The scheduling analysis phase is insured by the TASTE-CV tool which includes Cheddar in a similar way as AADL Inspector does.

These examples show that Cheddar can also be profitable to bring the benefit of early scheduling analysis into industrial development environments and processes. The next section describe another use of Cheddar in a code generation framework.

VI. RAMSES: AADL Models Refinement and Scheduling Analysis

RAMSES[1] is a model-to-model transformation framework where both input and output models are based on AADL [40].

In MDE, model transformations play an important role: they formalize in reusable artefacts (i.e. model transformation source code), the implementation of recurrent design patterns (e.g. safety or security design patterns), design decisions (e.g.

[1]Refinement of AADL Models for Synthesis of Embedded Systems

deployment of software components on hardware components), or model refinements (e.g. transformation steps towards code generation). Model refinements were first implemented in RAMSES in order to evaluate the impact of code generation patterns on Non-Functional Properties (NFP) of a real-time embedded system. Figure 9 gives an overview of RAMSES. The execution of RAMSES starts with (i) an input model, (ii) a selected platform (among platforms supported by RAMSES), and (iii) a selected set of analysis methods. The current version of RAMSES supports ARINC653 and OSEK compliant platforms.

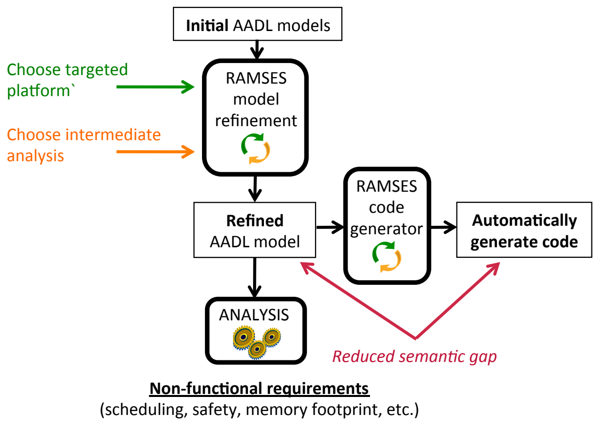

Initial AADL models
Choose targeted platform`
Choose intermediate analysis
RAMSES model refinement
Refined AADL model
RAMSES code generator
Automatically generate code
ANALYSIS
Reduced semantic gap
Non-functional requirements
(scheduling, safety, memory footprint, etc.)

Fig. 9. RAMSES Overview.

Then, RAMSES transforms the input model into a refined model: abstract AADL elements such as remote connections, modes, health-monitoring configuration, and local connections are transformed into tasks and shared variables. This intermediate model is then used for both code generation, and analysis of NFPs. In particular, we have shown in [41] how AADL Inspector (as an interface to Cheddar) could be combined with RAMSES in order to provide a model-based evaluation of tasks response times while taking into account the timing overheads due to code generation.

This approach has been experimented on a use case inspired from the railway domain. These experiments, based on ARINC653 operating system, have shown that overhead due to tasks and partitions communication was not negligible and required updates on the scheduling analysis after code generation.

From a technical viewpoint, integration of AADL Inspector and RAMSES was quite easy since both tools rely on AADL: a standardized ADL. As a consequence, AADL Inspector could be used both for scheduling analysis on the input model and on the refined models. However, scheduling analysis of intermediate models produced by RAMSES required to evaluate worst case execution times of the generated code. This was done in RAMSES by analyzing the intermediate AADL models and generating new AADL models for timing analysis. Depending on the complexity of the behavior expressed in the input model, this analysis can be computation demanding as it requires to analyze several scheduling scenarii [41]. AADL Inspector was then launched as a command line tool in order to automate the computation of tasks worst case response time in each scheduling scenario. Launching AADL Inspector with its graphical interface was also practical in order to verify the content of models used for analysis.

The approach implemented in RAMSES has also been experimented for the evaluation of other NFPs (such as memory consumption, and reliability) and other model transformations such as safety and security design patterns. This work is being pursued towards the composition of model transformations, either as transformation chains [42] or using higher order transformations [43].

VII. Conclusion

In this article, we have introduced Cheddar, a scheduling analysis tool that can be used to teach real-time scheduling analysis. We have presented Cheddar ADL, the input language of Cheddar allowing users to express the design of their real-time systems to analyze. We have also presented some of the analysis features of the tools: the tools implements the main concept and analysis methods about real-time scheduling that are usually part of real-time systems Master-level curriculum [44]. We have given few details about Cheddar implementation and the use of a model driven approach for such a purpose. Finally, we have shown how to integrate a scheduling analysis tools as Cheddar in two software engineering toolsets. This may help students to understand how scheduling analysis can take place in a software engineering process. To complete this presentation, this article is extended by an annex where teachers may find a sample of hand-outs they can use to illustrate real-time scheduling theory with their students.

VIII. 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] 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, no. 4, pp. 1–8, Dec. 2004.

[2] J. Stankovic, S. H. Son, J. Hansson *et al.*, “Misconceptions about real-time databases,” *Computer*, vol. 32, no. 6, pp. 29–36, 1999.

[3] L. Sha, T. Abdelzaher, K.-E. Årzén, A. Cervin, T. Baker, A. Burns, G. Buttazzo, M. Caccamo, J. Lehoczky, and A. K. Mok, “Real time scheduling theory: A historical perspective,” *Real-time systems*, vol. 28, no. 2-3, pp. 101–155, 2004.

[4] 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, no. 1, pp. 46–61, January 1973.

[5] J. Stankovic, M. Spuri, M. D. Natale, G. C. Buttazzo *et al.*, “Implications of classical scheduling results for real-time systems,” *Computer*, vol. 28, no. 6, pp. 16–25, 1995.

[6] L. Cucu and J. Goossens, “Feasibility intervals for fixed-priority real-time scheduling on uniform multiprocessors,” in *Emerging Technologies and Factory Automation, 2006. ETFA’06. IEEE Conference on*. IEEE, 2006, pp. 397–404.

[7] M. Harbour, J. Gutierrez, J. Drake, P. Martinez, and J. Palencia, “Modeling distributed real-time systems with MAST 2,” *Journal of Systems Architecture*, vol. 59, no. 6, pp. 331–340, Jun, 2013.

[8] T.-P. S. Inc., “Tri-pacific software inc. : RAPID-RMA,” 2014, http://www.tripac.com/rapid-rma.

[9] R. Henia, A. Hamann, M. Jersak, R. Racu, K. Richter, and R. Ernst, “System level performance analysis - the SymTA/s approach,” *IEE Proceedings - Computers and Digital Techniques*, vol. 152, no. 2, p. 148, Mar, 2005.

[10] 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.

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

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

[13] 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.

[14] 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.

[15] D. C. Schmidt, “Model-driven engineering,” *IEEE Computer*, vol. 39, no. 2, February 2006.

[16] P. H. Feiler and D. P. Gluch, *Model-Based Engineering with AADL: An Introduction to the SAE Architecture Analysis & Design Language*. Addison-Wesley, 2012.

[17] M. OMG, “Modeling and analysis of real-time and embedded systems,” *Object Management Group*, 2008.

[18] S. Li, F. Singhoff, S. Rubini, and M. Bourdellès, “Applicability of real-time schedulability analysis on a software radio protocol,” *ACM SIGAda Ada Letters*, vol. 32, no. 3, pp. 81–94, 2012-12.

[19] F. Singhoff, J. Legrand, L. Nana, and L. Marcé, “Scheduling and memory requirements analysis with aadl,” in *ACM SIGAda Ada Letters*, vol. 25, no. 4. ACM, 2005, pp. 1–10.

[20] C. Fotsing, F. Singhoff, A. Plantec, V. Gaudel, S. Rubini, S. Li, H. N. Tran, L. Lemarchand, P. Dissaux, and J. Legrand, “Cheddar architecture description language,” *Lab-STICC technical report.*, 2014.

[21] R. I. Davis and A. Burns, “A survey of hard real-time scheduling for multiprocessor systems,” *ACM Comput. Surv.*, vol. 43, no. 4, pp. 35:1–35:44, Oct. 2011. [Online]. Available: http://doi.acm.org/10.1145/1978802.1978814

[22] 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.

[23] 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.

[24] V. Gaudel, “Applicabilité des méthodes d’analyse et interopérabilité des outils de développement pour systèmes embarqués temps-réel critiques,” *Thèse de l’Université de Bretagne Occidentale, Brest, France*, Décembre 2014.

[25] S. Li, F. Singhoff, S. Rubini, and M. Bourdelles, “Extending schedulability tests of tree-shaped transactions for tdma radio protocols,” in *Emerging Technology and Factory Automation (ETFA), 2014 IEEE*. IEEE, 2014, pp. 1–8.

[26] A. Easwaran, I. Lee, O. Sokolsky, and S. Vestal, “A compositional scheduling framework for digital avionics systems,” in *Embedded and Real-Time Computing Systems and Applications, 2009. RTCSA’09. 15th IEEE International Conference on*. IEEE, 2009, pp. 371–380.

[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] J. Carpenter, S. Funk, P. Holman, A. Srinivasan, J. Anderson, and S. Baruah, “A categorization of real-time multiprocessor scheduling problems and algorithms,” *Handbook on scheduling algorithms, methods, and models*, pp. 30–1, 2004.

[29] N. C. Audsley, A. Burns, M. F. Richardson, and A. J. Wellings, “real-time scheduling: the deadline-monotonic approach,” in *in Proc. IEEE Workshop on Real-Time Operating Systems and Software*, 1991, pp. 133–137.

[30] N. Audsley, “Optimal priority assignment and feasibility of static priority tasks with arbitrary start times,” *Real-Time Systems*, 1991.

[31] H.-N. Tran, F. Singhoff, S. Rubini, and J. Boukhobza, “Addressing cache related preemption delay in fixed priority assignment,” in *Proceedings of the 2015 IEEE Emerging Technology and Factory Automation, ETFA 2015, Luxembourg, September, 2015*, 2015.

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

[33] 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,” *HDR de l’Université de Bretagne Occidentale, Brest, France*, Novembre 2012.

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

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

[36] 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*, ser. Ada-Europe’07. Berlin, Heidelberg: Springer-Verlag, 2007, pp. 180–191.

[37] 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*, ser. ECMDA-FA ’08. Berlin, Heidelberg: Springer-Verlag, 2008, pp. 432–443. [Online]. Available: http://dx.doi.org/10.1007/978-3-540-69100-6_31

[38] P. Dissaux, O. Marc, S. Rubini, C. Fotsing, V. Gaudel, F. Singhoff, A. Plantec, V. Nguyen-Hong, and H.-N. Tran, “The smart project: Multi-agent scheduling simulation of real-time architectures,” in *Embedded Real Time Software and Systems*, 2014.

[39] E. Conquet, M. Perrotin, P. Dissaux, T. Tsiodras, and J. Hugues, “The taste toolset: turning human designed heterogeneous systems into computer built homogeneous software.” *5th European Congress ERTSS Embedded Real Time Software and System. Toulouse, France.*, May 2010.

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

[41] E. Borde, S. Rahmoun, F. Cadoret, L. Pautet, F. Singhoff, and P. Dissaux, “Architecture models refinement for fine grain timing analysis of embedded systems,” in *25th IEEE International Symposium on Rapid System Prototyping (RSP), 2014*, Oct 2014, pp. 44–50.

[42] C. Castellanos, T. Vergnaud, E. Borde, T. Derive, and L. Pautet, “Automatic production of transformation chains using structural constraints on output models,” in *Proceedings of the 40th Euromicro Conference series on Software Engineering and Advanced Applications*, ser. SEAA’14, 2014.

[43] G. Loniewski, E. Borde, D. Blouin, and E. Insfran, “An automated approach for architectural model transformations,” in *22nd International Conference on Information Systems Development (ISD2013)*, Sevilla Spain, Sep. 2013.

[44] P. Caspi, A. Sangiovanni-Vincentelli, L. Almeida, A. Benveniste, B. Bouyssounouse, G. Buttazzo, I. Crnkovic, W. Damm, J. Engblom, G. Folher *et al.*, “Guidelines for a graduate curriculum on embedded software and systems,” pp. 587–611, 2005.