Applicability of Real-Time Schedulability Analysis on a Software Radio Protocol

Shuai Li+*, Frank Singhoff*, Stéphane Rubini*, Michel Bourdellès+

+THALES Communications & Security, 4 Avenue des Louvresses, 92622 Gennevilliers, France
*Lab-STICC/UMR 6285, UBO, UEB, 20 Avenue Le Gorgeu, 29200 Brest, France
{shuai.li,michel.bourdelles}@fr.thalesgroup.com, {rubini,singhoff}@univ-brest.fr

ABSTRACT

In this paper, we present our experience on integrating timing constraint verification and analysis, by using the real-time scheduling theory, in an industrial context. The verification process has been integrated into a design flow at THALES Communications & Security. We focus our work on Software Radio Protocols (SRP). We have used Model-Driven Engineering technologies and the Cheddar schedulability analysis tool for our experiment. We show how we have modeled a complete SRP in UML MARTE, a profile for real-time embedded systems, before using model transformation to extract information for schedulability analysis with Cheddar.

Categories and Subject Descriptors

D2.4 [**Software Engineering**]: Software/Program Verification—*Validation*

General Terms

Performance, Reliability, Verification

Keywords

Real-Time Embedded System, Software Radio Protocol, Real-Time Scheduling, Non-Functional Properties, Model-Driven Engineering, UML, MARTE

1. INTRODUCTION

In this paper, we explore the modeling and the schedulability analysis of a Software Radio Protocol (SRP).

A SRP is a Real-Time Embedded System (RTES) in the telecommunication domain. As a RTES, a SRP has timing constraints. Schedulability analysis is thus necessary. This analysis is done by using the real-time scheduling theory.

This theory dates back to the 70s with the seminal Liu and Layland work [17]. Although the real-time scheduling theory is mature [36] the breakthrough in the industry is not significant [39]. We believe this reluctance is due to the lack of domain-specific guidelines for system engineers. Even though tools exist to ease RTES' schedulability analysis, we believe they must be integrated directly into the domain-specific design tools. Developing bridges between the design tools and the analysis tools is one such possibility to accomplish this goal.

In this paper we study the applicability of the real-time scheduling theory on a SRP. In our solution, we use models coupled with schedulability analysis tools. Modeling and Analysis of Real Time Embedded systems (MARTE) [26] is a UML profile dedicated to RTES analysis. Cheddar [38] is a real-time schedulability analysis tool based on feasibility tests and simulation. We present our work on SRP modeling in MARTE and its analysis with Cheddar. We see how this process can be integrated into an industrial design flow at THALES Communications & Security (TCS), based on a Model-Driven Engineering (MDE) approach [34].

The rest of the article is organized as follows. Some related works are presented in section 2. Section 3 exposes our work's context. Section 4 presents the SRP architecture properties for schedulability analysis. Section 5 lists the modeling and schedulability analysis requirements on our work. SRP modeling with MARTE is shown in section 6. Section 7 presents the Cheddar scheduling analysis tool. The mapping between a SRP MARTE model and Cheddar is given in section 8. The suggested solution is experimented on a SRP application in section 9 and we also give its evaluation in this section. Finally we conclude in section 10 and we list some future works.

2. RELATED WORKS

System performance analysis, and particularly schedulability analysis, has been extensively studied in the literature. The analysis flows that interest us consist of modeling the system with an Architecture Description Language (ADL) [20] and analyzing it with a schedulability analysis tool.

In [8], a system modeled in the Architecture Analysis & Design Language (AADL) [9] is analyzed with Cheddar. AADL is derived from MetaH and it is possible to model both the software and hardware architecture of a RTES with this modeling language. Although AADL has been defined as a generic ADL, it is mostly used in the avionic domain due to its historical origins.

In [16], the authors explore timing analysis with an AUTOSAR-based [3] modeling language. The commercial SymTA/S [44] analysis tool is used for schedulability analysis. SymTA/S is based on compositional scheduling. Although less accu-

HILT'12, December 2–6, 2012, Boston, Massachusetts, USA.

rate in some cases, since its does not have a global view of the system, this technique is well adapted to system scalability [12]. The AUTOSAR ADL is dedicated to the automotive world. At the same time SymTA/S is tuned for the automotive industry, supporting typical domain specific buses like CAN or FlexRay.

EAST-ADL2 [7] has been extended with MARTE for schedulability analysis. Although the domain-specific language EAST-ADL2 uses a generic modeling language like MARTE for its schedulability analysis extensions, the suggested approach is applied on automotive systems.

Other works bridge MARTE system models with schedulability analysis tools.

RapidRMA [13] is a tool developed and commercialized by Tri-pacific Software. It is integrated into IBM's Rational Software Architect (RSA) [42] which supports the MARTE profile. In the RapidRMA model, software resources are mapped onto hardware resources. The analysis tool features rate monotonic and deadline monotonic analysis [17].

Experimentation on using MARTE for schedulability analysis has been done through the MARTE to MAST [19] transformation tool. MAST [11] is an open-source schedulability analysis tool developed by the University of Cantabria. MAST allows to verify if the system will meet its timing requirements while also offering the possibility to see how close the system is to meeting these requirements. MAST main features are worst-case response time analysis, blocking time calculation, sensitivity analysis, and optimized priority assignment techniques. The MARTE model for MAST focuses on activity diagrams, transactions and activity triggers. This is well suited for end-to-end flow analysis [47].

As explained in section 6, our MARTE model is different from the MARTE model used for MAST. We describe the architecture and we map it to resources. This is similar to the Rapid-RMA approach.

To our knowledge, no work has been done on the applicability of a MARTE/Cheddar-based approach for schedulability analysis in the specific SRP domain.

In the following sections we present the current modeling technologies and we characterize the SRP system in order to establish requirements on the modeling language and schedulability analysis tool.

3. CONTEXT OF THE WORK

The current modeling language used at THALES Communications & Security for SRP development is MyCCM [46], an implementation of the CORBA Component Model (CCM) [23]. MyCCM uses a component-based approach. The SRP is modeled as inter-connected components that communicate through their ports implementing an interface. In MyCCM, it is possible to define a deployment plan by allocating the functional components onto threads and processes. The model is used to generate wrapper code for the components, i.e. Interface Definition Language (IDL) files and Component Deployment Plan (CDP) files. PrismTech proposes solutions for CCM (SpectraCX [31]) and the modelers are used at THALES. This model-driven development tool is built above the Eclipse framework.

The SRP modeling guideline for schedulability analysis suggested in this paper, is based on the European ITEA project Validation-drivEn design foR component-baseD architEctures (VERDE) [48]. We use in particular the functional entities modeling from VERDE. Figure 1 highlights the analysis parts in VERDE that we have adapted for Cheddar. In this figure our contribution concerns the schedulability analysis.

4. SOFTWARE RADIO PROTOCOL

A Software Radio Protocol (SRP) is a radio protocol embedded in a radio equipment. A SRP is composed of one or several applications called waveforms that are running on a platform.

A waveform is composed of different software components that manage radio channel access technologies, radio protocol and routing. The waveform design may be separated into several functional layers following coarsely the OSI model [50]:

- PHY: Synchronization, data transmission/reception
- LINK: Protocol management
- NET: User packet handling

The platform offers services to the waveform. The platform is composed of:

- the Operating System (OS)
- the Hardware (HW)

Since the waveform is generally developed independently of the platform for interoperability issues, a waveform's performance analysis must be carried out on each platform. Performance analysis must investigate timing constraints and processor utilization.

In the next paragraphs we establish a list of the entities composing a SRP's waveform and platform. We also characterize the SRP's architecture from a schedulability analysis point of view.

4.1 Entities of a SRP

The SRP entities have been gathered into three groups: functional entities, OS entities and hardware entities.

4.1.1 Functional entities

The waveform is composed of functional entities which are the following:

Functional Component A set of functionalities and services that can be connected with other components sharing its interfaces.

Port A component's port is the entry-point for other components. Through a port a component requires or provides services. A port is typed with an interface.

Interface An interface regroups operations. Operations are the services and functionalities implemented by the component.

Datatype and Enumeration Operations have parameters that are typed by Datatypes, Primitives (a specific type of Datatype), and Enumerations.

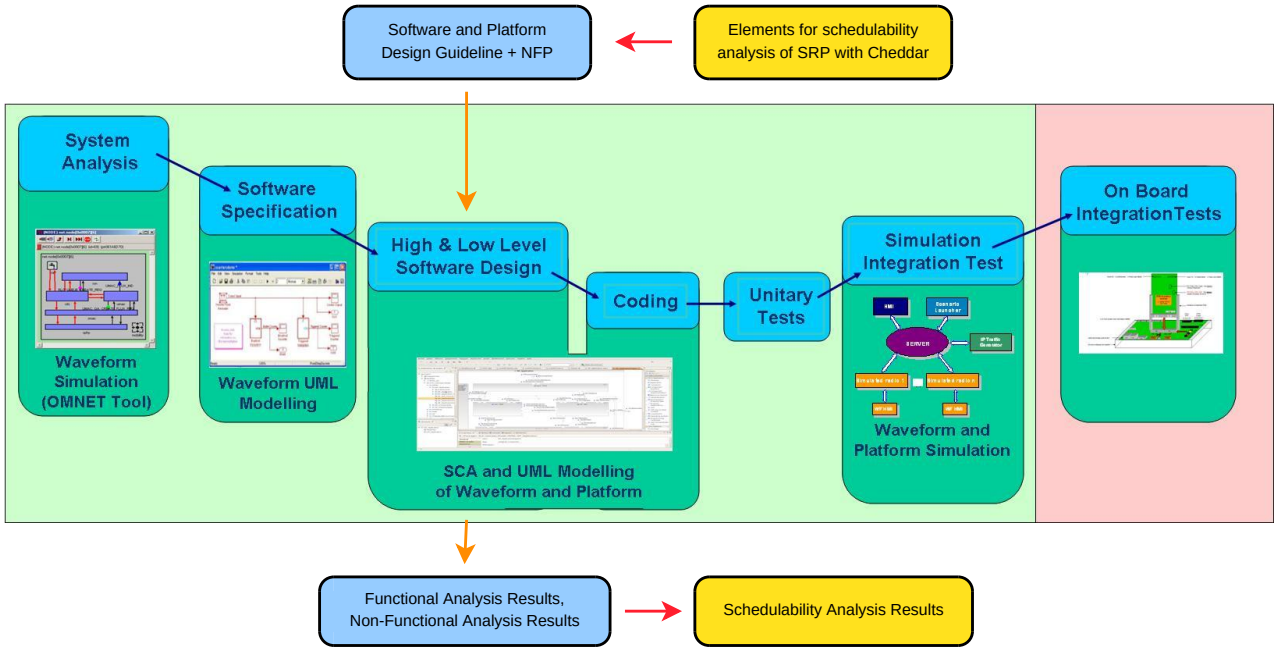

Figure 1: Contribution to THALES Communications & Security Design Flow

4.1.2 *Operating system entities*

OS entities are elements composing the runtime for the waveforms. These entities are executable software units, memory spaces, critical resources, and their managers. The OS entities needed for analysis are:

Thread A software unit of execution in the OS.

Process Threads are allocated onto processes. While being a unit of execution itself, a process is defined as a memory space in our work.

Scheduler The different units of execution are scheduled according to a policy defined by the scheduler in the OS. Note that several schedulers may be present in the model.

Shared Resource A shared resource is one that is accessed concurrently by different units of execution. They are protected with a protocol implemented in the OS.

Buffer A buffer is a resource used in the producer/consumer pattern. E.g. one thread writes periodically into a FIFO buffer while another thread reads periodically from it.

4.1.3 *Hardware entities*

Key hardware resources also need to be identified for the analysis tools. The degree of details of the hardware model also depends on the schedulability analysis tool. In a first step, we consider a simple hardware model composed of the following entities:

Processor The Central Processing Unit (CPU) is a key element in the platform model as it executes the threads.

Core A core is a computing resource. A processor may contain one or several cores (monocore and multicore processors).

Cache A cache is a memory shared by processors and cores. L1 caches are shared by cores and L2 caches are shared by processors.

4.2 Architecture Properties for Schedulability Analysis

Several works have been done on schedulability analysis at the PHY layer [45]. Threads have a periodic nature and the application is a signal processing one. With such properties, the analysis is made much more easier because signal processing and multimedia applications can be abstracted as data-flow applications. Unfortunately what stands true at the PHY layer is not necessarily true at LINK and NET layers. The architecture properties related to schedulability analysis are:

Heterogeneous Platforms SRP platforms are heterogeneous and threads run on different partitions, on different processors. This partitioning is due, in particular, to security issues (separation of critical and non-critical data) [5].

Scheduling Policy Scheduling policies are fixed priority-based and preemptive for the LINK and NET layers.

Execution time Execution times are variable and non-cyclic. E.g. Quality of Service (QoS) and the state in which the system is, influence the threads' execution time.

Thread Activation Pattern Not all threads in the LINK and NET layers are triggered periodically. Some threads may also be activated on event arrival. Threads may have a latency between its activation and its real execution. Threads in the system are not necessarily activated at the same time.

Thread Deadline Some threads are activated periodically but their execution may span over several periods, e.g. non-critical threads for radio stations synchronization.

Data flow and control flow Several data and control flows transit in the system, going through several functional components. Verifying that these flows finish before a defined deadline is a key issue.

Thread communication Communication between threads may be synchronous or asynchronous.

Resource Standard protocols (e.g. PCP, PIP [37]) for mutual exclusion are used to protect shared resources access between threads. This depends on the used platform and the protocols may not be used or implemented on certain platforms.

Following our work's context presented in section 3 and the SRP presentation in section 4, we have established the requirements on the modeling and schedulability analysis of a SRP.

5. MODELING AND ANALYSIS REQUIREMENTS

Table 1 lists the modeling requirements. These modeling requirements meet the current software engineering practices at THALES, extended with the modeling of elements required for schedulability analysis. Table 2 lists the schedulability tool requirements.

Table 1: Modeling Requirements

Ref.	Description
MREQ1	Component-based model: Due to the nature of MyCCM, the waveform and platform models are designed with a component approach.
MREQ2	Modular platform model: Platform entities must be separated into several layers for modularity.
MREQ3	Push-button logic: The proposed schedulability analysis method must not need extended expertise in the real-time scheduling theory.

6. MODELING A SRP WITH MARTE

In the previous sections, we exposed the modeling context of the work, the architecture entities and its schedulability properties. We now present our contribution on SRP modeling for schedulability analysis with the UML MARTE profile.

A UML profile is a mechanism that extends UML for a specific domain. As a profile is a generic extension of the base modeling language, it does not contradict the original language's semantics. Concretely, a UML profile is implemented as a set of stereotypes, tag values, and constraints applied to elements inside UML, e.g. classes, operations, activities, interactions. A stereotype is a mechanism to extend the UML vocabulary by defining a UML element as something specific for a domain, with specific attributes.

The MARTE UML profile extends UML to add support for modeling elements and concepts that are specific to RTES. The MARTE profile is composed of several sub-profiles to represent different concepts of a RTES at different levels of abstraction of the real system.

Table 2: Schedulability Analysis Tool Requirements

Ref.	Description
SREQ1	Heterogeneous multiprocessor platforms must be supported.
SREQ2	Processor context switch overhead must be supported.
SREQ3	Hierarchical scheduling must be supported. Spatial distribution (allocation of threads on processes on processors) must be supported.
SREQ4	Highest Priority First scheduling policies must be implemented.
SREQ5	User can express specific schedulers.
SREQ6	Specific priorities assignment can be implemented.
SREQ7	User can express specific thread parameters.
SREQ8	Variable execution times can be specified.
SREQ9	Periodic, sporadic and aperiodic dispatching protocol must be supported.
SREQ10	Dynamic/static jitters and offsets for task release time must be supported [27].
SREQ11	User can express specific thread release time.
SREQ12	User can assign any values for thread deadlines.
SREQ13	End-to-end timing analysis must be implemented.
SREQ14	Thread precedencies must be supported.
SREQ15	Blocking and non-blocking communications must be supported.
SREQ16	Shared resources between threads must be supported.
SREQ17	Standard shared resource protocols must be implemented.
SREQ18	Resource access duration by threads can be specified.

As MARTE is a big profile, with scarce guidelines for the designer [2] [32], it can become confusing to decide how to model a system in MARTE. Especially since many methods are possible and best practices are not always evident. It is thus important to define a guideline to model the RTES with MARTE for a specific kind of analysis. MARTE acts as a framework to define the modeling guideline.

In section 4.1, we have decided to separate the entities into three categories: the functional waveform, the OS, and the hardware. Figure 2 gives the corresponding model packages. In the following paragraphs we expose our guideline for modeling the entities with MARTE sub-profiles. The sub-profiles are presented first before we show how we use them. The associations between the different entities are also explained at the end.

6.1 Waveform Modeling

The Generic Component Model (GCM) sub-profile addresses designer needs in applying the component-based model paradigm. Its main contribution to our model relies in the annotations it provides to model the component's ports.

High Level Application Model (HLAM) is a sub-profile that provides designer with facilities to annotate their model to represent high level artifacts such as set of functionalities in a block.

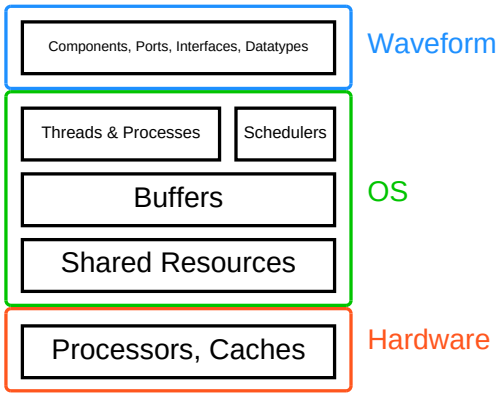

Figure 2: Model Packages

Figure 3 describes the stereotypes in the GCM and HLAM sub-profiles that we have used.

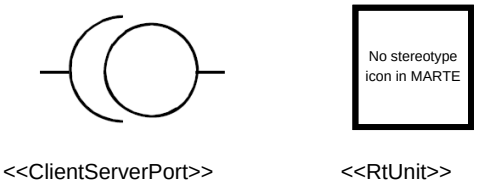

Figure 3: Applied GCM and HLAM Stereotypes

Table 3 maps the different functional entities to their stereotyped UML elements in MARTE.

Table 3: Waveform Modeling in UML and MARTE

Functional Entity	MARTE Model
Functional Component	**Component** stereotyped ⟨⟨*RtUnit*⟩⟩ from **HLAM**
Port	**Port** stereotyped ⟨⟨*ClientServerPort*⟩⟩ from **GCM**
Interface	**Class** stereotyped ⟨⟨*Interface*⟩⟩ from **UML**
Datatype	**Class** stereotyped ⟨⟨*Datatype*⟩⟩ from **UML**
Enumeration	**Enumeration** from **UML**

6.2 Operating System Modeling

The General Resource Modeling (GRM) and Software Resource Modeling (SRM) sub-profiles have been used to model the OS entities. GRM enables execution platform modeling and provides the foundations needed for a more refined modeling of both hardware (Hardware Resource Modeling) and software (Software Resource Modeling) resources.

Designing a multitasking application is based on a Real-Time Operating System (RTOS) that offers resources through Application Programming Interfaces (API). The SRM sub-profile offers artifacts to model the support provided by the RTOS, i.e. its API. SRM has been built upon observing elements present in several standard RTOS API (POSIX, OSEK/VDX, ARINC 653) and RTOS (VxWorks, RTAI, QNX).

Figure 4 enumerates the GRM and SRM stereotypes used in the OS resources modeling.

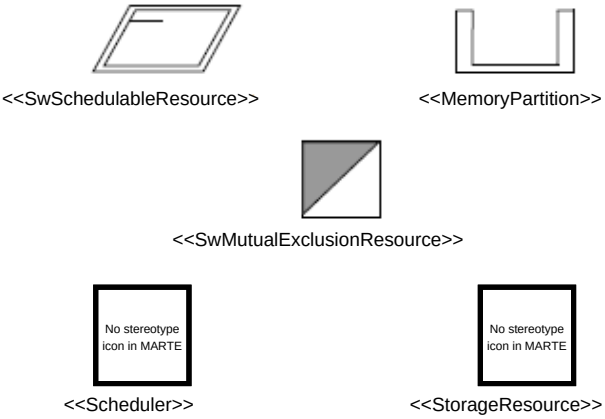

Figure 4: Applied GRM and SRM Stereotypes

Table 4 maps the different OS resources to their stereotyped UML elements in MARTE.

Table 4: OS Entities to MARTE

OS Entity	MARTE Model
Thread	**Component** stereotyped ⟨⟨*SwSchedulableResource*⟩⟩ from **SRM**
Process	**Component** stereotyped ⟨⟨*MemoryPartition*⟩⟩ from **SRM**
Shared Resource	**Class** stereotyped ⟨⟨*SwMutualExclusionResource*⟩⟩ from **SRM**
Buffer	**Component** stereotyped ⟨⟨*StorageResource*⟩⟩ from **GRM**
Scheduler	**Component** stereotyped ⟨⟨*Scheduler*⟩⟩ from **GRM**

6.3 Hardware Modeling

The Hardware Resource Modeling (HRM) sub-profile has been used to model hardware entities. The HRM sub-profile provides several stereotypes to model the platform hardware through three different views: a high-level architectural view, a specialized view and a detailed physical view. We have decided to stay at a high-level architectural view with enough annotations needed for analysis. This is because our goal is to have a first estimation of the application's performance on a platform, without the need for the system engineer to supply too much detail on the hardware components.

Figure 5 enumerates the stereotypes in the HRM sub-profile needed for our model.

Table 5 maps the different hardware resources to their stereotyped UML elements in MARTE.

6.4 Associations Between Entities

Specific UML associations (e.g. Abstraction, Usage) stereotyped with MARTE are used to represent the relationships between the entities of each category. Each association has

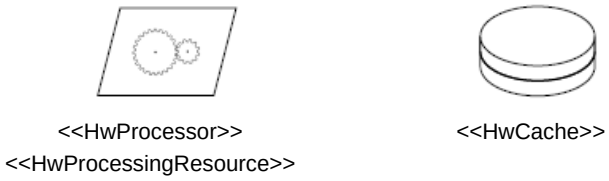

Figure 5: Applied HRM Stereotypes

Table 5: Hardware Resources to MARTE

HW Entity	MARTE Model
Processor	**Component** stereotyped $\langle\langle HwProcessor \rangle\rangle$ from **HRM**
L2 Cache	**Property** of the processor typed with the **Component** stereotyped $\langle\langle HwCache \rangle\rangle$ from **HRM**
Core	**Property** of the processor typed with the **Component** stereotyped $\langle\langle HwComputingResource \rangle\rangle$ from **HRM**
L1 Cache	**Property** of the core typed with the **Component** stereotyped $\langle\langle HwCache \rangle\rangle$ from **HRM**

a client and a supplier depending on the role of each end. UML interactions are also used to represent the relationships, which are shown in different views:

Spatial Distribution View The spatial distribution view links the functional entities to the OS entities and the OS entities to the hardware entities. Concretely the spatial distribution view shows how entities at different levels are allocated onto one another. An allocation is a UML Abstraction stereotyped $\langle\langle Allocate \rangle\rangle$ from MARTE. The UML Abstraction has a client (the element that is allocated) and a supplier (the element that is allocated onto). The different possible allocations are listed in Table 6.

Critical Resource Usage View The critical resource usage view shows how threads use shared resources and buffers. A resource usage is a UML Usage stereotyped $\langle\langle ResourceUsage \rangle\rangle$ from the MARTE GRM subprofile. The UML Usage has a client (the resource user) and a supplier (the resource used). Through the $\langle\langle ResourceUsage \rangle\rangle$ stereotype, we can specify the resource usage time. We use a UML Constraint stereotyped $\langle\langle TimedConstraint \rangle\rangle$ from the MARTE Time sub-profile to specify the resource usage start time in the thread's execution time. The UML Constraint constrains the UML Usage.

Dependency View Communications between entities are represented through a dependency view that is a communication diagram in UML. In the communication diagram, a UML Lifelines represents a thread or a buffer. A dependency is established between two entities if there is a UML Message between the two UML Lifelines representing the entities. The direction of the message matters while establishing the dependent and dependable roles.

Table 6: Allocations

Client	Supplier	Description
Component	Thread	The functional component contains a thread triggered by events.
Port	Thread	The port's operations are handled by a thread triggered by calls to any of the operations.
Operation	Thread	A specific operation is handled by a thread triggered by a call to the operation.
Thread	Process	A thread is executed in the context of a memory partition (process).
Shared Resource	Process	A shared resource is located in a memory partition.
Buffer	Process	A buffer is located in a memory partition.
Process	Processor	The threads of a process are executed on a processor. At least one process exists for each processor.

7. CHEDDAR, AN ADA SCHEDULABILITY ANALYSIS TOOL

Cheddar is an open-source schedulability and timing constraints analysis tool developed and maintained by the University of Western Brittany/Lab-STICC since 2001. From 2008 and onward, Ellidiss Technologies participates in the tool's industrial distribution by providing industrial support [8].

System and architecture models can be analyzed in Cheddar using the AADL standard or the tool's own design language. The Cheddar analysis environment includes two main elements: a graphical editor and a core analysis framework. The graphical editor helps the designer to describe the real-time system architecture to be analyzed. The core analysis framework includes the different scheduling algorithms, feasibility tests, and analysis found in the real-time scheduling literature.

The core analysis framework is composed of a simulation engine and a feasibility tests analyzer. It can be seen as a library of tests and analysis methods that the designer can choose to verify timing constraints.

When no analysis is applicable to a system's task model and scheduling policy, this does not mean that important properties of the system cannot be found, e.g. the system's non-schedulability. The Cheddar simulator provides features for these kind of systems. Furthermore the simulator can be coupled with another one of Cheddar's key feature: the modeling of user-defined parameters for tasks and the implementation of user-defined schedulers. The system can then be analyzed by simulations.

Cheddar is partly generated from a meta-model with a model-driven process [30]. The Cheddar meta-model is defined in EXPRESS [24], a modeling language. Figure 6 shows the Cheddar meta-model. For the sake of space and

clarity, class properties and child classes are not present in the diagram. Cheddar is implemented in Ada and runs on Unix-based OS (Solaris, Debian Linux) and Windows.

From the Cheddar model, we have defined the transformation rules between MARTE and Cheddar in the following section.

8. MARTE TO CHEDDAR TRANSFORMATION

Model transformation [35] is part of MDE and it ensures the consistency between different models. In model transformation an appropriate model (i.e. conform to the input meta-model) is used as input for the transformation tool. The tool's output is a model conform to the output meta-model. The output model represents complementary information of the system on a different level, in a different view or for a different use.

Our work is based on the first transformation [18] developed by THALES. This transformation was implemented as a IBM RSA plug-in. Several Cheddar entities are not taken into account in this work (e.g. multicore processor) and we have modified it for SRP analysis.

Table 7 lists the mapping of MARTE entities to Cheddar entities.

Table 7: MARTE to Cheddar Mapping

MARTE	Cheddar
$\langle\langle HwProcessor \rangle\rangle$	Processor
$\langle\langle HwProcessingResource \rangle\rangle$	Core_Unit
$\langle\langle HwCache \rangle\rangle$	Cache
Properties typed $\langle\langle HwCache \rangle\rangle$	Cache_System
$\langle\langle Scheduler \rangle\rangle$	Scheduler
$\langle\langle MemoryPartition \rangle\rangle$	Address_Space
$\langle\langle StorageResource \rangle\rangle$	Buffer
$\langle\langle SwMutualExclusionResource \rangle\rangle$	Resource
$\langle\langle SwSchedulableResource \rangle\rangle$	Task
$\langle\langle Allocation \rangle\rangle$	Address_Space_Name property of Task, Resource, Buffer. Cpu_Name property of Address_Space, Task, Resource, Buffer
$\langle\langle ResourceUsage \rangle\rangle$	Task_Tab property of Resource (list of tasks using the resource). Roles property of Buffer (list of roles for the buffer)
$\langle\langle GaScenario \rangle\rangle$'s Message (UML)	Dependency

The MARTE to Cheddar transformation has been implemented as an Eclipse plug-in using the Papyrus [28] modeler and its MARTE implementation. The size of the Eclipse modeling ecosystem is large, with numerous tools and modelers implemented using the Eclipse Modeling Framework (EMF) [41], [28], [42], [6], [40]. Industrial modeling tools like SpectraCX [31] are also built upon Eclipse. We choose an Eclipse-based implementation for integration with tools used at THALES. Experiments on the system's modeling and transformation to Cheddar are presented in the following section.

9. EXPERIMENT

In order to test our SRP schedulability analysis method in an industrial context, we decided to experiment with a system's simulation developed at THALES.

9.1 Software Radio Protocol Case-Study

The IP TDMA-Based Protocol (ITBP) protocol has been designed to expose and demonstrate the main constraints encountered in modern protocols designed at THALES. ITBP is IP-based and uses Time Division Multiple Access (TDMA) [4] for radio channel separation. In TDMA, time is divided into several time slots. At each slot a station has an action to perform and the mapping between the actions and the time slots is called a TDMA allocation. As explained in section 4, the system's waveform design follows the OSI model. Figure 7 shows the different layers in the waveform.

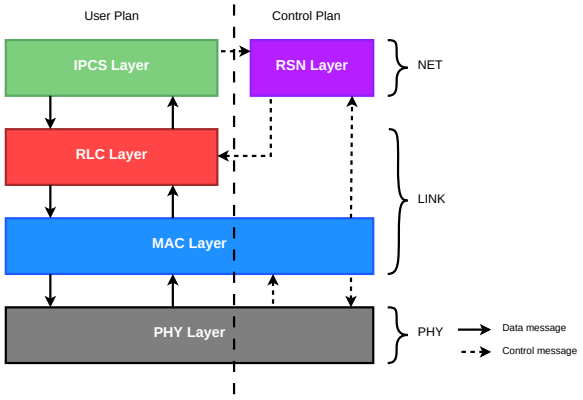

Figure 7: ITBP Layers and Sub-layers

Several data (Protocol Data Unit (PDU)) and control flows transit through the waveform's different layers. The data and control flows depend on the type of the time slot being processed, and the system's state. Under the experimental conditions, the waveform's functional behavior is the following:

1. IP Packets arrive from the IP stack and are segmented into RLC PDU before being stored in the RLC layer.
2. RLC PDU are handled and segmented into MAC PDU before being stored in the MAC.
3. A tick from synchronization source (e.g. GPS) indicates the start of a new slot.
4. The current slot type and next one are checked according to the slots allocation.
5. PDU stored in the MAC is passed to the PHY for transmission over air on the next slot.
6. Received data is transferred from the PHY to the IPCS.
7. The system is prepared for the next slot.

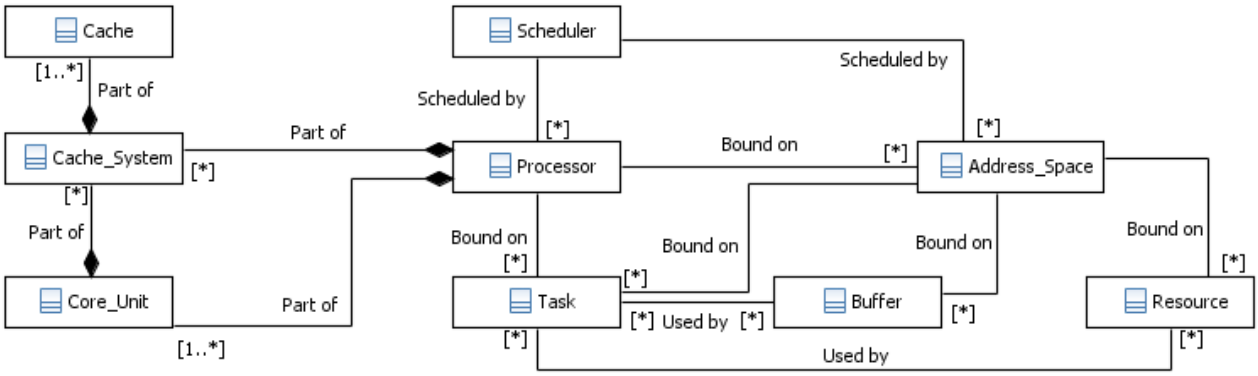

Figure 6: Cheddar Meta-Model

The waveform is deployed on a platform with five threads. Figure 8 summarizes the threads' shared resources and communication dependencies. The threads' detailed properties are given in table 8. Table 9 shows the shared resources and their usage. The shared resources are protected by Linux's default FIFO mutex protocol, i.e. there is not protection against priority inversion [37].

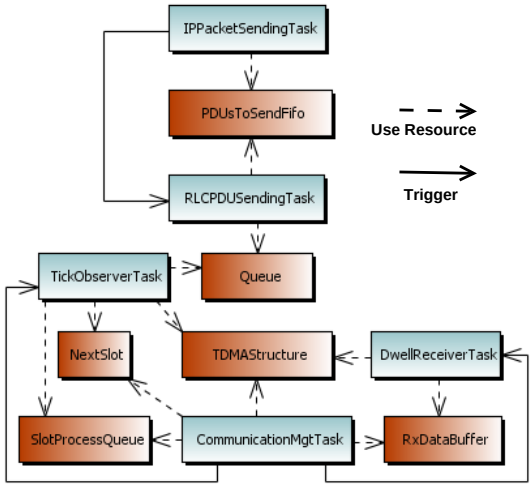

Figure 8: ITBP Threads and Resources Dependencies

IPPacketSendingTask This thread reads from the IP stack and transmits packets to the *PDUsToSendFifo*. All station routes to other stations have already been established previously.

RLCPDUSendingTask This thread is awaken by the *IPPacketSendingTask*, each time there are data in the *PDUsToSendFifo*. It segments the PDU and transmits them to the *Queue*.

CommunicationManagementTask[1] This thread is activated by a tick from the exterior synchronization source. It processes the current slot and verifies its correctness by checking the *TDMAStructure*. Afterwards it updates *NextSlot* with the next slot id, and wakes the *TickObserverTask* thread. On reception it stores received data in the *RxDataBuffer*. All slot allocations are static for the experiment.

TickObserverTask[2] This thread is awaken by the *CommunicationManagementTask*. It reads *NextSlot* and configures PHY for the next slot. It also transmits the data to be send over air on the next slot (stored originally in the *Queue*) to the *SlotProcessQueue*. The *TDMAStructure* is checked for slot consistency. These operations have to be done before the next slot.

DwellReceiverTask This thread is awaken by the *CommunicationManagementTask*, each time there are data in the *RxDataBuffer*. It processes the data and sends it to the IP stack. The *TDMAStructure* is checked for slot consistency.

9.2 Experiment Setup

The setup for the ITBP simulation that we have used to study its schedulability is shown in figure 9.

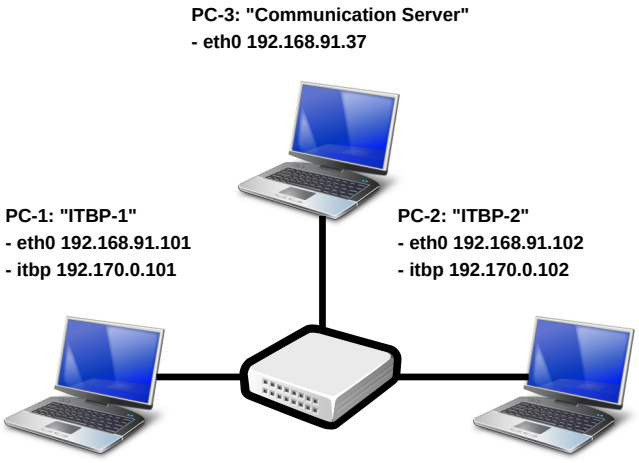

Figure 9: Experiment Setup

[1]May be abbreviated as "CommMgtTask" for the rest of this paper.

[2]May be abbreviated as "TickObs" for the rest of this paper.

Table 8: ITBP Threads Properties

Thread	Property	Value
IPPacketSendingTask	Activation	Sporadic
	Inter-arrival	6000μs
	Priority	20
	Capacity	207μs
	Deadline	None
RLCPDUSendingTask	Activation	Sporadic
	Inter-arrival	6000μs
	Priority	10
	Capacity	1725μs
	Deadline	None
TickObserverTask	Activation	Periodic
	Period	5000μs
	Priority	10
	Capacity	773μs
	Deadline	5000μs
DwellReceiverTask	Activation	Sporadic
	Inter-arrival	10000μs
	Priority	10
	Capacity	917μs
	Deadline	None
CommMgtTask	Activation	Periodic
	Period	5000μs
	Priority	40
	Capacity	1115μs
	Deadline	5000μs

Three PC are connected through a central hub. PC-1 and PC-2 each run a ITBP protocol while PC-3 (called the "Communication Server") plays the role of a network clock, i.e. it broadcasts ticks to PC-1 and PC-2. PC-3 also manages data routing, i.e. PC-1 and PC-2 send their packets to PC-3 which redirects the packets to the correct station.

On PC-1 and PC-2 the ITBP protocol is launched as a virtual network interface (called *itbp*) that reads packets coming from the main *eth0* interface. The packets coming from *eth0* are in the ITBP proprietary format. An user application reads the assembled IP packets from the virtual *itbp* interface by using the virtual address 192.170.0.x. It also sends user data to another station by sending to a virtual address. For our experiment we used VLC [49] to stream a video between PC-1 and PC-2.

The ITBP protocol is run on a quad-core PC with Ubuntu Linux. At start up, the *taskset* command is used to set the PID 1 process core-affinity to 0-2. This core-affinity is inherited by all child processes spawn by process 1. Core 3 is then available to run ITBP alone. Only system processes still persist on core 3 as these processes are duplicated on all cores during kernel boot.

The ITBP threads are POSIX threads with properties SCHED_ FIFO and PTHREAD_SCOPE_SYSTEM. The first property ensures that the threads are scheduled with a fixed priority-based scheduler, i.e. they can only be preempted by other SCHED_FIFO threads and system threads. By setting the threads' scope to system scope, they are scheduled as system threads.

9.3 MARTE Model

The system's MARTE model's diagrams are shown in the appendix section. With respect to section 6, the waveform has been modeled as inter-connected components in UML MARTE. Figure 14 gives a cut view of the MAC sub-layer.

Figure 15 shows the spatial distribution view with elements present in the execution platform (OS and hardware).

Figure 12 shows the shared resource usage by the threads.

Figure 13 shows the threads communication dependencies through the UML communication diagram.

Table 9: ITBP Shared Resources

Resource	Used by	Start (μs)	End (μs)
PDUsToSendFifo	IPPacketSendingTask	30	34
	RLCPDUSendingTask	1	3
Queue	RLCPDUSendingTask	127	1624
	TickObserverTask	16	142
TDMAStructure	DwellReceiverTask	65	67
	TickObserverTask	4	6
	CommMgtTask	109	111
RxDataBuffer	DwellReceiverTask	1	27
	CommMgtTask	93	107
NextSlot	TickObserverTask	0	772
	CommMgtTask	329	330
SlotProcessQueue	TickObserverTask	84	90
	CommMgtTask	0	325
	CommMgtTask	708	758

9.4 Analysis Evaluation

In the following sections we present and analyze three significant results given by the conducted experiment. In order to evaluate Cheddar's analysis results on the described system, we used execution trace coming from code instrumentation [3].

9.4.1 Software Design Mistake

A first analysis was done by injecting the data in table 8 directly into the MARTE model and generating the matching Cheddar model. A model where all threads are periodic (sporadic threads are abstracted as periodic threads), start at the same time, and run at their full capacity (WCET), is generated. With this model, feasibility tests based on processor utilization and response time can be done. Table 10 shows the Worst Case Response Time (WCRT) given by Cheddar through feasibility tests. Figure 10 shows the measured response times distribution for *CommunicationManagementTask* and *TickObserverTask* (the threads with deadlines). When comparing the results, we see that the *TickObserverTask* exceeds the predictions. Note that the small *IPPacketSendingTask* measured response time is explained by the fact that we did not instrument system functions so we could not capture the exact arrival of an IP packet. As this thread does not have a hard deadline, this issue is not important for our experiment.

This difference can be explained by a software design mistake. The *CommunicationManagementTask*'s inactive time represents the time it waits for a message from the Communication Server (PC-3). During this time the other threads with lower priority can run. The *CommunicationManagementTask* still locks the *SlotProcessQueue* shared resource during its wait. This blocks in turn the *TickObserverTask*

[3]Time measurements were done by reading the counter value in the RDTSC Pentium register.

Table 10: Analyzed WCRT from Cheddar

Thread	WCRT
IPPacketSendingTask	1322
RLCPDUSendingTask	3047
CommMgtTask	1115
TickObserverTask	4737
DwellReceiverTask	2239

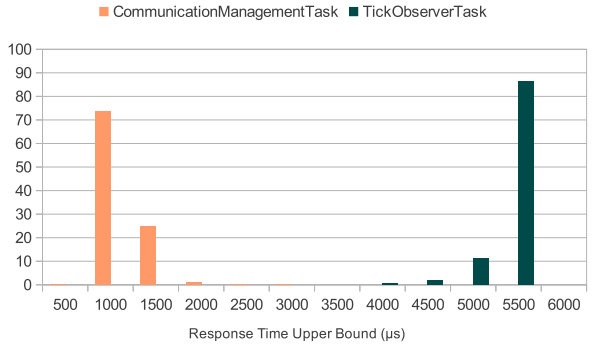

Figure 10: Measured Response Times: Each category represents an interval with the previous category as the lower bound and the current category as the upper bound. Each bar represents the percentage of the corresponding thread's measured response time in the category.

when it wants to access the resource. Since the wait time (equal more or less to the *CommunicationManagementTask*'s 5000μs period) is much greater than the threads' execution time, the *TickObserverTask* takes in average 5000μs to complete. As it cannot finish during the *CommunicationManagementTask*'s wait time, this latter thread is blocked when it wants to access the *NextSlot* shared resource on its next iteration. *TickObserverTask* thus impacts on the *CommunicationManagementTask*'s response time. Blocking a resource for any arbitrary time (e.g. during a thread's inactive time) can not be modeled directly with Cheddar.

We modify the original model in order to verify our assumption on the design mistake.

The modification consists in first applying the lowest priority to the *TickObserverTask*. This is not far from reality since: (1) this thread already has a lower priority than the *IPPacketSendingTask*, (2) *DwellReceiverTask* is always activated before in the scenario we analyze, and (3) *RLCPDUSendingTask* does not have a hard deadline so it is more important for its execution time to reflect on the *TickObserverTask*'s response time.

The second step consists in adding a new thread, called *SleepTask* in the rest of the paper, that is higher in priority than the *TickObserverTask* but lower than the other threads. *SleepTask*'s execution time is computed with the following equation:

$$C(SleepTask) = P(CommMgtTask) - St(SleepTask) \quad (1)$$

Where $C(i)$ is the i thread's capacity, $P(i)$ is i thread's period, $St(i)$ is i thread's release time.

$St(SleepTask)$ is computed as the following:

$$\left(\sum_{x \in h_p(SleepTask)} C(x) \right) + S_{TickObs}(SlotProcessQueue) \quad (2)$$

Where $h_p(i)$ is the set of threads with higher priority than the i thread, and $S_i(r)$ is the time the i thread starts using shared resource r. With the data in table 8 and 9, we get $C(SleepTask) = 952\mu$s and $St(SleepTask) = 4048\mu$s. The *SleepTask* will thus preempt the *TickObserverTask* when it wants to access the *SlotProcessQueue*. The preemption lasts until the *CommunicationManagementTask* is ready to run again, thus simulating the fact that *SlotProcessQueue* is still locked during *CommunicationManagementTask*'s downtime.

The last step of the modification is to change the release time of the *SleepTask* so it preempts the *TickObserverTask* at the exact moment it wants to use the *SlotProcessQueue* shared resource. *TickObserverTask* and *DwellReceiverTask*'s release times have also been changed for more accurate computed WCRT. With this new model it is not possible to apply feasibility tests because of the different release times. We thus do a simulation on a time span of 2 slots, i.e. 10000μs. All threads are still kept periodic to make sure that they have the maximum impact when computing WCRT. Note that the modification only works for computing response times on a time span of 2 slots. *TickObserverTask* will start later in its second activation, having delayed *CommunicationManagementTask*, but *SleepTask*'s release time and capacity cannot be modified. Table 11 shows the computed WCRT from simulation.

Table 11: Simulated WCRT from Cheddar

Thread	WCRT
IPPacketSendingTask	1322
RLCPDUSendingTask	3964
CommMgtTask	3736
TickObserverTask	7619
DwellReceiverTask	2131

By comparing table 11 with the measured response times distribution in figure 10, we see that the results given by simulation, with the more realistic system model, are consistent. This validates our assumption on the design mistake.

The currently suggested MARTE modeling, very near the Cheddar model in its platform design, is limited since an extra modification needed to be done to represent the real system behavior. Furthermore Cheddar's own model is limited as, even with the modification, the simulated behavior only stands true for a limited time span in the case of our experiment.

9.4.2 Missed Deadline

From the simulation by Cheddar, we see that there is at least one case where the *TickObserverTask* misses its 5000μs deadline. This has also been observed in the application's execution, where the thread's maximum observed response time is 5387μs, the average being 5058μs. In reality the system still works because this is a soft deadline. Furthermore not all threads actually run at their WCET, i.e. lateness in certain threads may be compensated by threads running faster in the next slot. The only hard deadline concerns the

CommunicationManagementTask. As long as this thread's response time does not exceed 5000μs, the system works.

We have noticed that there are no feasibility tests implemented in Cheddar for the thread synchronization of our case-study. This justifies the use of the simulator.

9.4.3 Priority Inversion

Another interesting result comes from the Cheddar priority inversion detection tool. Cheddar detects *CommunicationManagementTask* has a priority inversion from 5329 to 7950. 5329 corresponds to the time at which *CommunicationManagementTask* wants to access *NextSlot* and we have:

$$\begin{aligned} & C(IPPacketSendingTask) \\ & + C(RLCPDUSendingTask) \\ & + C(TickObs) \\ & - S_{TickObs}(SlotProcessQueue) \\ & = 2621 = 7950 - 5329 \end{aligned} \tag{3}$$

We remind that in the modified task model we used, *TickObserverTask* has the lowest priority, this is why it is preempted by *RLCPDUSendingTask* too. From the execution traces on the real application, we see that this priority inversion does happen with the *IPPacketSendingTask*. The Gantt chart in figure 11 illustrates this issue. The priority inversion is not surprising as there is no protocol (e.g. PCP, PIP) implemented in un-patched Ubuntu Linux to prevent this phenomenon. The implementation thus need to be deployed on a platform with a real-time OS that supports priority-inversion prevention.

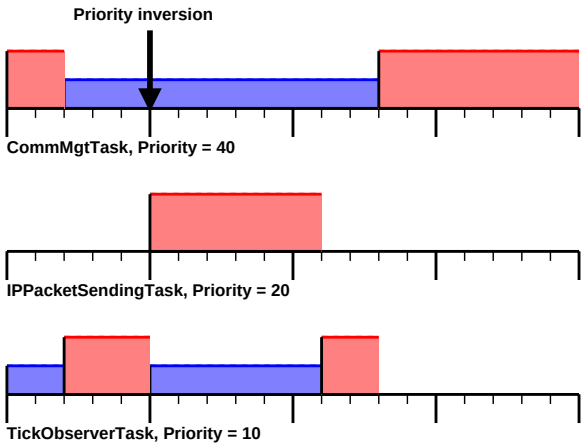

Figure 11: Priority Inversion: A high rectangle means the thread is running, a low rectangle that it is waiting.

9.4.4 Analysis Evaluation Conclusion

In conclusion of our experiment, we can state that we have found a software design mistake in the case-study SRP. The mistake have been detected by comparing analysis results with real measurements. An assumption on the design mistake has been verified with a modified model. With the modified model, the Cheddar schedulability analysis tool gave consistent results. From the results we see that there is at least one case where a task misses its deadline, a characteristic which has been observed in the execution traces. Finally we also detected that there is at least one case where there is a priority inversion. This phenomenon has also been observed in the execution traces. Such analysis in design phase makes it possible to warn of the importance of priority-inversion prevention mechanisms when choosing a platform for the waveform.

10. CONCLUSION AND FUTURE WORKS

In this paper we have studied the applicability of the real-time scheduling theory on an industrial Software Radio Protocol. We have suggested a solution to apply this theory on a SRP, using a Model-Driven Engineering approach. Our solution uses the UML MARTE profile and the Cheddar schedulability analysis tool. We first analyzed the SRP architecture and its properties for schedulability analysis. Afterwards we defined requirements on the modeling and schedulability analysis. We then modeled the SRP in MARTE and transformed the model to a Cheddar model in order to analyze the system's schedulability. To evaluate the tool's analysis results, we have compared the data with our own measurements on the THALES SRP. We have shown that simulation results from Cheddar are consistent if specific modifications are used to express accurately the system's behavior. The suggested MARTE modeling thus needs to be extended to make it easier for system engineers to express complex thread semantics and interactions.

We plan to work on the following topics in the future. We would like extend the MARTE modeling to include sequence diagrams to describe particular scenarios to analyze. The transformation would then evaluate the diagrams in order to create an accurate Cheddar model.

The tool itself may also need to be extended to express complex semantics and interactions. We have noticed the following lacks in the Cheddar model and the implemented theory: defining an instant for a thread dependency (i.e. the dependent thread does not have to wait for the entire execution of the depended thread), aperiodic burst activation, thread dependency with different periods for dependent and depended, arbitrary time values (e.g. a thread using a shared resource longer than its capacity).

Most SRP developed at THALES are Software Defined Radios (SDR) [21]. These systems are conform to the Software Communications Architecture (SCA) [14]. The SCA adds several mechanisms for interoperability (e.g. middlewares, CORBA [25]) to the platform which make schedulability analysis difficult to do[34]. Implementing the semantics of a software communication bus in Cheddar, and exploring its influence on the analysis results, is part of our future works.

Furthermore the generic task model used in this paper is much too restrictive for analysis of LINK and NET layers. Several works on extending the classical task models have been done [22, 1, 45, 10, 33, 43, 15]. We would like to extend our task model by exploiting the radio channel access protocol's time division nature. By using the time slots allocation, we would like to predict future execution times of tasks instead of using a constant WCET. This is to get less pessimistic analysis results and exploit the prediction for Dynamic Voltage and Frequency Scaling (DVFS) [29] features.

11. ACKNOWLEDGMENTS

This work is performed in the framework of the ARTEMIS funded project PRESTO (http://www.presto-embedded.eu) and FP7 funded project PHARAON. The views expressed in this document do not necessarily represent the views of the complete consortium. The community is not liable for any use that may be made of the information contained herein.

12. REFERENCES

[1] A. Atlas and A. Bestavros. Statistical rate monotonic scheduling. In *Proceedings of the 19th IEEE Real-Time Systems Symposium*, pages 123–132. IEEE Comput. Soc, 1998.

[2] N. Audsley, I. Gray, and S. Indrusiak. Model-based development of embedded systems - the MADES approach. In *Proceedings of the 2nd Workshop on Model Based Engineering for Embedded Systems Design*, 2011.

[3] AUTOSAR. AUTOSAR specification, 2011.

[4] T. S. Chan. Time-Division multiple access. In *Handbook of Computer Networks*, pages 769–778. John Wiley & Sons, Inc., Hoboken, NJ, USA, 2011.

[5] J. A. Davidson. On the architecture of secure software defined radios. In *Proceedings of the 2008 IEEE Military Communications Conference*, pages 1–7. IEEE, 2008.

[6] J. DeAntoni and F. Mallet. TimeSquare: treat your models with logical time. In *Objects, Models, Components, Patterns*, volume 7304, pages 34–41. Springer Berlin Heidelberg, Berlin, Heidelberg, 2012.

[7] V. Debruyne, F. Simonot-Lion, and Y. Trinquet. EAST-ADL - an architecture description language. In *Architecture Description Languages*, volume 176, pages 181–195. Springer-Verlag, New York, 2005.

[8] P. Dissaux and F. Singhoff. Stood and cheddar : AADL as a pivot language for analysing performances of real time architectures. In *Proceedings of the 4th European Congress on Embedded Real Time Software and System*, 2008.

[9] P. H. Feiler, D. P. Gluch, and J. J. Hudak. The architecture analysis & design language (AADL): an introduction. Technical Report ADA455842, Software Engineering Institute, Pittsburgh, 2006.

[10] C. Fotsing, A. Geniet, and G. Vidal-Naquet. A realistic model of Real-Time systems for efficient scheduling. In *Proceedings of the 33rd IEEE Software Engineering Workshop*, pages 3–12. IEEE, 2009.

[11] 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*, pages 125–134. IEEE Comput. Soc, 2001.

[12] R. Henia, A. Hamann, M. Jersak, R. Racu, K. Richter, and R. Ernst. System level performance analysis - the SymTA/S approach. *IEEE Computers and Digital Techniques*, 152(2):148, 2005.

[13] T. S. Inc. Tri-Pacific software inc. : RAPID RMA. http://www.tripac.com/rapid-rma.

[14] JTRS. Software communication architecture specification, 2012.

[15] L. Ju, A. Roychoudhury, and S. Chakraborty. Schedulability analysis of MSC-based system models. In *Proceedings of the IEEE Real-Time and Embedded Technology and Applications Symposium 2008*, pages 215–224. IEEE, 2008.

[16] K. Klobedanz, C. Kuznik, A. Thuy, and W. Mueller. Timing modeling and analysis for AUTOSAR-based software development: a case study. In *Proceedings of the Conference on Design, Automation and Test in Europe 2010*, 2010.

[17] C. L. Liu and J. W. Layland. Scheduling algorithms for multiprogramming in a Hard-Real-Time environment. *Journal of the ACM*, 20(1):46–61, 1973.

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

[19] J. Medina and l. G. Cuesta. From composable design models to schedulability analysis with UML and the UML profile for MARTE. In *Proceedings of the 3rd Workshop on Compositional Theory and Technology for Real-Time Embedded Systems*, 2010.

[20] N. Medvidovic and R. Taylor. A classification and comparison framework for software architecture description languages. *IEEE Transactions on Software Engineering*, 26(1):70–93, 2000.

[21] J. Mitola. The software radio architecture. *IEEE Communications Magazine*, 33(5):26–38, 1995.

[22] A. Mok and D. Chen. A multiframe model for real-time tasks. *IEEE Transactions on Software Engineering*, 23(10):635–645, 1997.

[23] OMG. CORBA component model specification, 2006.

[24] OMG. EXPRESS specification, 2010.

[25] OMG. CORBA specification, 2011.

[26] OMG. MARTE specification, 2011.

[27] J. Palencia and M. G. Harbour. Schedulability analysis for tasks with static and dynamic offsets. In *Proceedings of the 19th IEEE Real-Time Systems Symposium*, pages 26–37. IEEE Comput. Soc, 1998.

[28] Papyrus. Papyrus. http://www.eclipse.org/modeling/mdt/papyrus/.

[29] P. Pillai and K. G. Shin. Real-time dynamic voltage scaling for low-power embedded operating systems. *ACM SIGOPS Operating Systems Review*, 35(5):89, 2001.

[30] A. Plantec and F. Singhoff. Refactoring of an ada 95 library with a meta CASE tool. In *Proceedings of the 2006 annual ACM SIGAda international conference on Ada*. ACM Press, 2006.

[31] PrismTech. Spectra CX the SCA development tool. http://www.prismtech.com/spectra/products/spectra-cx.

[32] I. R. Quadri, A. Sadovykh, and L. S. Indrusiak. MADES: a SysML/MARTE high level methodology for real-time and embedded systems. In *Proceedings of the 2012 Embedded Realtime Software and Systems Conference*, 2012.

[33] X. Renault, F. Kordon, and J. Hugues. Adapting models to model checkers, a case study : Analysing AADL using time or colored petri nets. In *Proceedings of the 20th IEEE International Workshop on Rapid System Prototyping*, pages 26–33, Paris, 2009. IEEE.

[34] D. Schmidt. Guest editor's introduction:

Model-Driven engineering. *Computer*, 39(2):25–31, 2006.

[35] S. Sendall and W. Kozaczynski. Model transformation: the heart and soul of model-driven software development. *IEEE Software*, 20(5):42–45, 2003.

[36] L. Sha, T. Abdelzaher, K.-E. Arzen, 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*, 28(2-3):101–155, 2004.

[37] L. Sha, R. Rajkumar, and J. Lehoczky. Priority inheritance protocols: an approach to real-time synchronization. *IEEE Transactions on Computers*, 39(9):1175–1185, 1990.

[38] F. Singhoff, J. Legrand, L. Nana, and L. Marcé. Cheddar: a flexible real time scheduling framework. In *Proceedings of the 2004 Annual ACM SIGAda International Conference on Ada*, pages 1–8. ACM Press, 2004.

[39] F. Singhoff, A. Plantec, P. Dissaux, and J. Legrand. Investigating the usability of real-time scheduling theory with the cheddar project. *Real-Time Systems*, 43(3):259–295, 2009. 10.1007/s11241-009-9072-y.

[40] Smartesting. Smartesting. http://www.smartesting.com/index.php/cms/en/home.

[41] Softeam. Modelio. http://www.modeliosoft.com/.

[42] I. Software. IBM software - rational software architect family. http://www-01.ibm.com/software/awdtools/swarchitect/.

[43] M. Stigge. Schedulability analysis with variable computation time of tasks. Technical report, UPPSALA, 2007.

[44] Symtavision. Symtavision - SymTA/S. http://www.symtavision.com/symtas.html.

[45] N. Tchidjo Moyo, E. Nicollet, F. Lafaye, and C. Moy. On schedulability analysis of non-cyclic generalized multiframe tasks. In *Proceedings of the 2010 22nd Euromicro Conference on Real-Time Systems*, pages 271–278. IEEE, 2010.

[46] THALES. MyCCM. http://sourceforge.net/apps/trac/myccm-hi/wiki.

[47] K. Tindell and J. Clark. Holistic schedulability analysis for distributed hard real-time systems. *Microprocessing and Microprogramming*, 40(2-3):117–134, 1994.

[48] VERDE Consortium. ITEA VERDE. http://www.itea-verde.org/.

[49] VideoLAN. VideoLAN. http://www.videolan.org/vlc/index.html.

[50] H. Zimmermann. OSI reference Model–The ISO model of architecture for open systems interconnection. *IEEE Transactions on Communications*, 28(4):425–432, 1980.

APPENDIX

A. UML DIAGRAMS

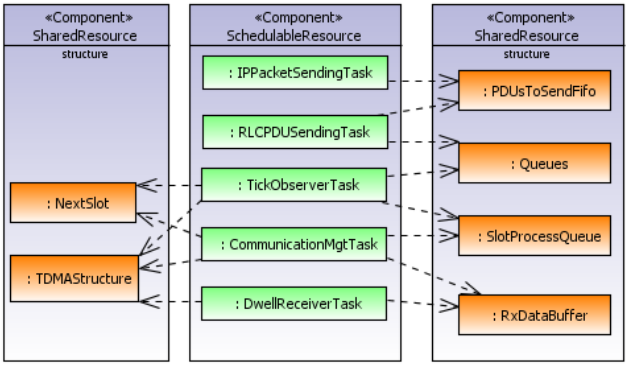

Figure 12: Critical Resource Usage View

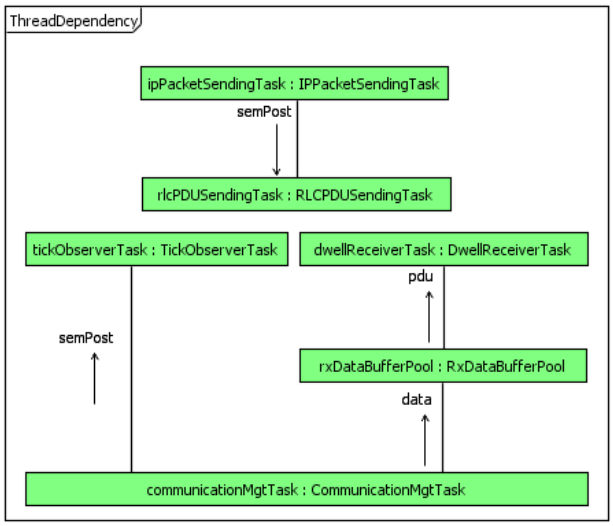

Figure 13: Dependency View

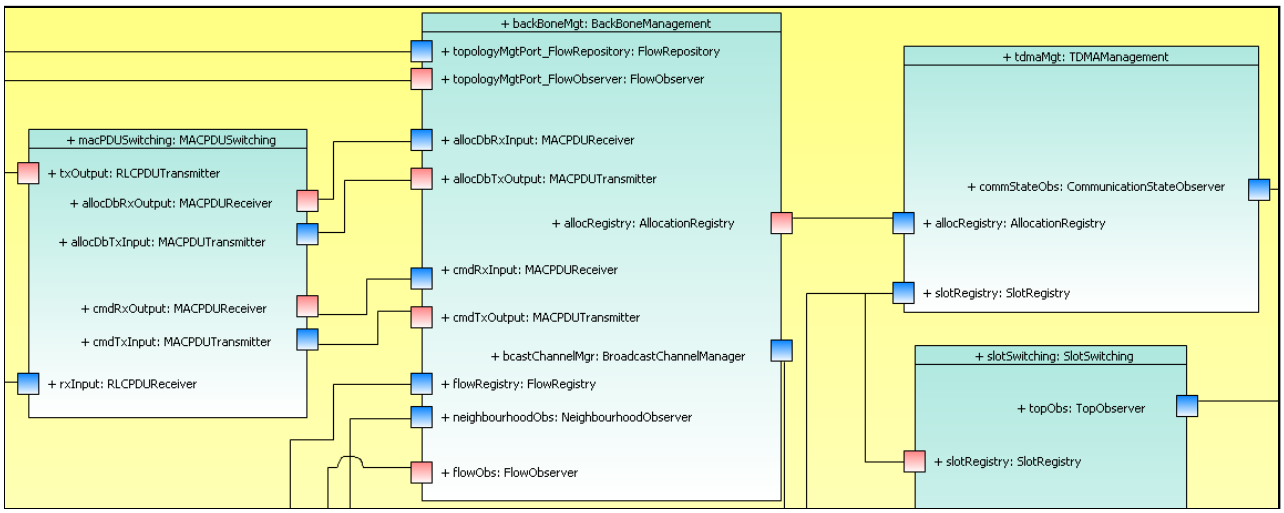

Figure 14: MAC Sub-layer Model

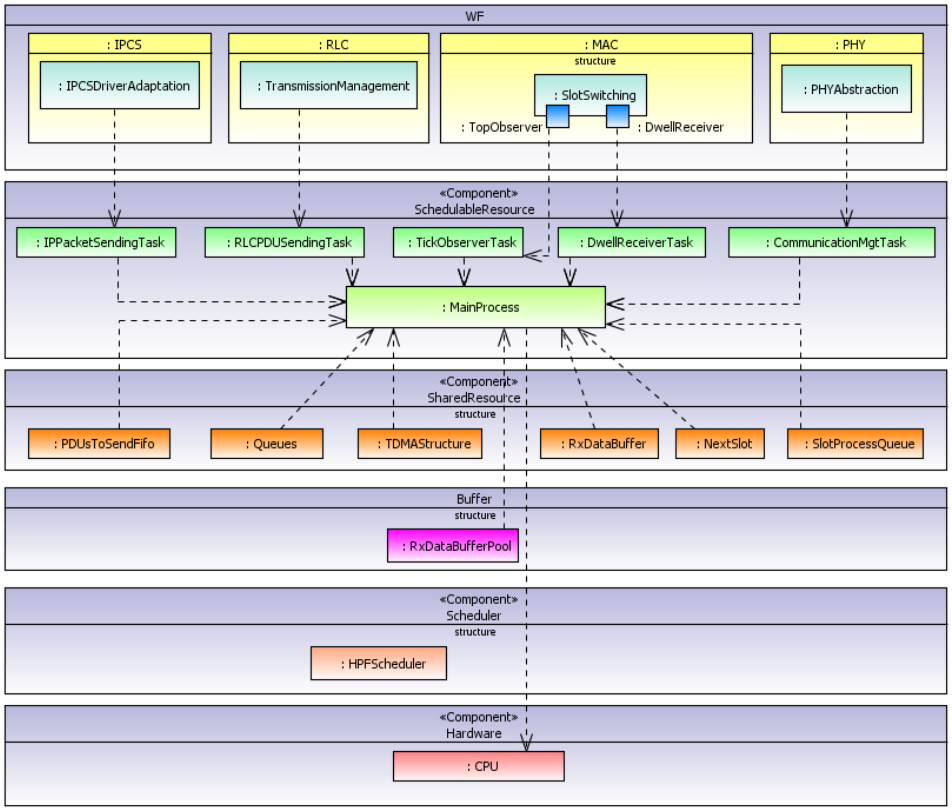

Figure 15: Spatial Distribution View