Applying Holistic Schedulability Tests to Industrial Systems: Experience and Lessons Learned

Shuai Li*†, Stéphane Rubini†, Frank Singhoff†, Michel Bourdellès*
*Thales Communications & Security, 4 av. des Louvresses, 92622 Gennevilliers, France
*Email: {first-name}.{last-name}@fr.thalesgroup.com
†Lab-STICC/UMR 6285, UBO, UEB, 20 av. Le Gorgeu, 29200 Brest, France
†Email: {last-name}@univ-brest.fr

***Abstract*—Several holistic schedulability tests exist in the literature, but they are not always used in the industry. One possibility to increase the usability of such tests, by system designers, is to implement them in scheduling analysis tools. This paper shows an experience on applying holistic schedulability tests to an industrial TDMA software radio protocol from Thales, by implementing the tests in Cheddar, a scheduling analysis tool. Our experience learned through this experience shows advantages and issues of applying such tests in the industry.**

I. Introduction

A Real Time Embedded System (RTES) has limited resources (e.g. processors) and processing depends on time. Most RTES have several concurrent tasks, with deadlines, scheduled on processors. One aspect of designing a RTES is to verify that task deadlines will be met, under a specific scheduler; otherwise said that the system is schedulable. This can be done through schedulability tests [20].

To apply a particular schedulability test, a system is abstracted with a specific *task model*. Two of the first proposed schedulability tests are the Liu and Layland test [11], based on processor utilization, and the Joseph and Pandya test [6], based on response times of tasks (i.e. time from release to completion). Both of these tests are applied to the periodic task model with constrained deadlines (less than period). Since then, several schedulability tests have been proposed for different task models, including models to describe multiprocessor partitioned systems (communicating tasks allocated on processors in a network) with shared resources. The transaction model [22], [17] is one such task model and in this paper we focus on its holistic schedulability tests. Holistic schedulability tests [17] compute upper-bounds of response times of precedence related tasks, by using the response time of the predecessor(s) to compute the release events of the successor(s). The response time computation is iterative. The system starts in an initial state and task response times are updated at each iteration of the test, until a convergence is reached.

To apply schedulability tests in an industrial context, they must be implemented in tools. The tools generally let the user create a model of their system according to an Architecture Description Language (ADL) [13]. Unfortunately, tools that implement holistic schedulability tests are not common [14]. This non-availability is thus one factor that explains why holistic analysis isn't widely used in the industry, the other being the pessimism [17] of response time upper-bounds if not adapted to a specific system.

In this paper, we investigate the applicability of holistic schedulability tests to industrial TDMA Software Radio Protocols (SRP) [9] developed by Thales, through the implementation of such tests in Cheddar, a real-time scheduling analysis tool [3]. Besides running on a multiprocessor partitioned system, a TDMA SRP is both a time-triggered [8] (TDMA) and event-triggered [8] (tasks handling data/control flows in the radio protocol) system. Numerous works [12] have been done previously to analyze schedulability of TDMA systems, but they only handle the time-triggered aspect of such systems, and they do not consider shared resources. For these reasons, our approach consists to model a TDMA SRP with the transaction model and assess schedulability with holistic tests. Indeed, with the transaction model, both tasks released by other tasks (event-triggered), and tasks released in time (time-triggered), can be modeled [16].

The rest of the paper is structured as follows: Section II compares some schedulability analysis tools. Section III presents Cheddar. Section IV defines the transaction model. In Section V, we expose and discuss our solution to implement holistic schedulability tests in Cheddar. In Section VI, a holistic schedulability test is applied on a TDMA SRP. Finally we conclude with future works.

II. Scheduling Analysis Tools

In this paper we focus on Cheddar, a real-time scheduling analysis tool. There exists of course several other state-of-the-art tools that perform scheduling analysis. These tools propose different ADLs and scheduling analysis methods.

Some scheduling analysis tools are based on equations to assess schedulability of a system. MAST [4] is a modeling and analysis suite for real-time applications. In the MAST toolset, an architecture is modeled with an ADL based on events. Events are sent between tasks that have precedence dependency. Tasks are allocated on processors and they may use shared resources. MAST then transforms the event-based architecture model to transactions for scheduling analysis. Holistic schedulability tests for transactions can then be applied.

SymTA/S [5] is a scheduling analysis tool originally dedicated to the automotive industry. As such, SymTA/S handles

an ADL based on entities found in automotive systems, e.g. OSEK, ECU. The architecture is modeled as components allocated on bus and processors. Components have ports through which they receive and send event streams. SymTA/S thus uses an event stream propagation model for scheduling analysis. and the tool applies a composition approach to assess schedulability. In the compositional approach, local scheduling analysis is first performed on a component and then propagated through the system (using event streams) to reach a global analysis result.

Rubus-ICE [14] is a tool suite for model-driven development of real-time systems, with modelers, code generators and analysis methods. The architecture is modeled in the Rubus Component Model (RCM) language. In RCM, software functions are modeled as components that communicate through a producer-consumer scheme. Time parameters are extracted from the component-based model and scheduling analysis methods can then be applied. Holistic schedulability tests have been implemented as plug-ins in Rubus-ICE.

Other scheduling analysis tools are based on simulation to verify (non-)schedulability. STORM [24] is a simulator for multiprocessor architectures. The architecture is described as software and hardware components. After scheduling simulation is conducted, analysis results can be shown as textual reports or graphical diagrams.

Finally some scheduling analysis tools provide several methods, including equation-based methods, formal methods, and/or simulation. Rapid RMA [23] is a set of modeling and scheduling analysis tools. The architecture is modeled with components, with the support for CORBA (an architecture standard focused on interoperability) compliant architectures. Design scenarios are then modeled for scheduling analysis. Rapid RMA uses the rate-monotonic analysis [11] to determine schedulability but it also provides a simulator.

TimeSquare [2] is a model development kit provided as a set of Eclipse plug-ins. The architecture is modeled with an UML MARTE (a UML profile for RTES) component-based model. The UML MARTE model is then transformed to a logical time model called CCSL. Model simulation can then be performed, as well as formal verification of time constraints, to assess schedulability.

Real-Time at Works (RTaW) [15] is a set of tools for timing analysis of real-time systems. Systems are first modeled in SysML (a modeling language for system engineering). RTaW is composed of several tools, including a simulator and formal methods to compute response times, for different architectures respecting industrial standards. For example, RTaW supports a number of communication buses in real-time systems, e.g. CAN, ARINC, Ethernet.

In conclusion we see that most of the existing scheduling analysis tool have an ADL based on components. We also see that holistic schedulability tests are not wide-spread among tools, which limits their usability by system designers that wish for a ”push-button easy” tool.

III. Cheddar

Cheddar is a GPL-licensed open-source real-time scheduling analysis tool written in Ada. The project was started in 2001 and since 2008 the tool is distributed as a module in AADL Inspector [3]. Fig. 1 illustrates the Cheddar tool.

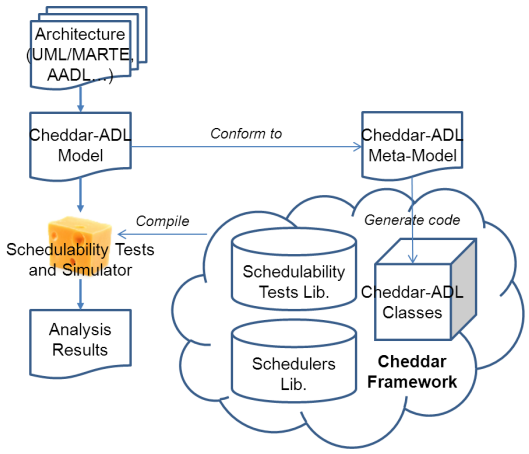

Fig. 1. Cheddar, Real-Time Scheduling Analysis Tool

Users first specify their **architecture** in Cheddar's Architecture Description Language (*Cheddar-ADL*). The GUI provided by Cheddar can be used to generate the **Cheddar-ADL model**, or model transformation [19] can be used to build an architecture in Cheddar-ADL from a standard ADL (e.g. MARTE to Cheddar [9], AADL to Cheddar [3]). A scheduling analysis method provided by the tool (**schedulability test** or **simulation**) is then used to get **analysis results** (i.e. schedulability or simulation trace).

A Cheddar-ADL model is conform to its **Cheddar-ADL meta-model**, which is specified in EXPRESS, a data modeling language. Cheddar-ADL is a language that is close to seminal scheduling analysis methods [11], [6], [21]. For example, entities of tasks, processors and shared resources are defined.

Through a model-driven process,the Cheddar-ADL meta-model is used to generate code of **Cheddar-ADL classes**, a part of the **Cheddar framework**. This ensures that the code of the Cheddar-ADL classes is always conform to the Cheddar-ADL meta-model. The Cheddar framework is composed of generated code from Cheddar-ADL and of manually written code in the **schedulability tests library** and the **schedulers library**.

To extend Cheddar, the general approach is to extend the **schedulability tests** or **schedulers libraries** of the framework. If necessary, the **Cheddar-ADL meta-model** is modified, and code of **Cheddar-ADL classes** is generated. This is the approach we have used to implement holistic schedulability tests in the tool.

IV. Transaction Model

The transaction model, proposed by [22], does not currently exist in Cheddar. This model must be implemented to specify systems on which holistic schedulability tests are applied. Let

us remind the definition of the transaction model, according to [16].

A transaction Γ_i is a group of tasks. A transaction is released by a periodic event. A particular instance of a transaction is called a job. A job of a task in a transaction is released after the event that releases the job of the transaction. Assuming Γ_i is released at t_0, each task $\tau_{ij} \in \Gamma_i$ is defined by the following parameters:

- WCET (C_{ij}) and BCET (C^b_{ij}): A task has a Worst Case Execution Time (WCET) and a Best Case Execution Time (BCET).
- Offset (O_{ij}): The offset of a task is its earliest release time after the time the transaction is released, i.e. a job of τ_{ij} is released at earliest at $t_0 + O_{ij}$.
- Jitter (J_{ij}): A task release is delayed by an arbitrary amount of time between 0 and the maximum jitter, i.e. a job of τ_{ij} is released in $[t_0 + O_{ij}; t_0 + O_{ij} + Jij]$.
- Deadline (D_{ij}): The global deadline [16] of a task is relative to the transaction release time, i.e. a job of τ_{ij} must complete execution before $t_0 + D_{ij}$.
- Blocking time (B_{ij}): Tasks may use shared resources in critical sections [21]. Shared resources access is mutually exclusive so tasks may be blocked. Shared resources are assumed to be protected by a protocol [21] that makes it possible to bound the maximum blocking time of each task, denoted B_{ij}.
- Priority ($prio(\tau_{ij})$): In case of fixed-priority scheduling, a task has a fixed priority. When two tasks want to access the processor, the higher priority task is given access in preference to lower priority task.

Fig. 2 illustrates a transaction Γ_i with tasks τ_{ij} and τ_{ik}.

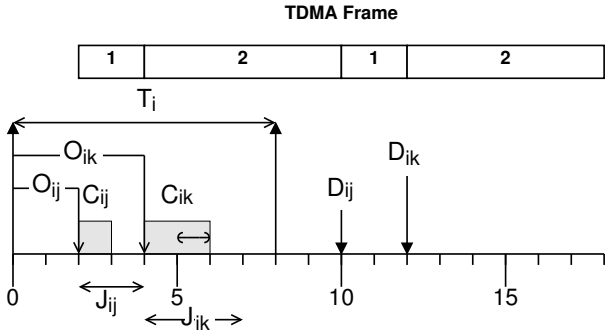

Fig. 2. Transaction Example: Upward arrows are transaction job releases; Double curved arrows are critical sections

Tasks in a transaction are related by precedence dependency. A precedence dependency between a predecessor task and a successor task is a constraint that means that a job of the predecessor task must complete its execution before a job of the successor task can be released [16].

Holistic schedulability tests compute Worst Case Response Times (WCRT) of tasks: the maximum time between a task's earliest release time and its latest completion time.

V. Implementing Transactions and Holistic Schedulability Tests in Cheddar

We now show how Cheddar was extended for transactions and holistic schedulability tests. First our modifications to Cheddar-ADL is shown. Then we discuss how the tests were implemented, focusing on implementation issues and choices we faced.

A. Extending the Cheddar-ADL Meta-Model

To see if Cheddar-ADL is sufficient to implement the transaction model, we focus on a partial meta-model of Cheddar-ADL with task entities, in Fig. 3. In the following sections, reference of entities are those in Fig. 3.

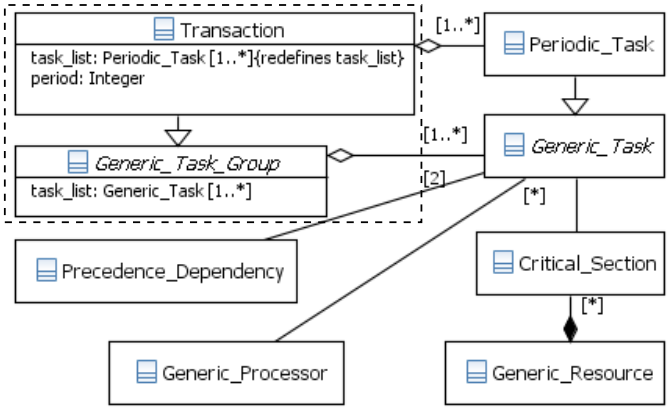

Fig. 3. Cheddar-ADL Partial Meta-Model: Dashed box highlight extensions

1) Re-use of Existing Entities: A number of entities that already exist in Cheddar-ADL can be re-used to implement the transaction model. The `Periodic_Task` entity has attributes of a task in the transaction model. In Cheddar-ADL any task entity can have a `Critical_Section` where it uses a `Generic_Resource` entity, representing a shared resource. Furthermore a task is allocated on a `Generic_Processor` entity. Through the `Precedence_Dependency` entity, a precedence dependency can be specified between two tasks.

The main entity in the transaction model, that cannot be modeled in the current Cheddar-ADL, is the transaction entity itself.

2) New Task Group Entities: To model a transaction entity, we introduce a *task group* concept in the Cheddar-ADL meta-model. A *task group* is modeled by the new entity `Generic_Task_Group` in Fig. 3.

A `Generic_Task_Group` is a set of tasks. Like task entities, any *task group* entity inheriting from `Generic_Task_Group` may have attributes. *Task group* attributes constrain attributes of tasks in the *task group*. The type of tasks that can be in a *task group* are also constrained. For example the `Transaction` entity in Fig. 3 is used to model a transaction with period T_i represented by its attribute `period`. A `Transaction` can only contain `Periodic_Task` entities.

Fig. 4 illustrates the transaction in Fig. 2 modeled in Cheddar-ADL (XML) with the new *task group* entities.

```
<periodic_task id="tau_ij">
 <name>tau_ij</name>
 <capacity>1</capacity>
 <offsets>
  <offset_type>
   <offset_value>2</offset_value>
   <activation>0</activation>
  </offset_type>
 </offsets>
 <jitter>2</jitter>
 <deadline>10</deadline>
 <blocking_time>0</blocking_time>
 <priority>1</priority>
 <period>8</period>
</periodic_task>

<transaction id="tdma_tasks">
 <name>TDMA_Tasks</name>
 <task_list>
  <periodic_task ref="tau_ij"/>
  <periodic_task ref="tau_ik"/>
 </task_list>
 <period>8</period>
</transaction>

<periodic_task id="tau_ik">
 <name>tau_ik</name>
 <capacity>2</capacity>
 <offsets>
  <offset_type>
   <offset_value>4</offset_value>
   <activation>0</activation>
  </offset_type>
 </offsets>
 <jitter>3</jitter>
 <deadline>10</deadline>
 <blocking_time>0</blocking_time>
 <priority>1</priority>
 <period>8</period>
</periodic_task>
```

Fig. 4. Task Group Example

B. Implementation of Holistic Schedulability Tests

After extending the Cheddar-ADL meta-model with transactions, code for the Cheddar-ADL classes was generated. In total 738 lines of code were generated for the new *task group* entities. No extra entities or structures were added to the framework. The holistic tests in [1], [22], [16], [18], [10] were implemented, using the generated code. The main differences between these tests is that they reduce pessimism of response time upper-bounds when considering a specific release pattern in a transaction (e.g. a task can release several tasks immediately [18] and non-immediately [10]).

We now discuss some implementation choices we made and expose issues we faced.

1) Advantages of Implementation Solution: The solution we proposed to model transactions, introduces the *task group* entities but re-uses most of the Cheddar-ADL mechanism for tasks. This has an advantage in terms of meta-model and code maintenance.

The `Transaction` entity we introduced, is generic enough to model any kind of transaction. Indeed, the main difference between different kinds of transactions is their release pattern, i.e. tasks can have more or less successors and predecessors [16], [18], [7]. Since the `Precedence_Dependency` entity in Cheddar-ADL is used to determine precedence between tasks, the order in which tasks are grouped in a `Transaction` does not determine their precedence dependencies. Any task precedence dependency can be represented with the `Precedence_Dependency` entity.

2) Drawbacks of Implementation Solution: Our implementations of holistic schedulability tests use most of the entities that are already present in Cheddar. The main issue from this implementation choice is the time performance of the schedulability tests. Indeed, like stated previously, the `Transaction` entity and `Precedence_Dependency` entity are enough to represent any kind of transaction. On the other hand, these entities may not be the best structure to represent some kinds of transaction.

For example, let us consider the operation to get the successors/predecessors of a task, a common operation among those necessary for holistic schedulability tests. A linear transaction [16], where tasks have at most one successor/predecessor, is best represented with a table. A table reduces considerably the complexity of the operation to get the successor/predecessor of a task in a linear transaction. Indeed, with a table, the operation is $O(1)$ while in our implementation, we have to loop through all entries in the set of `Precedence_Dependency`, so the operation to get a successor/predecessor is $O(n)$.

The general solution to the complexity problem is to implement each kind of transaction with the best adapted data structure. However, later in this paper, we will see that experimental results show that the current implementation stays scalable to a real TDMA software radio protocol.

VI. Experiment

To evaluate our implementation in an industrial context, we apply the test in [10] (called WCDOPS+_NIM and based on [18]) to a real TDMA SRP developed by Thales. In the following sections the TDMA SRP system is first presented before we show how the test is applied.

A. TDMA Software Radio Protocol

A TDMA SRP is a communication protocol embedded in a radio station in a mobile ad-hoc wireless network.

1) System View: From a system point of view, a SRP is divided into several layers according to the OSI model for communication systems. Fig. 5 shows an example of such layers.

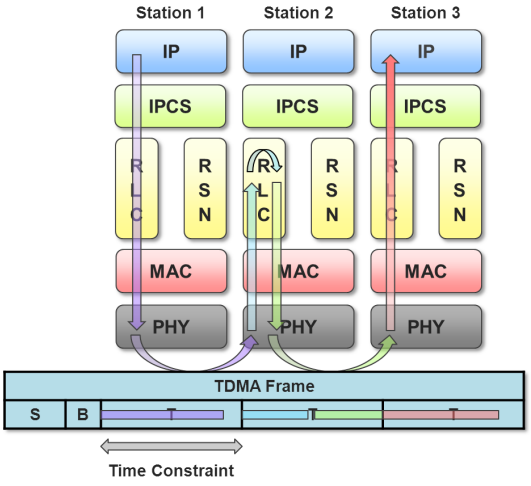

Fig. 5. TDMA SRP System View

In Fig. 5, the **IPCS** layer interfaces with the IP stack of the user system above. The **RLC** layer handles translation between IP packets and radio protocol packets. It also re-routes incoming packets if necessary (e.g. a received packet's destination is a neighbor). The **RSN** layer handles network topology and address updates (e.g. address of neighbor stations in the network appearing/disappearing). When a SRP uses TDMA, the **MAC** layer handles the TDMA protocol by preparing/receiving protocol packets for/from the **PHY** layer that sends/receives them over the air.

In Fig. 5, control and data flows pass through the different layers. The flows are constrained by the TDMA frame. A

TDMA frame is divided into several time slots of different types, durations, and modes. For example in Fig. 5, the TDMA frame has three kinds of slot: **Service** (**S**) for synchronization between stations; **Broadcast** (**B**) for observation/signaling on the network; **Traffic** (**T**) for effective data transmission/reception. Slots of different types do not have the same duration (e.g. a **B** slot is shorter than a **S** and **T** slot). Slots can either be in **Tx** (transmission), **Rx** (reception), or **Idle** mode. A TDMA configuration defines the combination of slots (type and mode) in a TDMA frame. A TDMA frame is repeated after it finishes, with possibly a different configuration. We assume that in a TDMA configuration, only the slot modes change from one TDMA frame to the next.

2) Software and Execution Platform: Fig. 6 shows an example of a the software and the execution platform architecture of a SRP.

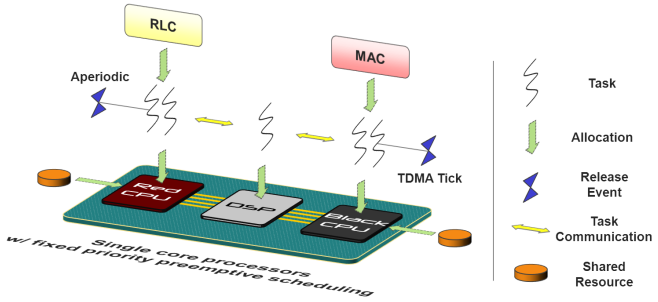

Fig. 6. Software and Execution Platform Architecture

From Fig. 6, we see that the layers are implemented by tasks allocated on processors. In our case-study, tasks that implement layers are POSIX threads so from now on we will call them ”pthread”. Pthreads are scheduled by a fixed priority preemptive policy. Pthreads may communicate and use shared resources. Pthreads handle the flows and they are also constrained by the TDMA frame. For example a pthread may be released by a TDMA tick indicating the start of a slot. Furthermore, each pthread has an execution time that depends on a specific slot, and phreads must finish before some next slot.

Pthreads in the **MAC** layer have hard deadlines to meet, since they handle the TDMA protocol. For this reason the **MAC** layer is the one that interests us for schedulability analysis. The **MAC** layer is implemented on the *Black CPU* and the *DSP* in Fig. 6.

B. MAC Layer Schedulability

In this section we show how to apply the WCDOPS+_NIM test to a **MAC** layer. First the system to analyze is exposed and modeled with a transaction. Then, after applying the test, the schedulability analysis results are discussed. Finally we discuss the modeling of our system with the transactions.

1) System to Analyze: The **MAC** layer has several pthreads constrained by a TDMA frame. The time parameters of pthreads instances are illustrated in Fig. 7. For readability issues, sizes in the figure are not proportional to time values.

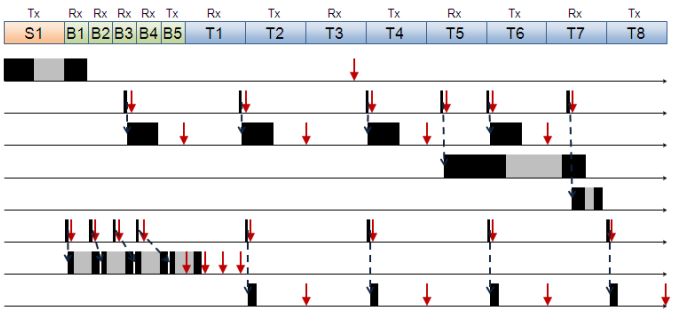

Fig. 7. TDMA Frame and Pthreads: Line = Instances of a pthread; Down arrow = Deadline; Dashed arrow = Precedence; Black = Exec on *Black CPU*; Gray = Exec on *DSP*; 9 pthreads (36 instances) in total

We see that pthreads execute on the *Black CPU* but they may call a function on *DSP* and wait for the answer (i.e. blocking call). We also see that pthreads communicate (i.e. have precedence dependency).

Pthreads are dedicated to either transmission, reception or utility. The basic pthread release pattern is that a reception pthread is released at the start of a **Rx** slot. A transmission pthread is released before a **Tx** slot so data is ready before transmission over the air in the **Tx** slot. There is only one utility pthread that is released at the beginning of the TDMA frame, to prepare the configuration of the next TDMA frame.

Fig. 8 shows the transaction model of pthreads in the **MAC** layer, constrained by the TDMA frame.

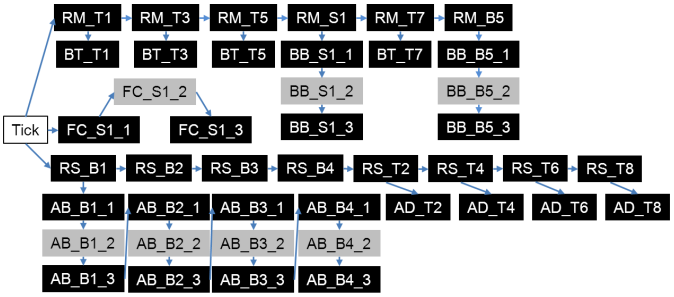

Fig. 8. Tree-Shaped Transaction of **MAC** Layer: Black tasks on *Black CPU*; Gray tasks on *DSP*; Tick task is *ghost root task* [18] on unique processor with 0 WCET

2) Analysis Evaluation: To assess the advantage of applying a holistic schedulability test to our system modeled with a transaction, we compare WCRTs given by WCDOPS+_NIM with those computed by the test in [6]. We choose to compare to the test in [6] because it is a classic one for periodic tasks and because the same approach is applied at Thales.

For each task, let us call the WCRT given by [6] R_{RM}, and $R_{WCDOPS+\_NIM}$ the WCRT given by WCDOPS+_NIM. The ratio $R_{RM}/R_{WCDOPS+\_NIM}$ is computed for each task. In average this ratio is 5.1 so, in average, the test in [6] gives a WCRT more than 5 times higher than WCDOPS+_NIM. Thus to limit pessimism of analysis results, the particular task releases (in time and by other tasks) of our system needs to be considered. It is then beneficial to use transactions to model

a TDMA SRP and apply a holistic schedulability test, since it increases such system's schedulability compared to the test for classic periodic tasks in [6].

3) Modeling Evaluation: Notice that there are 9 pthreads in Fig. 7 but 43 tasks in the transaction in Fig. 8. This is because several instances of a pthread in Fig. 7 are modeled as several tasks in Fig. 8. A pthread on *Black CPU* that makes a blocking call to a function on *DSP*, is also modeled as several tasks in Fig. 8. The transaction used to analyze the system is thus more complex than the original model of the system. In general, the more instances of a pthread there are, the more tasks there are in the transaction. Similarly, in a pthread's execution, the more blocking calls of functions (implemented by other pthreads) there are, the more tasks there are in the transaction.

The difference between model complexities raises several issues. When WCDOPS+_NIM is applied on our model, the time to compute WCRTs takes 7 seconds on an Intel Core i5 @ 2.40GHz. The actual time taken by the analysis is much higher than the time taken to extract information from Fig. 7 and model the system with the transaction in Fig. 8, if done manually. With the XML in Fig. 4, we also see that the modeling process in Cheddar can be tedious. Finally, when the system is modeled manually with transactions, the modeling asks for scheduling analysis theory expertise from designers and the risk of mistakes is not nonexistent. For this reason model transformation tools should be developed.

VII. Conclusion

In this paper we showed how we implemented holistic schedulability tests in the Cheddar real-time scheduling analysis tool, to apply them to an industrial TDMA SRP developed by Thales. We extended the Cheddar-ADL with transactions and generated code of the Cheddar-ADL classes. New tests [1], [22], [16], [18], [10] were then integrated into Cheddar. Experimental results show that a holistic schedulability test, applied to our system, gives WCRT bounds more than 5 times less than classic tests used at Thales. Holistic schedulability tests thus reduce considerably the pessimism of computed WCRTs for our TDMA SRP.

The implementation of the tests in a tool like Cheddar, increases their usability by system designers in the industry. On the other hand, the transaction model of our system is more complex than the original model. Thus model transformations should be developed. In the future we will develop model transformations to integrate the Cheddar tool in a development process at Thales, by transforming system design models to Cheddar-ADL models automatically.

Acknowledgment

The authors would like thank Vuong Nguyen Hong for his contribution to this work.

References

[1] N. Audsley, K. Tindell, and A. Burns. The end of the line for static cyclic scheduling? In *Proceedings: 5th Euromicro Workshop on Real-Time Systems*, 1993.

[2] J. DeAntoni and F. Mallet. TimeSquare: treat your models with logical time. In *Objects, Models, Components, Patterns*, volume 7304 of *Lecture Notes in Computer Science*, pages 34–41. Springer Berlin Heidelberg, Berlin, Germany, 2012.

[3] P. Dissaux, O. Marc, S. Rubini, C. Fotsing, V. Gaudel, F. Singhoff, A. Plantec, V. Nguyen Hong, and H. N. Nam Tran. The SMART project: Multi-agent scheduling simulation of real-time architectures. In *Proc. 7th European Congr. Embedded Real Time Software and Syst.*, Toulouse, France, 2014.

[4] M. G. Harbour, J. G. Garcia, J. Palencia, and J. Drake Moyano. MAST: modeling and analysis suite for real time applications. In *Proc. 13th Euromicro Conf. on Real-Time Syst.*, pages 125–134, Delft, Netherlands, 2001.

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

[6] M. Joseph and P. Pandya. Finding response times in a real-time system. *Comput. J.*, 29(5):390–395, 1986.

[7] J. Kany and S. Madsen. Design optimisation of fault-tolerant event-triggered embedded systems. Master's thesis, Tech. Univ. of Denmark, Lyngby, Denmark, 2007.

[8] H. Kopetz. Event-triggered versus time-triggered real-time systems. In *Operating Systems of the 90s and Beyond*, volume 563 of *Lecture Notes in Computer Science*, pages 86–101. Springer Berlin Heidelberg, 1991.

[9] S. Li, F. Singhoff, S. Rubini, and M. Bourdellès. Applicability of real-time schedulability analysis on a software radio protocol. *ACM SIGAda Ada Lett.*, 32(3):81–94, Dec, 2012.

[10] S. Li, F. Singohff, R. Stéphane, and M. Bourdellès. Extending schedulability tests of tree-shaped transactions for TDMA radio protocols. In *Proc. 19th IEEE Intl. Conf. on Emerging Technology & Factory Automation*, Barcelona, Spain, 2014.

[11] C. Liu and J. Layland. Scheduling algorithms for multiprogramming in a hard-real-time environment. *J. ACM*, 20(1):46–61, Jan, 1973.

[12] N. Malcolm and W. Zhao. The timed-token protocol for real-time communications. *Comput.*, 27(1):35–41, Jan, 1994.

[13] 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.

[14] S. Mubeen, J. Mäki-Turja, and M. Sjödin. Implementation of holistic response-time analysis in rubus-ICE: preliminary findings, issues and experiences. In *Proc. 32nd IEEE Real-Time Syst. Symp., WIP Session*, Vienna, Austria, 2011.

[15] N. Navet, S. Louvart, J. Villanueva, S. Campoy-Martinez, and J. Migge. Timing verification of automotive communication architectures using quantile estimation. In *Proc. 7th European Congr. Embedded Real Time Software and Syst.*, Toulouse, France, 2014.

[16] J. Palencia and M. Harbour. Exploiting precedence relations in the schedulability analysis of distributed real-time systems. In *Proc. 20th IEEE Real-Time Syst. Symp.*, pages 328–339, Phoenix, AZ, 1999.

[17] A. Rahni, E. Grolleau, M. Richard, and P. Richard. Feasibility analysis of real-time transactions. *Real-Time Syst.*, 48(3):320–358, May, 2012.

[18] O. Redell. Analysis of tree-shaped transactions in distributed real time systems. In *Proc. 16th Euromicro Conf. Real-Time Syst.*, pages 239–248, Catania, Italy, 2004.

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

[20] 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 Syst.*, 28(2-3):101–155, Nov-Dec, 2004.

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

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

[23] Tri-Pacific Software Inc. Tri-pacific software inc. : RAPID RMA.

[24] R. Urunuela, A. Déplanche, and Y. Trinquet. STORM a simulation tool for real-time multiprocessor scheduling evaluation. In *Proc. 2010 IEEE Conf. Emerging Technologies and Factory Automation*, pages 1–8, Bilbao, Spain, 2010.