Mapping AADL models to a repository of multiple schedulability analysis techniques

Yassine Ouhammou
LIAS/ISAE-ENSMA
86961 Futuroscope, France
ouhammoy@ensma.fr

Emmanuel Grolleau
LIAS/ISAE-ENSMA
86961 Futuroscope, France
grolleau@ensma.fr

Jérôme Hugues
Université de Toulouse, ISAE
31055 Toulouse, France
jerome.hugues@isae.fr

***Abstract*—To fill the gap between the modeling of real-time systems and the scheduling analysis, we propose a framework that supports seamlessly the two aspects: (1) modeling a system using a methodology, in our case study, the Architecture Analysis and Design Language (AADL), and (2) helping to easily check temporal requirements (schedulability analysis, worst-case response time, sensitivity analysis, etc.). We introduce the usefulness of an intermediate framework called MoSaRT, which supports a rich semantic concerning temporal analysis. We show with a case study how the input model is transformed into a MoSaRT model, and how our framework is able to generate the proper models as inputs to several classic temporal analysis tools.**

I. Introduction

The design phase of real-time embedded systems can span over several years, such as in the avionics domain. Moreover, software development costs are sharply impacted by wrong design choices made in the early stages of development but often detected after the implementation. Timing faults are among vulnerabilities which must be detected at an early stage of the life-cycle, to ensure that a system is feasible. This kind of prediction is based on scheduling analysis. Thus, the design must be as clear as possible, especially concerning the temporal behavior, and it should offer facilities for designers to detect any timing anomaly.

Nowadays, many domain specific languages (like AADL, MARTE, etc.) are used to design real-time systems, by modeling the different system's artifacts and non-functional properties so as to support different kinds of analysis. Nevertheless, the use of these design languages during the real-time design phase provides abstract models, and puts real-time designers in front of efficiency and expressiveness problems. The problem is that the more expressive the model is, the more complex is the feasibility test. However, the choice of the abstraction level and the complexity of the analysis are very substantial points. Consequently, one needs to separate the modeling of a system from its analysis. We propose the use of a pivot language related to the temporal analysis, to help designers to choose the appropriate abstraction level (task models), and to orient designers to the appropriate temporal tests.

To support this vision, we have developed a pivot-language-based framework: the MoSaRT framework (Modeling Oriented Scheduling Analysis of Real-Time system) [15] [17]. This article presents a case study, which stresses the feasibility of this framework starting from an AADL model (Architecture Analysis and Design Language). The choice of AADL is due to its wide utilization in the industry.

The remainder of this article is organized as follows. Section II introduces the motivations of our work. Section III gives a brief description of MoSaRT. Section IV highlights the transition form AADL models to MoSaRT. Then, Section V depicts a conception chain from the model to the schedulability analysis. Finally, Section VI concludes this article.

II. State of Art

Real-time embedded systems have evolved in terms of hardware architectures (uniprocessor, multiprocessor, etc.) and software aspects. This advance leads, on one hand, to an evolution of real-time schedulability analysis tests. On the other hand many standard design languages have been proposed providing models with artifacts for schedulability analysis.

A. Schedulability Analysis : Techniques and Tools

The domain of a schedulability test is defined by a real-time context: tasks characteristics (timing requirements), relationships between tasks and execution platform characteristics. Scheduling theory has been originally studied for the basic Liu and Layland model [1], and extended to cover more precise task models. The basic task model is mathematically simple (periodic independent tasks), but is a distant abstraction regarding most practical applications. Since scheduling analysis performs a worst-case analysis, imprecision in models and methods increases pessimism leading to a severe over-dimensioning of critical systems. Refinements of the task model and analysis techniques have been proposed. They are based on seminal works like the worst-case response time [2], the sensitivity analysis [3], the priority assignment [4], the multi criteria optimization [5], and the approximation of response time [6].

Scheduling analysis techniques have benefited from the model-driven engineering. Several commercial and academic schedulability analysis tools provide some subsets of feasibility tests to help designers in the analysis phase, such as SymTA/S [7], MAST [8], Cheddar [9], etc. Each of these tools uses a different set of concepts to create the input models for simulation and analysis. However, the meta-models of those timing analysis tools differ because they depend on different real-time contexts. Therefore, designers are forced to modify their models to fit the input tool formalism.

B. AADL description and related concerns

AADL (Architecture Analysis and Design Language)[1] is an architecture description language for embedded systems. The description of an architecture in AADL consists in the description of its components and their hierarchic composition. There are three types of components category: software, hardware and system. Interactions between components are expressed through their interface (ports, access to bus, data, etc.).

AADL contains a rich semantics and set of properties to support many families of analysis, including schedulability analysis. This richness makes the use of AADL for temporal analysis expensive in terms of design expertise in addition to the lack of strictness in the definition of its semantics. So, designers have to choose between having models containing an adequate set of elements allowing different verifications techniques, and enriching the scope of the applicable schedulability analysis techniques. Referring to AADL as a standard language dedicated to different kinds of analysis, the latter choice leads to loose this benefit.

Recently, some AADL-related research works aim at helping designers during the modeling phase in order to get a coherent system architecture. Moreover, several analysis tools have been proposed to analyze AADL models. Some works are more generic and they consist in transforming AADL to a formal model in order to treat the model behavior like Fiacre [10], or the work of [11] where authors are interested in the architectural validation to ensure the architecture coherence by using formal methods and constraint satisfaction problems. Peres et al. [12] have also proposed a verification method for real-time system by using model checking. These approaches face technical complexity and combinatorial explosion. Thus, these problems complicate their utilization. Some works are focusing on schedulability analysis. For instance, Cheddar can use design patterns [13] by extracting a set of information from AADL models. This extraction requires a set of ad-hoc information that must be mentioned to support the analysis of AADL models, and which is recognized only by the Cheddar tool. On the other side, the recognition of the design patterns is based only on the architectural model, then it does not consider the behavior of the modeled system.

C. Problem Statement

We note that most activities around AADL analysis assume that the designers know which specific technique they have to apply. Yet, choosing the appropriate schedulability tests is a complex task: one needs to understand the whole real-time context of the system. A real-time context is based on the system architecture but also on the behavior implied by the values of real-time properties (i.e. arbitrary deadlines, concrete tasks, task dependency, etc.).

The difficulty of using the schedulability tests has an influence on the development of systems. Indeed, the steep learning curve behind many of the current model analysis methods has been one of the major impediments to their adoption.

The classical conception from the modeling phase to the temporal analysis phase leads to a potential gap (pessimism and over-dimensioning) due to the estrangement between the abstract model and the practical application [17]. The gap becomes more glaring due to the passage from technical space modeling to the analysis tools.

We summarize the tackled problems as follows: the applicability of real-time scheduling techniques remains a strong issue, as one has to ensure that the model is compliant with implicit design patterns, to select the appropriate analysis tests, to find appropriate tools providing those tests, then to transform the model to other formalisms corresponding to the input formalism of each tool. All these issues are covered by the MoSaRT framework, presented in the next sections.

III. MoSaRT Framework

The MoSaRT framework supports engineers during the design phase to dimension their models, and to analyze their applications with several third-party tools. MoSaRT framework is an intermediate between real-time design languages and temporal or scheduling analysis tools. It is an open framework linking both sides. First, the modeling side which is based on a modeling specific language dedicated for designers. Second, the analysis side which is based on an analysis repository provided by real-time analysts in order to have a chain of transformation and verification starting from design languages down to analysis tools.

A. MoSaRT design language

The MoSaRT language [14] [15] is a domain specific language enriched by a set of non-functional properties related to the timing analysis (see Figure 1). It offers a design based on two layers: the operational layer and the application layer. The operational layer is composed of the hardware model, the software architecture model and the behavioral model of this software architecture. The application layer corresponds to the functional model. In this article, we restrict our focus to the operational layer, that covers the analytical temporal properties. They are conceived as generic concepts in order to support current and future real-time contexts. This allows MoSaRT language to cover several complex temporal models and permits to have an open meta-model which is easily extensible.

MoSaRT language is enriched by several rules to enforce structural correctness, e.g. all required properties are correctly set, a correct topology for connections, etc. Once the model passes the structural verification, then it is "ready" for temporal analysis. The structural rules are organized in three categories. Architectural rules, safety rules and vivacity rules.

The implementation of MoSaRT language is based on Ecore [16] which is a scripting language for meta-models. Moreover, the structural rules are injected in the MoSaRT meta-model as a set of constraints implemented in Object Constraint Language (OCL).

B. MoSaRT Analysis Repository

We propose a flexible way to unify modeling and analysis efforts to support scheduling analysis. The MoSaRT framework proposes an analysis repository, which can be enriched by the analysts. The goal of the analysis repository is to identify

[1] http://www.aadl.info

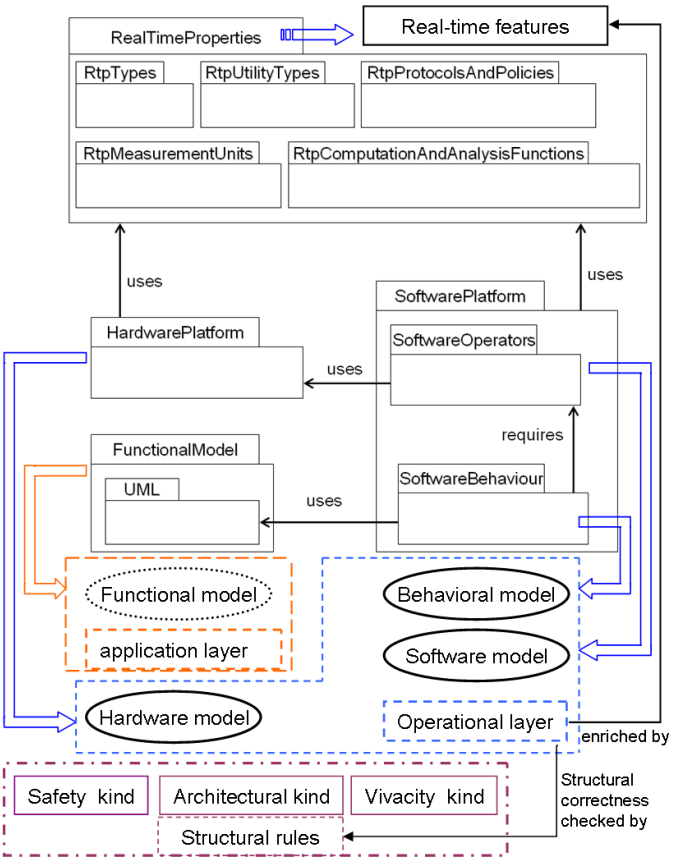

Fig. 1. General structure of MoSaRT language

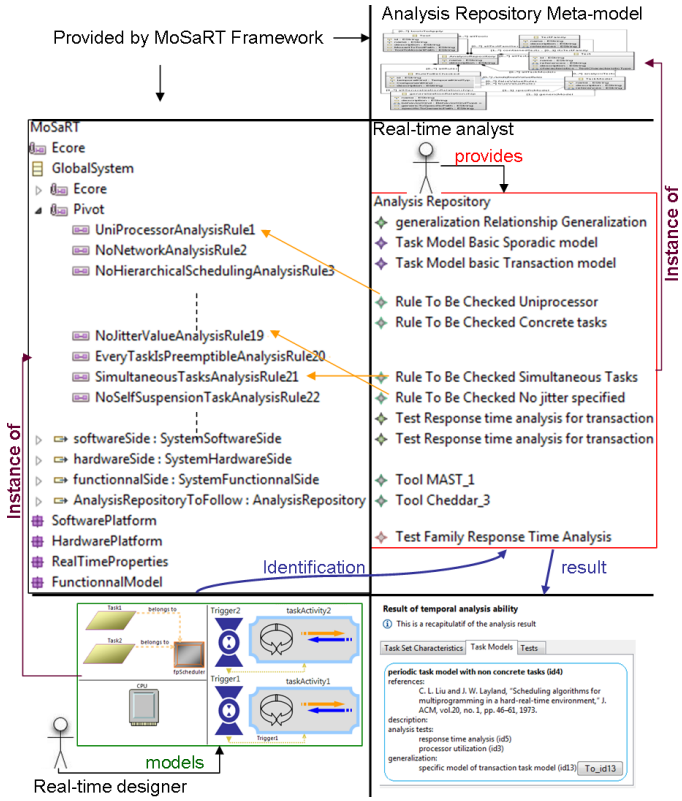

Fig. 2. Different models at different conception levels showing the interaction between analysts and designers via MoSaRT framework

which analysis model is applicable to the design model. Thus, the analysis repository orients the designers to the appropriate analysis tests and also to the prospective analysis tools which provide these tests [17].

Figure 2 shows the interactions between the real-time analyst and the designer via MoSaRT framework.

The analyst provides an analysis repository model (or edits an existing one) by specifying the different components (task models, tools, tests, etc.). The relevant elements of a repository model must be well-defined, each rule being related to a scheduling analysis ability rule. This kind of rules is injected in MoSaRT language as OCL constraints. Thanks to these rules, the repository analysis model provided by an analyst processes the design model, and the identification process can be run for checking the fitting task models, tests and tools which match the design model. If the analysis repository model is rich enough, the designer obtains the checking results showing details concerning the design model and offering also the transformation functions to run the analysis process.

IV. From AADL to MoSaRT

The proximity between the real-time concepts of AADL and MoSaRT reduces the mapping difficulty and eases its utilization as a pivot language. Moreover, the non-functional properties related to the schedulability analysis are highly required by analysis tools. MoSaRT contains most of those properties as generic concepts, and thus enables MoSaRT to be easily connected to several tools.

The temporal analysis concepts contained in MoSaRT language include the ones contained in AADL. Still, the mapping rules are not based just on simple matching operations: it is also based on merging operations, forking operations, etc. Table I represents the mapping of some relevant elements and some schedulability analysis properties. The prefix of the name of each element is related to the package where the element is contained. Then, “Hp” means “HardwarePlatform” package, “So” means “SoftwareOperators” package, “Sb” means “SoftwareBehaviour” package, and “rtp” means that the element represents a temporal property.

A. Transformation implementation

There are many possible formalisms supporting AADL designs. We have opted for AAXL which is the XML formalism of AADL. The AAXL formalism facilitates the interoperability of AADL models.

AADL (version 2) provides two meta-models. A meta-model of the AADL declarative models and a meta-model of AADL model instances. Both kinds of meta-models conform to the Meta Object Facility (MOF) [18]. In other words, if one would like to model a real-time system using AADL, one has firstly to do a model (it conforms to the meta-model of the AADL models). Then, this model has to be instantiated to get the model instance (it conforms to meta-model of the AADL model instances) which represents the practical system.

The implementation of the mapping rules uses ATL (Atlas Transformation Language) [19]. It is a language for transforming models to other models. ATL is a bidirectional trans-

Elements	
AADL	MoSaRT
System	GlobalSystem SystemSoftwareSide GlobalBehaviour SystemFunctionnalSide SystemHardwareSide
Process	SoSpaceProcess
Thread	SoSchedulableTask SbTaskAcitvity SbTimeTrigger
Processor	HpProcessingUnit HpProcessorInterconnector (in multiprocessor case)
Data	SoLocalCommResource or SoRemoteCommResource
Data port	SbCommunicationRelation
Properties	
AADL	MoSaRT
Scheduling_Protocol	rtpSchedulingPolicy
Dispatch_Protocol	rtpPeriodicity
Period	Period or InterArrival
Compute_Execution_Time	rtpExecutionTime

TABLE I. PART OF THE MAPPING BETWEEN AADL AND MOSART

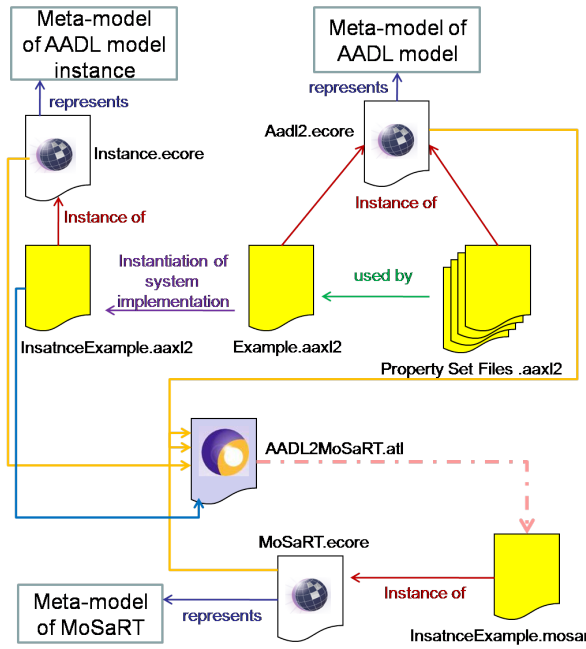

Fig. 3. Architecture of the transformation process between AADL and MoSaRT

formation language, this quality of ATL allows us to keep traceability information. Then, through MoSaRT framework we can ensure an incremental process for editing and analyzing models. Figure 3 shows the input and output files which are required for a successful transformation process. Thus, in order to get an output file model (e.g "InstanceExample.mosart") corresponding to the meta-model of MoSaRT language (represented by "MoSaRT.ecore" file), the transformation process implemented by ATL (e.g. "AADL2MoSaRT.atl" program file) needs four kinds of files. The three meta-models of AADL models (represented by "Aadl2.ecore" file), AADL model instances (represented by "Instance.ecore" file) and MoSaRT language. The fourth type is the model instance file (e.g. "InstanceExample.aaxl2").

Listing 1 represents an example of a forking operation transforming the "Thread" AADL element to "SoSchedulableTask" and "SbTaskActivity" MoSaRT elements.

Listing 1. Example of a transformation rule using ATL

```
lazy rule Thread_To_SoSchedulableTask {
from th:INST!ComponentInstance(th.category=#thread)
using{listOfProperty :
Sequence(AADL2!PropertyAssociation)=
th.ownedPropertyAssociation;}
to st:MoSaRT!"MoSaRT::SoftwarePlatform
::SoftwareOperators::SoSchedulableTask"(
name <- th.name,
representedActivity<-thisModule.Thread_To_SbTaskActivity(th))
do {for (prop in listOfProperty) {
thisModule.PropertiesOfTask(prop,st);}}}
```

B. *Transformation impact on the MoSaRT framework facilities*

Figure 4 shows a global scenario stressing the different facilities of MoSaRT framework. In order to enhance and to ease the utilization of different design language and schedulability analysis techniques, the position of MoSaRT framework between both sides allows a friendly utilization. MoSaRT goal is to take benefit from the modeling spaces (supports of different design languages) and the variety of analysis tools (implementation of different techniques).

For instance, one can import a model instance of a standard meta-models (like AADL, MARTE, East-ADL, etc.), analyze it by different analysis tools and compare the results. The difficulty of this transition is reduced by MoSaRT framework thanks to its capabilities of structural verification, temporal analysis identification and refinement / abstraction. While the importation is using a bidirectional transformation technique like ATL [19], that ensures the reverse transformation and maintains the traceability for the restitution process.

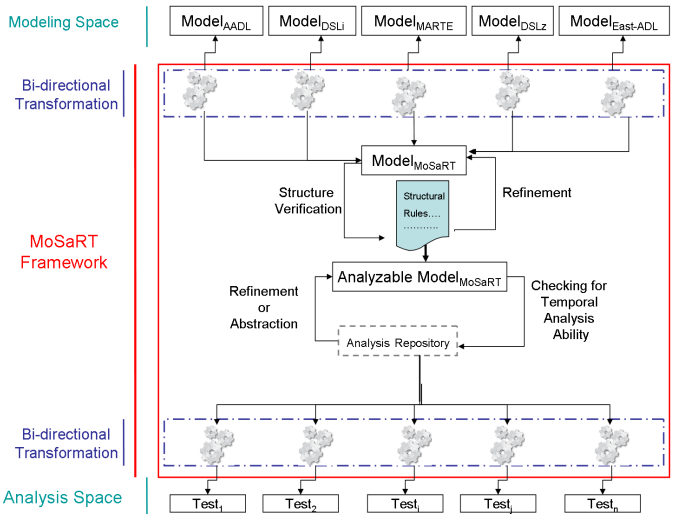

Fig. 4. Global scenario showing various MoSaRT framework facilities

V. CASE STUDY

Through this case study, we illustrate the transformation process from AADL model to MoSaRT model and we highlight the temporal analysis capabilities of MoSaRT framework. As an example, we have chosen to design a simplified mini UAV (Unmanned Aerial Vehicle) [**?**].

A. System description

The UAV is an aircraft able to fly and perform a mission without human presence on board. The practical application of this system is based on a set of interactions and communications between the aircraft and the ground station, where the system application embedded on the aircraft represents a critical point because of its resource limitations and aerodynamic constraints. In this paper we are interested in the assisted mode: in this mode, the operator gives a high level order to the UAV, like an angle of turn, a ground speed, and a vertical speed. The flight control system is embedded and is in charge of the control of the surfaces and engine in order to comply to the operator orders. Then, Figure 5 represents different sensors and actuators related to the flight control and the surrounding functions.

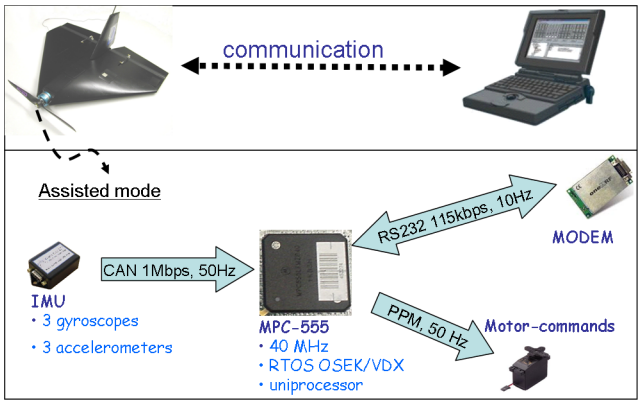

Fig. 5. Different UAV components and their interactions during the assisted mode

The used components of the embedded system are:

- A modem for the transmission and reception of messages.
- An Inertial Measurement Unit, IMU, to acquire the attitude (roll, pitch, yaw) of the UAV.
- The servomotors which are rotary actuators that allow a precise control of the angular position of the ailerons/elevators. The same kind of output is used to control the engine throughput.
- The micro-controller: an uniprocessor using OSEK/VDX as a real-time operator system. It is the main element of this architecture. It reads in parallel the information coming from different sensors, treats them and sends commands to actuators. Equipments connected to the microcontroller operate at different rhythms within specific time constraints. For example, the modem receiver connected to the serial port operates at a frequency of $10Hz$ and sends its data to the microcontroller byte by byte at a rate of 115 kbits per second. The set of bytes transmitted during a period represents the frame. The microcontroller must read each byte before the arrival of the next byte to avoid the risk of losing the entire frame. The IMU has the same type of requirements. Similarly, the system has to refresh the commands sent to the actuators at a period of $20ms$. These commands are sent as PPM signals (Pulse Position Modulation).

The table shown in Figure 6 summarizes the different properties of the task set. T is the period of a task, WCET its worst-case execution time, D its deadline and P its priority. (xN) assigned to "imuAcquisition" and "modemAcquisition" tasks means that the task has to be executed N times during its period to receive a message CAN frame by CAN frame or byte by byte. All the tasks are executed on a processor using a fixed priority scheduling policy. The priority ceiling protocol is used to handle resources.

Task	T	WCET	D	P	Shared data access time		
					PPM	Attitude	Instructions
imuAcquisition (x3)	20000	96	360	3	--	--	--
imuAcquisition_Regulate	20000	16000	18000	4	1500	50	50
modemAcquisition (x10)	100000	12	80	1	--	--	--
modemAcquisitionEnd	100000	240	80000	9	--	--	50
motorCommand	20000	1000	20000	6	500	--	--
transmission_to_Station	100000	3360	50000	7	--	50	--

Fig. 6. The UAV task characteristics (s is the time unit used)

B. System conception using AADL

Figure **??** represents an excerpt of the architecture of the studied system. It shows the different tasks and their communication relationships.

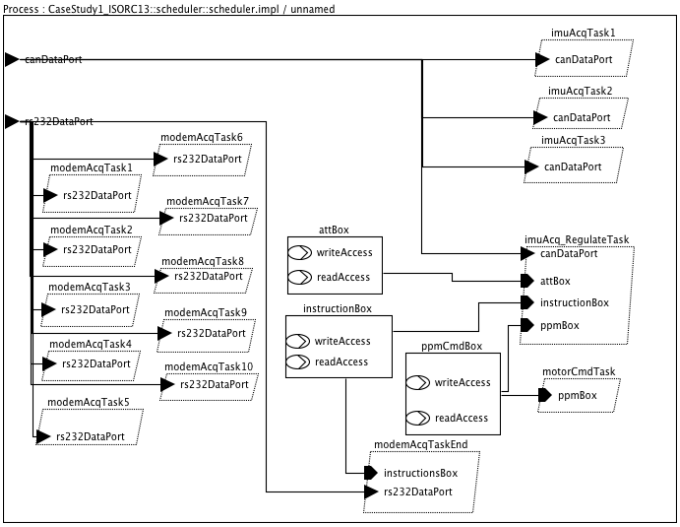

Fig. 7. Part of AADL model of the case study example

We present only a part of the AADL model, expressed in textual language (see Listing 2) to illustrate some relevant properties. We use the property $First\_Dispatch\_Time$ to represent the tasks which have to be executed N times during their periods. In our system, $First\_Dispatch\_Time$ of $\text{Task}_{i+1} \geq Deadline$ of Task_i. For example, $First\_Dispatch\_Time$ of imuAcqTask1 is $0\mu s$, $First\_Dispatch\_Time$ of imuAcqTask2 is $360\mu s$ and $First\_Dispatch\_Time$ of imuAcqTask3 is $720\mu s$.
The system's implementation (shown in Figure **??**) should be instantiated in order to get the instantiation file. Then, we will have all the required elements for the transformation.

Listing 2. AADL model of the case study example

```
.........
thread implementation imuAcquisition_Regulate.impl
 properties
  Priority => 4;
  Dispatch_Protocol => periodic;
  Period => 20000us;
  Compute_Execution_Time => 16000us .. 16000us;
  Deadline => 18000us;
end imuAcquisition_Regulate.impl;
.........
process implementation scheduler.impl
 subcomponents
 imuAcqTask1 : thread imuAcquisition.impl
 {First_Dispatch_Time => 0us;};
 imuAcqTask2 : thread imuAcquisition.impl
 {First_Dispatch_Time => 360us;};
.........
 imuAcq_RegulateTask : thread imuAcquisition_Regulate.impl
 {First_Dispatch_Time => 1080us;};
 modemAcqTask1 : thread modemAcquisition.impl
 {First_Dispatch_Time =>0us;};
.........
 modemAcqTask10 : thread modemAcquisition.impl
 {First_Dispatch_Time =>720us;};
 modemAcqTaskEnd : thread modemAcquisitionEnd.impl
 {First_Dispatch_Time =>800us;};
.........
```

C. Transformation to MoSaRT language

The MoSaRT model obtained after the transformation is shown in figure 7. The part (a) of Figure 7 represents an extract from the software architecture of the system. The part (b) is an extract from the hardware architecture of the system. Then, the part (c) of Figure 7 shows an extract from the software behavior of the system, this model allows to design the temporal characteristics of tasks, their relationships and the manner that each task is triggered. Part (d) will be discussed later (in Section **??**). We admit that the model meets every structural requirement so we can move to the step of the schedulability analysis.

D. Schedulability Analysis Step

This step starts by running the MoSaRT analysis repository. This latter can be provided by real-time expert analyst without requiring a high modeling knowledge. The result found depends on the content of the analysis repository. Figure 8 shows the result provided by our current MoSaRT analysis repository. Part (a) of Figure 8 presents the task set characteristics extracted from the MoSaRT model (Parts (a), (b) and (c) of Figure 7). Part (b) of Figure 8 presents the schedulability tests which correspond to the real-time context of the model and the scientific papers where designers can find more details about the proposed tests. This result also offers to the designers some tools which support the appropriate analysis techniques and transforms the model to different input formalism tools (see Part (c) of Figure 8).

Cheddar and MAST are among proposed analysis tools. After the transformation process from MoSaRT to these schedulability analysis tools, we obtain the results presented in Parts (x) and (y) of Figure 9. That means the model of our case study is not schedulable because the processor utilization is over 100%. However, the result provided by Cheddar (Part (y)) means Cheddar tool is not able to propose a test supporting the temporal characteristics of the model.

The designer can try to change the design characteristics or to modify the hardware architecture by using a multicore

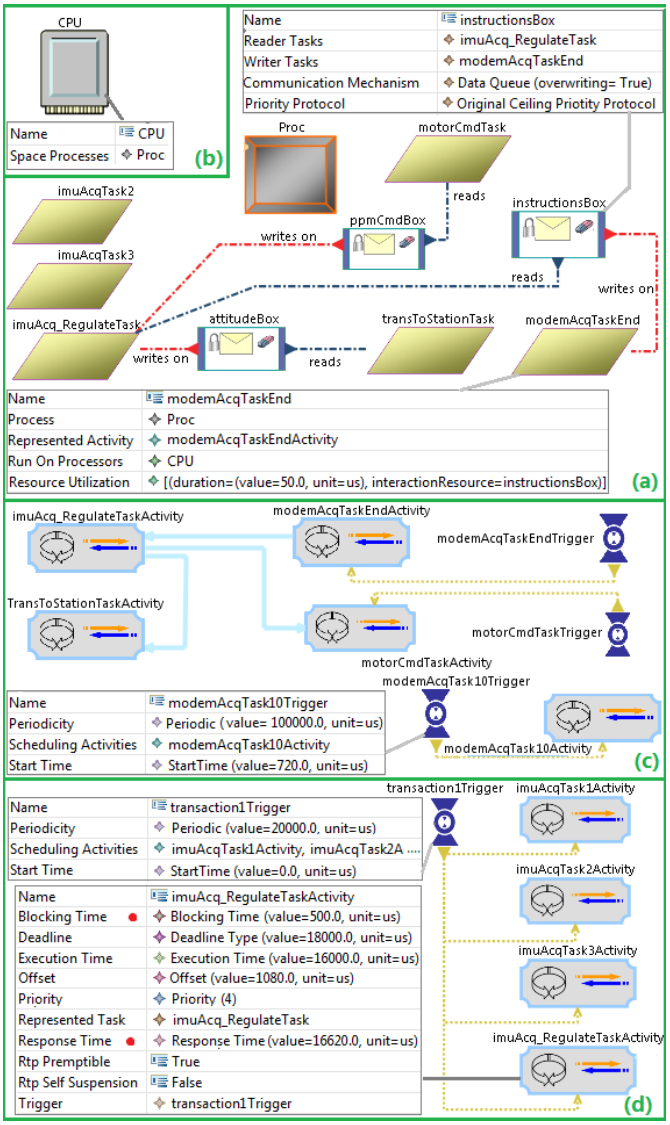

Fig. 8. Parts (a), (b) and (c) represent an extract from the MoSaRT model obtained via the transformation process from AADL model. Part (d) represents an extract from the refined model [The refinement is realized by a transformation from MoSaRT to MoSaRT]

system for example. These choices requires an additional development time, and they may lead to the over-dimensioning. Nonetheless, We note that a generalized task model (i.e. When model 'A' is a generalization of model 'B', then, the behavior of model 'A' includes the worst and the best cases behavior of model 'B') is proposed by the analysis repository (parts (b) of Figure 8).

This means our model can be refined in order to be supported by this generalized model. In this case, the refinement is not based on adding more specifications, but it is related to a semantic specification. This means that the behavior of the model can be viewed differently by maintaining the same architecture. The MoSaRT analysis repository proposes this refinement process which is based on an endogenous transformation (from MoSaRT to MoSaRT).

An excerpt of the refinement result is illustrated on Part

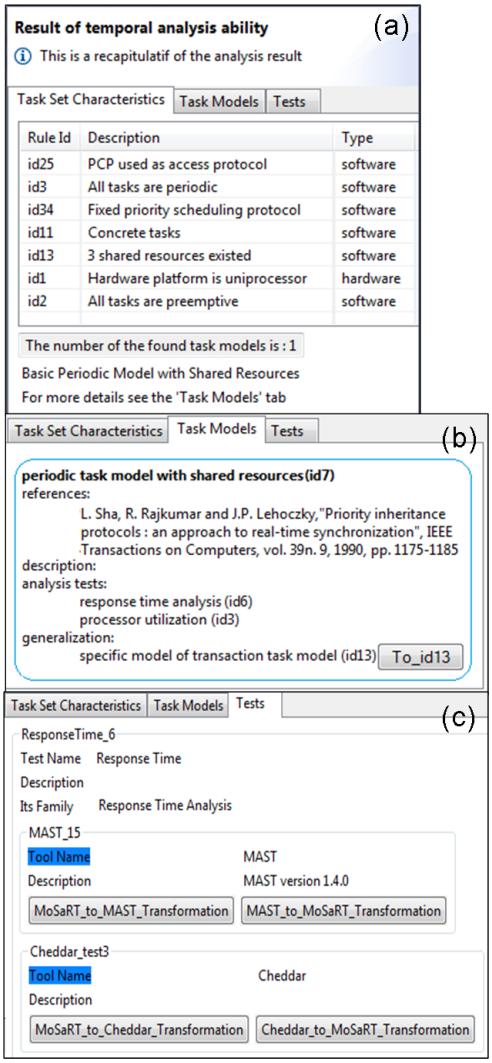

Fig. 9. The result provided by MoSaRT Analysis Repository after applying the model shown in Parts (a), (b) and (c) of Figure 7

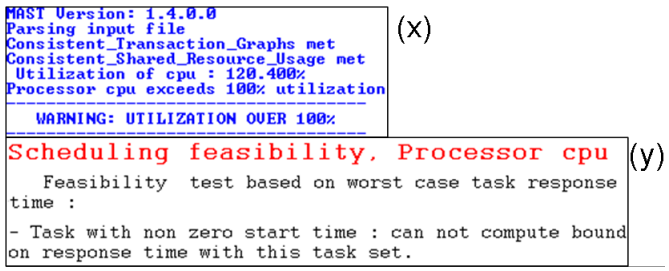

Fig. 10. Parts (x) and (y) are the results provided by MAST and Cheddar tools, these results are related to the model of Figure 7 (Parts a, b and c)

(d) of Figure 7. The new model maintains the same hardware and software architecture. The sole impact is related to the behavioral side. Thus, the new model is composed from Parts (a), (b) and (d) of Figure 7. Due to this refinement, the independent tasks having the same period are regrouped in one transaction. Hence, we obtained an offset-based model instead of having a periodic task model. The new model is composed of four transactions. Moreover, the analysis of the new model gives an interesting result. It is provided by a third-party offset analysis tool based on Tindell work [20]. Figure 10 shows the analysis result of the refined model. C is the worst-case execution time, D represents the deadline, T is the period and B is the blocking time related to the use of shared resources. r corresponds to the response time and R corresponds to the original response time. The comparison between r and R values shows a significant difference related to the pessimism of the periodic model compared to the transaction model.

```
modemAcqTask1,C:12,D:80,r:12,R:12,T:100000,B:0,
modemAcqTask2,C:12,D:80,r:12,R:24,T:100000,B:0,
modemAcqTask3,C:12,D:80,r:12,R:36,T:100000,B:0,
modemAcqTask4,C:12,D:80,r:12,R:48,T:100000,B:0,
modemAcqTask5,C:12,D:80,r:12,R:60,T:100000,B:0,
modemAcqTask6,C:12,D:80,r:12,R:72,T:100000,B:0,
modemAcqTask7,C:12,D:80,r:12,R:84,T:100000,B:0,
modemAcqTask8,C:12,D:80,r:12,R:96,T:100000,B:0,
modemAcqTask9,C:12,D:80,r:12,R:108,T:100000,B:0,
modemAcqTask10,C:12,D:80,r:12,R:120,T:100000,B:0,
imuAcqTask1,C:96,D:360,r:120,R:216,T:20000,B:0,
imuAcqTask2,C:96,D:360,r:120,R:312,T:20000,B:0,
imuAcqTask3,C:96,D:360,r:120,R:408,T:20000,B:0,
imuAcq_RegulateTask,C:16000,D:18000,r:16620,R:16908,T:20000,B:500,
motorCmdTask,C:1000,D:20000,r:17458,R:17458,T:20000,B:50,
transToStationTask,C:3360,D:50000,r:38106,R:38106,T:100000,B:50,
modemAcqTaskEnd,C:240,D:80000,r:38176,R:38296,T:100000,B:0,

                          | transaction2,100000
transaction1,20000        | ,modemAcqTask1
,imuAcqTask1              | ,modemAcqTask2
,imuAcqTask2              | ,modemAcqTask3
,imuAcqTask3              | ,modemAcqTask4
,imuAcq_RegulateTask      | ,modemAcqTask5
                          | ,modemAcqTask6
transaction3,100000       | ,modemAcqTask7
,transToStationTask       | ,modemAcqTask8
                          | ,modemAcqTask9
transaction4,20000        | ,modemAcqTask10
,motorCmdTask             | ,modemAcqTaskEnd
```

Fig. 11. Analysis result provided by offset analysis tool. The result is related to the refined model of Figure 7

E. Restitution of the analysis results

Once the schedulability analysis step is finished, we move to the restitution step. That starts by enriching the MoSaRT model through its output properties like “blocking time” and “response time” (see Part (d) of Figure 7). Then, we run the reverse transformation from MoSaRT to AADL.

Listing 3. AADL model adjusted after the analysis step

```
system uav
.........
end uav;
.........
thread group transaction1
features
canDataPort : in data port;
attBox : requires data access attitudeBox.impl;
ppmBox : requires data access ppmCmdBox.impl;
instructionsBox : requires data access instructionsBox.impl;
end transaction1;
.........
thread group implementation transaction1.impl
subcomponents
 imuAcqTask1 : thread imuAcquisition.impl
 {Dispatch_Offset => 0us;
        Worst_Case_Response_Time => 120us;};
 imuAcqTask2 : thread imuAcquisition.impl
 {Dispatch_Offset => 360us;
        Worst_Case_Response_Time => 120us;};
 imuAcqTask3 : thread imuAcquisition.impl
 {Dispatch_Offset => 720us;
 Worst_Case_Response_Time => 120us;};
 imuAcq_RegulateTask : thread imuAcquisition_Regulate.impl
 {Dispatch_Offset => 1080us;
 Worst_Case_Response_Time => 16620us;
 Blocking_Time=> 500us;};
```

```
 .........
 properties
 Period => 20000us;
 end transaction1.impl;
.........
process implementation scheduler.impl
 subcomponents
 transaction1 : thread group transaction1.impl;
 transaction2 : thread group transaction2.impl;
 transaction3 : thread group transaction3.impl;
 transaction4 : thread group transaction4.impl;
.........
end scheduler.impl;
.........
```

Listing 3 represents a piece of the adjusted model. The transaction is interpreted as a "thread group" element. Each "thread group" contains a set of tasks having the same period and being independent from each other. Thus, we only changed the semantic view of the model without modifying the architecture of the UAV control system. The $First\_Dispatch\_Time$ property has been converted into $Dispatch\_Offset$. $Worst\_Case\_Response\_Time$ and $Blocking\_Time$ have been added in the model to refine scheduling metrics.

VI. Conclusions

This paper presented an approach to assist designers throughout the design process to cope with the difficulty of design and analysis of real-time systems. We presented the utility of the MoSaRT framework to reduce the gap between the design languages and the temporal analysis tools. Moreover, we highlighted the utilization of the MoSaRT language as a pivot temporal analysis design language, and the utilization of the MoSaRT analysis repository as a schedulability analysis advisor. We tested our approach using AADL as a formalism of input model and then we called a set of automatized actions until arriving to the appropriate analysis tests which correspond to the real-time context of the input system model. Our future works will focus on stabilizing this work in order to support complex architectures. We are also working on the transformation process from other design languages to MoSaRT framework. To benefit from temporal analyst skills and designer skills and to unify their efforts we intend to make this work available by sharing a stable version as soon as possible under a LGPL license.

References

[1] C. L. Liu and J. W. Layland, "Scheduling algorithms for multiprogramming in a hard-real-time environment," *J. ACM*, vol. 20, no. 1, pp. 46–61, 1973.

[2] M. Joseph and P. K. Pandya, "Finding response times in a real-time system," *Comput. J.*, vol. 29, no. 5, pp. 390–395, 1986.

[3] E. Bini, M. Di Natale, and G. Buttazzo, "Sensitivity analysis for fixed-priority real-time systems," *Real-Time Syst.*, vol. 39, pp. 5–30, August 2008.

[4] N. Audsley and Y. Dd, "Optimal priority assignment and feasibility of static priority tasks with arbitrary start times," 1991.

[5] R. Mishra, N. Rastogi, D. Zhu, D. Moss, and R. Melhem, "Energy aware scheduling for distributed real-time systems," in *IPDPS 2003*, p. 21.

[6] T. H. C. NGuyen, P. Richard, and E. Bini, "Approximation techniques for response-time analysis of static-priority tasks," *Real-Time Systems*, vol. 43, pp. 147–176, 2009.

[7] R. Henia, A. Hamann, M. Jersak, R. Racu, K. Richter, and R. Ernst, "System level performance analysis - the symta/s approach," in *IEE Proceedings Computers and Digital Techniques*, 2005.

[8] J. L. Medina Pasaje, M. González Harbour, and J. M. Drake, "Mast real-time view: A graphic uml tool for modeling object-oriented real-time systems," in *RTSS 2001*, pp. 245–256.

[9] F. Singhoff, J. Legrand, L. Nana, and L. Marcé, "Cheddar: a flexible real time scheduling framework," in *SIGAda 2004*, pp. 1–8.

[10] B. Berthomieu, J.-P. Bodeveix, C. Chaudet, S. Zilio, M. Filali, and F. Vernadat, "Formal verification of aadl specifications in the topcased environment," in *Proceedings of the 14th Ada-Europe International Conference on Reliable Software Technologies*.

[11] M. de Roquemaurel, T. Polacsek, J.-F. Rolland, J.-P. Bodeveix, and M. Filali, "Assistance la conception de modles l'aide de contraintes," in *AFADL 2010*, pp. 181–196.

[12] F. Peres, P.-E. Hladik, and F. Vernadat, "Specification and verification of real-time systems using pola," *IJCCBS*, pp. 332–351, 2011.

[13] V. Gaudel, F. Singhoff, A. Plantec, S. Rubini, P. Dissaux, and J. Legrand, "An ada design pattern recognition tool for aadl performance analysis," in *SIGAda*, 2011, pp. 61–68.

[14] Y. Ouhammou, E. Grolleau, M. Richard, and P. Richard, "Towards a simple meta-model for complex real-time and embedded systems," in *MEDI 2011*, pp. 226–236.

[15] ——, "Model driven timing analysis for real-time systems," in *ICESS 2012*, pp. 1458–1465.

[16] D. Steinberg, F. Budinsky, M. Paternostro, and E. Merks, *EMF: Eclipse Modeling Framework*. Addison-Wesley Professional, 2008.

[17] Y. Ouhammou, E. Grolleau, M. Richard, and P. Richard, "Reducing the gap between design and scheduling," in *The 20th International Conference on Real-Time and Network Systems (RTNS)*, ACM, Ed., Nancy, France, November 2012.

[18] OMG, *OMG Meta Object Facility (MOF) Core Specification*. Object Management Group, Inc, 2011. [Online]. Available: www.omg.org/spec/MOF/2.4.1/

[19] F. Jouault, F. Allilaire, J. Bézivin, and I. Kurtev, "Atl: A model transformation tool," *Sci. Comput. Program.*, vol. 72, no. 1-2, pp. 31–39, 2008.

[20] K. Tindell, "Adding Time-Offsets to schedulability analysis," Department of Computer Science, University of York, Tech. Rep. YCS-1994-221, 1994.