A Contract-Based approach to support Goal-Driven Analysis

Guillaume Brau*†, Jérôme Hugues†, Nicolas Navet*
* University of Luxembourg, CSC Research Unit/LASSY
6 rue R. Coudenhove-Kalergi, L-1359 Luxembourg, Luxembourg
{guillaume.brau, nicolas.navet}@uni.lu
† Université de Toulouse – Institut Supérieur de l'Aéronautique et de l'Espace, DMIA
10 avenue E. Belin, 31055 Toulouse, France
jerome.hugues@isae.fr

***Abstract*—In the design of real-time systems, models are usual artifacts to capture and represent the various features of the system. They are later analyzed to check for their correctness. A key issue is to handle models and analyses in a systematic, consistent and efficient way. This paper presents an approach for the systematic and correct execution of analyses on real-time system models along with a proof-of-concept.**

The contribution aims at 1) directing the analyses targeting *goals* and 2) using *contracts* to reason about models, analyses and goals. An example of goal is to enrich a model with missing information or to obtain precise data to conclude about the system quality. In our approach, contracts are used to formally depict both the properties required and provided by the analyses ; but also models and goals. Through the concept of contracts, we identify all the feasible paths to execute the analyses in order to reach a goal.

I. Introduction

Distributed Real-time Embedded (DRE) systems are now used in safety-critical contexts such as for transportation, space, defense or telecommunications applications. It is mandatory to develop such systems with the required quality (*e.g.*, regarding timing, safety or security aspects).

Model-Based Engineering (MBE) emphasizes the rational use of *models* to build systems. Analysis techniques – *analyses* for short – can be applied to check conformity of the models with respect to the quality requirements. Usually, numerous analysis techniques are available for each quality concern ; *e.g.*, to check the respect of temporal constraints regarding tasks scheduling, communications delays or worst-case execution times.

An emerging challenge is to deal with the diversity of models and analyses to enable already existing modeling and analysis tools to collaborate. A first issue is to apply the "correct" analysis to the "right" model : each analysis makes assumptions on the system that must be held true in the model. Another issue is to manage the interactions between analyses. In some cases, it can be necessary to combine the execution of analyses in order to obtain targeted information. For instance, when an analysis outcome can be used by another analysis, these analyses can be executed successively.

We note a lack of theoretical and practical support to handle models and analyses at large in a systematic, consistent and efficient way. We observe that most of the present works focus either on the transformation aspects (*i.e.*, providing solutions to switch from model-oriented to analysis-oriented frameworks) such as in [1], or partially consider the management aspects [2], [3]. In the latter cases, both the exploitation of analyses outcomes and the sequencing of analyses remain unaddressed. Finally few works [4] look for a high-level and independent framework dedicated to the management and integration of models and analyses.

In this paper, we present an approach to systematize the application of analyses on real-time system models. We propose : 1) to target *goals* for directing the analyses ; 2) to use *contracts* as a formalism to abstract models, analyses and goals, and to enable reasoning about their interfaces. We build on the concept of contract [5] that we extend to formally describe the properties required and provided by the models, analyses and goals. We provide an implementation to handle contracts so as to : 1) state if there exists *at least* a path for the execution of the analyses in order to attain a goal according to an input model ; 2) find out *all* the feasible paths from an input model towards a goal.

The implementation of our approach relies on Alloy [6] which is a logic-oriented modeling language that we use for contracts description. In addition, the Alloy analyzer is used to solve the constraints related to the dependences between contracts. As a proof-of-concept, we propose to apply our strategy for assessing schedulability of real-time systems modeled with the Architecture Analysis & Design Language (AADL) [7]. The experimental results prove that our approach is capable of organizing analyses for systems models of different complexity in an affordable time.

The remainder of this paper is organized as follows. We first discuss related works (Section II). We then introduce contracts with simple examples (Section III). In Section IV, we detail the use of contracts in our approach and the implementation with Alloy. The approach is experimented in Section V on several case studies. We finally conclude in Section VI.

II. Background and Related works

Besides the numerous contributions made regarding modeling languages and analysis techniques, many studies look now to apply these analyses on design-oriented models – using AADL for instance. This can have different aims : verification of a system design, use of the analyses results in advanced design workflows including virtual integration [8], design space exploration or optimization [9], [10].

As most of the analyses are executed by independent tools relying on their own models, it is necessary to transform design-oriented models towards analysis-specific ones. As examples of external tools concerned by schedulability and timing analysis, applicable on AADL models : Cheddar [11] and MAST [12] rely on real-time scheduling theory, TINA [1] uses Petri Nets for model checking. The transformation approaches take care of the "technical" couplings (*i.e.*, regarding the syntax and eventually the semantics) between design-oriented models and analysis-oriented ones, thus enabling to analyze design-oriented models. Nevertheless, there is no support to decide when and in which manner to apply the analyses ; and limited support to handle the results produced by the analyses.

The two works in [2], [3] implement a similar approach to help choosing the analyses that can be executed on design-oriented models. The idea is to check the prerequisites of a given analysis on the model before its execution. In [2], this is achieved through the formalization and verification of *contexts* defined for each analysis. In [3], the authors propose to check AADL *subsets* before applying an analysis. We note that the solutions are restricted to the MoSaRT language and AADL and only target real-time scheduling analyses. Again, both the analyses outcomes and the sequencing of the analyses remain unaddressed.

Contracts have been investigated and used in very different contexts. In general, contracts are abstract structures consisting in a specification of *assumptions* and *guarantees*. A contract describes the expectations on its environment (the assumptions) and the satisfactions provided (the guarantees) under the environment assumptions. Assume-guarantee reasoning is useful to manage complex state spaces. Contracts can be used in very different settings ; *e.g.*, in the design of Cyber-Physical Systems that has to integrate modeling concerns from many disciplines and stakeholders [5], [13].

Recent works propose to extend contracts to manage the integration between analyses. Ruchkin et al. [4] aim to use contracts in order to prevent the incorrect ordering and execution of a set of analyses. They propose a specific language to specify contracts along with a verification algorithm to find inter-dependencies between contracts. Thereby, they ensure the results produced by the successively executed analysis tools do not enter in conflict. In a second paper [14], they detail the implementation of their approach in the OSATE tool platform [15] to manage analyses to be applied on AADL models. The main objective is to *preserve* the outcomes produced by the execution of a set of analyses : the way to manipulate the analyses to produce expected results is left to the analyst. As a second limitation, the implementation which is detailed in [14] is very close to and highly dependent on the AADL language.

In our works, we investigate a complementary approach : we use contracts to identify the correct execution order of a set of analyses to *produce* a target result (*i.e.*, attain a goal). In a way, we look for a "super-analysis" to get expected outcomes ; the way to integrate the "super-analyses" outcomes in the design space is not currently addressed. In addition, we aim at separating contracts reasoning from lower-level considerations ; *e.g.*, suitable means to syntactically or semantically interpret models can be highly dependent on the modeling languages or domain-specific design patterns. Our approach is applicable with any modeling language as soon as low-level means to map the modeling framework to the contract one exist. That is : 1) contracts are defined and handled in a dedicated environment ; 2) lower-level artifacts should be implemented separately with respect to the underlying models and analyses.

III. Contracts for models, analyses and goals

In this section, we present basics for contracts through examples. First, we provide a definition for models, analyses and goals, in the scope of our work. Then, we have to distinguish "properties" and "meta-properties" that can be manipulated by models, analyses and goals. At last, we introduce contracts and discuss an important feature : precedence between contracts.

A. Models, analyses and goals

A model is a representation, *i.e.*, a sound abstraction, of a system. For our concern, we propose the following definitions.

Definition 1 : Model**.** *A model is a couple* $M = (S, P)$ *:*

- S *is the model domain, i.e., a set of sorts,*

Sorts in a model are closely related to the modeled system and the aspects being considered. Sorts can be mathematics-oriented (*e.g.*, Booleans, Integers) or domain-specific (*e.g.*, real-time scheduling entities such as tasks, processors and scheduling algorithms).

- P *is a set of properties. A property is an association of sorts* $P : S \rightarrow S$.

Properties can be used to describe models invariants such as threads periods, processor scheduling policies ; but also a model status like being schedulable, safe, etc.

Informally, we can define[1] an analysis as a *"a careful study of something to* learn *about its parts, what they do, and how they are related to each other ; an explanation of the nature and meaning of something"*.

Definition 2 : Analysis**.** *An analysis is a function that operates over a model* $A : M \rightarrow M$.

[1] according to http://www.merriam-webster.com/

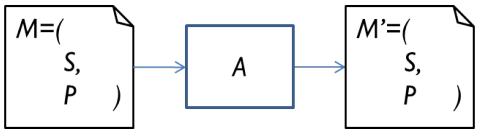

Fig. 1. An analysis A inputs a model M and outputs another model : $M' = A(M)$.

Models and analyses can be combined in order to produce other models. The approach discussed in this paper propose to manipulate models and analyses in order to produce "goal" models – reach *goals*.

Definition 3 : Goal**.** *Let $\mathcal{M}$ be a set of models and $\mathcal{A}$ be a set of analyses. A goal is a particular model required over a set of models and analyses $G : \mathcal{M} \times \mathcal{A} \rightarrow \mathcal{M}$.*

Example: Let us discuss the case of real-time systems modeling and analysis. The tasks model by Liu and Layland [16] enables to model processing aspects of real-time systems. In its simplest version, the execution of the system is carried out with respect to tasks characteristics (periods and execution times) and processing resources properties (scheduling policy) as specified on Table I.

property	*description*	*association*
Per	period	$\mathbb{R} \rightarrow task$
Exec	(worst-case) execution time	$\mathbb{R} \rightarrow task$
Sched	scheduling policy	$\{FP, RM, DM\} \rightarrow scheduler$

TABLE I
M_0 : PROPERTIES INVOLVED IN THE LIU AND LAYLAND'S TASKS MODEL.

Based on the latter tasks model (M_0), several schedulability analyses have been proposed – schedulability analyses aim to conclude a given configuration of a task model is schedulable (*i.e.*, all the tasks deadlines will be met). We can cite *processor utilization bound feasibility tests* [16] or *response time analyses* [17] (respectively A_1 and A_2 in the following).

The two analyses outputs two different models as shown in Table II. First one ($M_1 = A_1(M_0)$) provides a property (**isSched**) which assigns a binary value to each task – $true$ meaning the task is schedulable and $false$ meaning it is not. Second one ($M_2 = A_2(M_0)$) provides the worst-case response time for each task (**respTime** property). It is the time for a task to complete in the worst-case scenario.

model	*property*	*description*	*association*
M_1	**isSched**	tasks schedulability	$\{true, false\} \rightarrow task$
M_2	**respTime**	tasks response times	$\mathbb{R} \rightarrow task$

TABLE II
TWO MODELS PRODUCED BY SCHEDULABILITY ANALYSES. A *processor utilization bound feasibility test* [16] OUTPUTS THE MODEL M_1. A *response time analysis* [17] OUTPUTS THE MODEL M_2.

For an analyst, any of M_1 or M_2 can be a goal. In that case, it is noted G_1 (or G_2).

B. Discussion about properties and "meta-"properties

We explained that an analysis outputs a model (*i.e.*, sets of sorts and properties) according to an input model. We must distinguish *simple*-properties from what we will call in the sequel *meta*-properties. We choose to use the prefix "meta" to point out that properties and meta-properties are related to a different level of abstraction :

- Simple-properties are properties that are intended to be present or integrated in a model,
- Meta-properties – properties about properties – are properties about a model which are not intended to be present or integrated in this model.

Practically speaking, simple-properties can be constituent of a design-oriented model ; *e.g.*, periods of tasks. On the contrary, meta-properties are more related to providing information about a design-oriented model ; *e.g.*, the tasks are schedulable. Meta-properties can be used to conclude about a model while simple one cannot. As a matter of fact, meta-properties mostly involve Booleans. An analysis can provide properties, meta-properties or both.

Example: Considering the example on subsection III-A, the first analysis (*processor utilization bound feasibility tests*) outputs the **isSched** property. We assert this is a meta-property as it can be directly used to conclude about the schedulability of the tasks : if the boolean-valued property is true, the system is schedulable.

The second analysis (*response time analyses*) provides the response time for each task (**respTime** property). Without further interpretation, that property does not allow to conclude about the schedulability of the tasks. Nevertheless, it is possible to conclude about the system schedulability by comparing the tasks response-times against the tasks deadlines : if all the response-times are under the deadlines, the system is schedulable. Such an analysis requires access to an additional **Dline** property that explicitly specifies a deadline for each task. This is depicted on Figure 2.

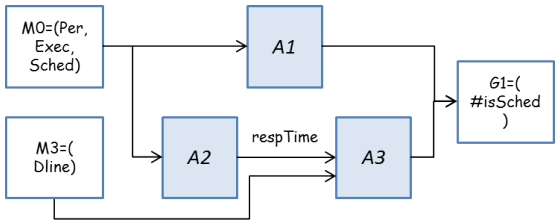

Fig. 2. Models, analyses and goals involve two kinds of properties : simple and meta-properties. Meta-properties are denoted with a # as prefix in the figure.

C. Contract

A contract is an abstract signature that can be indifferently related to a model, an analysis or a goal. More precisely, a contract describes in a formal way the properties required and provided by its related element.

Definition 4 : Contract**.** *A contract, related to an element (i.e., a model, an analysis or a goal), is a tuple : K=(I,O,A,G) :*

- *I are inputs : the set of properties required by the element,*

- *O are outputs : the set of properties provided by the element,*
- *A are assumptions : the set of meta-properties required by the element,*
- *G are guarantees : the set of meta-properties provided by the element.*

For the sake of clarity, we note $K_i.I$, $K_i.O$, $K_i.A$ or $K_i.G$ the elements of a contract, where $i \geq 0$ (A_i and G_i being reserved to express analyses and goals). Informally, $K(M)$, $K(A)$, $K(G)$ are contracts referring to : a model, an analysis, a goal.

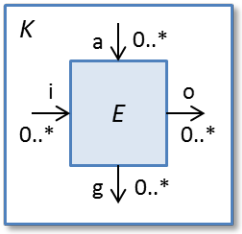

Fig. 3. A contract formally describes the properties required and provided by an element E (*i.e.*, a model, an analysis or a goal). Inputs-outputs use properties, assumptions-guarantees use meta-properties.

Example: We consider the example of Subsection III-A. *Processor utilization bound feasibility tests requires* several properties from the input model (Table I). In addition, the tests proposed by Liu and Layland [16] rely on a set of assumptions expressed over those properties (Table III) : the tasks should be periodic with an execution times fixed or upper bounded, the scheduling policy should be fixed, etc.

property	*description*	*association*
perTasks	tasks are periodic	$\{true, false\} \rightarrow task$
fixedExec	tasks with fixed execution times	$\{true, false\} \rightarrow task$
fixedSched	scheduling uses fixed priorities	$\{true, false\} \rightarrow scheduler$
...	other assumptions	

TABLE III
META-PROPERTIES REQUIRED TO APPLY LIU AND LAYLAND'S *processor utilization bound feasibility tests*.

Under Liu and Layland's assumptions, this analysis *provides* two kind of properties :

- the processor utilization rate (**U** property),
- a conclusion on the schedulability of the system (**isSched** meta-property).

The contract for this analysis is defined as follows : $K_3(A_1) = (I_3, O_3, A_3, G_3)$ with $I_3 = \{\textbf{Per}, \textbf{Exec}, \textbf{Sched}\}$, $O_3 = \{\textbf{U}\}$, $A_3 =$ "Liu and Layland's assumptions" $= \{\textbf{perTasks}, \textbf{fixedExec}, \textbf{fixedSched}, \ldots\}$ and $G_3 = \{\textbf{isSched}\}$.

Contracts defined for the various models, analyses and goals in Subsection III-B are listed in the table IV. We assume an additional analysis (A_0 in the following) to check Liu and Layland's assumptions on the input model.

contract	I	O	A	G
$K_1(M_0)$	$\emptyset$	**Per**, **Exec**, **Sched**	$\emptyset$	$\emptyset$
$K_2(A_0)$	**Per**, **Exec**, **Sched**	$\emptyset$	$\emptyset$	**perTasks**, **fixedExec**, ...
$K_3(A_1)$	**Per**, **Exec**, **Sched**	**U**	**perTasks**, **fixedExec**, ...	**isSched**
$K_4(A_2)$	**Per**, **Exec**, **Sched**	**respTime**	**perTasks**, **fixedExec**, ...	$\emptyset$
$K_5(M_3)$	$\emptyset$	**Dline**	$\emptyset$	$\emptyset$
$K_6(A_3)$	**respTime**, **Dline**	$\emptyset$	**perTasks**, **fixedExec**, ...	**isSched**
$K_7(G_1)$	$\emptyset$	$\emptyset$	**isSched**	$\emptyset$

TABLE IV
THE SET OF CONTRACTS FOR THE EXAMPLE IN SUBSECTION III-B.

D. Contracts complementarities and elements precedences

We note that inputs and outputs (assumptions and guarantees) defined within two distinct contracts can be complementary. In that situation, the underlying elements (which can models, analyses or goals) can be "connected" to each other. "Connections" are possible for level-compatible properties (between simple-properties, between meta-properties) and in a unidirectional way (from outputs to inputs, from guarantees to assumptions). Finally, we note that if two contracts are complementary then there is a precedence between their underlying elements. We give hereafter a more formal view on elements precedences.

Vertical precedence: A *vertical* precedence denotes a precedence between elements with respect to the production of meta-properties (guarantees to assumptions).

Property 1 : Vertical precedence**.** *Let :*

- *$\mathcal{E}$ be a set of elements, with $E_{current}$ and E_{next} be distinct elements (i.e., models, analyses or goals) from $\mathcal{E}$,*
- *$K_{current}$ and K_{next} be their contracts.*

A vertical precedence between $E_{current}$ and E_{next} exists, that is $next\_vertical(E_{current}, E_{next}) = true$, iff $K_{next}.A \cap K_{current}.G \neq \emptyset$.

Horizontal precedence: An *horizontal* precedence denotes a precedence between elements with respect to the production of simple-properties (outputs to inputs).

Property 2 : Horizontal precedence**.** *Let :*

- *$\mathcal{E}$ be a set of elements, with $E_{before}, E_{previous}, E_{current}$ and E_{next} be distinct elements (i.e., models, analyses or goals) from $\mathcal{E}$,*
- *$K_{current}$ and K_{next} be the contracts of $E_{current}$ and E_{next} respectively,*
- *$next\_vertical(E_{previous}, E_{current})$ and $next\_vertical(E_{before}, E_{next})$ be vertical precedences over elements of $\mathcal{E}$.*

An horizontal precedence between the two elements $E_{current}$ and E_{next} exists, that is

$next\_horizontal(E_{current}, E_{next}) = true$, *iff* $K_{next}.I \cap K_{current}.O \neq \emptyset$ *and* $(K_{current}.A = \emptyset$ *or* $next\_vertical(E_{before}, E_{current}) = true)$ *and* $(K_{next}.A = \emptyset$ *or* $next\_vertical(E_{previous}, E_{next}) = true)$.

Example: A graphical representation of the precedences between the models, analyses and goals using the contracts of the table IV is given on Figure 4. Following Property 1, there are 5 cases of vertical precedence. For instance, between an analysis and another : $K_3.A \cap K_2.G = \{\textbf{perTasks}, \textbf{fixedExec}, \ldots\} \neq \emptyset \implies next\_vertical(A_0, A_1) = true$. Following Property 2, there are 5 cases of horizontal precedence. For instance, between a model and an analysis : $K_3.I \cap K_1.O = \{\textbf{Per}, \textbf{Exec}, \textbf{Sched}\} \neq \emptyset \wedge K_1.A = \emptyset \wedge \quad next\_vertical(A_0, A_1) = true \implies next\_horizontal(M_0, A_1) = true$.

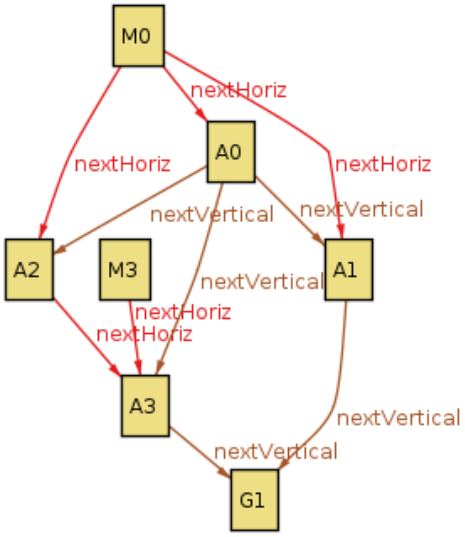

Fig. 4. Looking for the complementarities between the contracts in the table IV enables to set up the precedences between the models, analyses and goals of Subsection III-B.

From the use of models, analyses and goals precedences (Figure 4), it is possible to identify paths to execute the analyses in a suitable order to reach the goal.

IV. Exploiting contracts in our approach

In this section, we explain how contracts and contracts precedences – as defined in the previous section – are used in our approach to set up all feasible paths from an input model to a goal by crossing necessary analyses. We propose an implementation of the approach using Alloy.

A. Proposed approach

The previous examples illustrated how contracts can be defined for models, analyses and goals (see Table IV).

Our approach relies on contracts complementarities detection to set up necessary and feasible analysis path to attain a goal for an input model.

The approach consists in 4 steps (see Figure 5) :

1) Setting the configuration: which is made of :

- a set of models,
- a set of analyses intended to be applied on the models,
- a set of goals, *i.e.*, the simple- and/or meta-properties to compute with the analyses.

2) Declaration of the contracts: We derive contracts from the input configuration. As a result, the contracts declaration is made up of 3 different kinds of contracts :

- contracts for *models* abstract "initial elements" of the configuration that only provide outputs,
- contracts for *analyses* always describe required inputs, maybe a set of required assumptions and obligatorily the provided outputs and/or guarantees,
- contracts for *goals* abstract "terminal elements" that only require properties and/or meta-properties to be provided as a result of the analyses execution.

3) Search of the analysis paths: During this step, we use a) the contracts derived from the input configuration (step 2) together with b) the rules describing under which conditions two contracts are complementary (implementation of Properties 1 and 2 in Subsection III-D). Then, we proceed as follows :

i. given a) and b), we search the complementarities between the contracts,
ii. if a complementarity between two contracts exists, we force the precedence between the elements.

As a result of i) and ii), we find out the necessary and feasible paths to reach a goal from an input model by crossing all the necessary analyses. The implementation with Alloy discussed in the next subsection is optimal in the sense that it enables to identify all the instances of necessary analysis paths leading to a goal.

4) Execution of the analyses: Finally, analysis paths (resulting of step 3) can be used to execute the analyses in the correct order to obtain the required properties (goals). This step is not addressed in this paper.

B. Implementation with Alloy

We use Alloy [6] to implement steps 2 and 3. Alloy is a modeling language based on first order logic and relational calculus. It provides powerful analysis features : the analysis is performed by the Alloy analyzer, a tool capable of finding satisfiable instances for given Alloy models in a finite domain space using SAT-solving.

Firstly, the Alloy language is used to describe all the contracts of step 2 which are related to the input configuration in step 1. Basically, an Alloy model contains a collection of *signatures* which are the set of atoms manipulated by the Alloy analyzer. Signatures contain *fields* that define relations with other signatures.

Listing 1 gives the abstract signatures involved in the model : properties and contracts declaring different kinds of relations with properties. The `nextHoriz` and `nextVertical` relations (in the `Contract` abstract signature) make explicit the precedence order between elements underlying to the contracts (*i.e.*, models, analyses or goals).

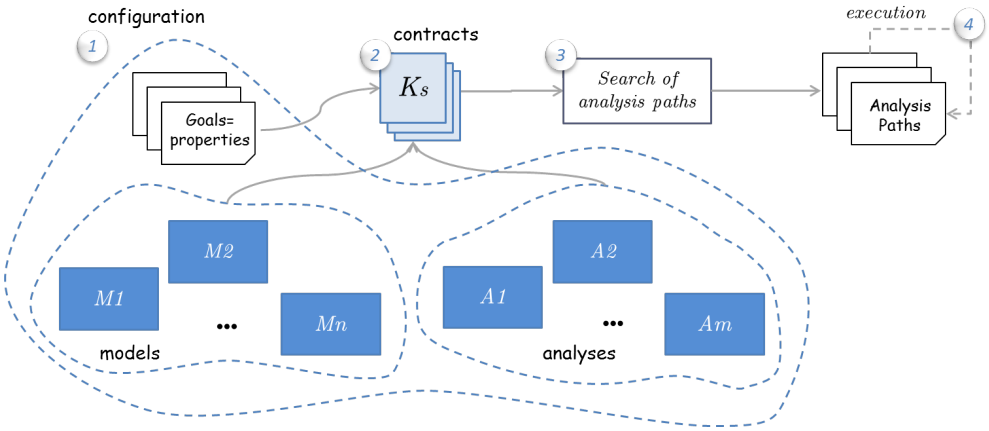

Fig. 5. Proposed approach. For a configuration (a set of models, analyses and goals) (1), the contracts are defined (2). The contracts are processed to identify the analysis paths to reach the goals (3). The analysis paths can be used to execute the analyses in a correct order to compute the required properties (4).

```
/* Abstract signatures in the Alloy model */
abstract sig Prop{}

abstract sig Contract{
   assumption: set Prop,       //required properties
   input: set Prop,
   guarantee: set Prop,        //provided properties
   output:set Prop,
   nextHoriz:set Contract,     // Prec: in->out
   nextVertical:set Contract   // Prec: assum->guar
}
```

Listing 1. Abstract signatures manipulated by the Alloy analyzer : contracts and properties.

```
/* Signatures issued from the configuration */

// A set of properties
one sig      Per, Exec, Sched, respTime, Dline,
             liu_test_assumptions,
             isSched extends Prop{}

// An example of contract
one sig A1 extends Contract{} {
       assumption=liu_test_assumptions
       input=Per+Exec+Sched
       guarantee=isSched
       output=none                }
```

Listing 2. A simple example of properties and contracts issued from a configuration involving a set of models, analyses and goals.

```
/* Constraints to be held true in instances */

//Horizontal precedence between two elements
fact HorizontalPrecedence{
  all c_current:Contract |
   c_current.nextHoriz={c_next:Contract|
      (c_current.output & c_next.input != none) and
      (all a :c_current.assumption| a in Contract.guarantee
           ) and
      (all a :c_next.assumption| a in Contract.guarantee)
}
```

Listing 3. Constraints to be held true in the instances found by the Alloy analyzer : Horizontal and Vertical Precedences.

Listing 2 is an excerpt of the Alloy model for the example in Subsection III-B : we use singleton to describe properties and contracts derived from the configuration.

In addition, the Alloy model (Listing 3) contains *facts* (`VerticalPrecedence` and `HorizontalPrecedence` facts) to detail under which conditions the `nextHoriz` and `nextVertical` relations between two contracts can be set.

The search for the analysis paths (steps 3i) and 3ii)) is handled by the Alloy analyzer. Alloy provides a SAT-solver that searches all the solutions that satisfy the specification in the Alloy model. Concerning the example of Subsection III-B, the Alloy analyzer finds only one instance. A graphical representation of the solution displayed in the Alloy GUI is given in Figure 4.

V. Experimentations

In this section, we first present a toolchain mixing modeling and analyses tools together with the Alloy tool. This toolchain is used to experiment our approach to investigate analysis paths on several case studies.

A. Settings

a) Toolchain: The toolchain used for system modeling and analysis is represented on Figure 6 :

- The systems are modeled with the architecture design language AADL [7] in the OSATE tool platform [15],
- Analysis tools :
  - MAST [12] and Cheddar [18] tool sets provide several analyses for evaluating tasks schedulability and/or response times.
  - RTaW-Pegase [19] and RTaW-Sim [20] are tools implementing analyses to compute networks traversal times. RTaW-Pegase focuses on network calculus for computing tight upper bounds of communication delays in AFDX networks. RTaW-Sim provides a set of analyses for the performance evaluation of CAN networks,
- REAL [21] theorems are directly applied on AADL models to check the assumptions of above-depicted analyses are held true in the AADL models,
- Alloy [6] is used to describe the contracts of the models and analyses implemented in the tools discussed above ; as well as the goals. The contracts are used by the Alloy analyzer to search the analysis paths (see Subsec. IV-B).

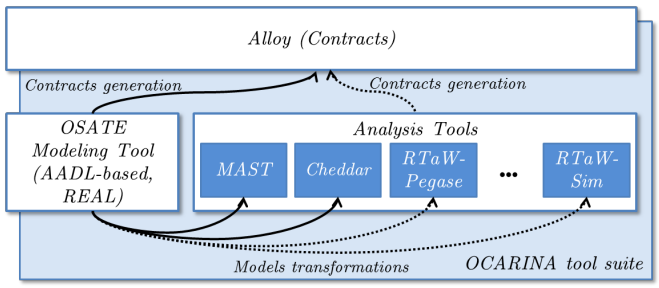

Fig. 6. The toolchain used to experiment our approach involves modeling and analysis tools. Alloy is used to declare the contracts related to models, analyses and goals and search the analysis paths.

Models transformations and contracts generations are partly backed by the OCARINA tool [22] – currently implemented transformations are represented with solid arrows in Figure 6.

b) Case studies models: We use AADL models for 5 case studies. The models are part of the AADLib project and are accessible at : http://www.openaadl.org/. The case studies are :

- M_1 : a model using the *ravenscar profile*,
- M_2 : a model for a simple *distributed real-time system*,
- M_3 : a model for the *mars pathinder system*,
- M_4 : a model for a simple *satellite system*,
- M_5 : a model for a *Flight Management System* (FMS).

The AADL models of the case studies are of different complexity. In the table V, we propose some metrics related to the complexity of the AADL models : lines of code, number of components, number of properties and average number of properties by component. For the remainder of the paper, we propose a complexity metric $\mathcal{O}_{AADL}$ encompassing together the number of components and the number of properties present in the AADL model :

$$\mathcal{O}_{AADL}(M_n) = \frac{NOC(M_n) \times NOP(M_n)}{NOC(M_5) \times NOP(M_5)} \quad (1)$$

According to $\mathcal{O}_{AADL}$, the model of the FMS is the most complex ($\mathcal{O}_{AADL}(M_5) = 1$). The AADL model using the ravenscar profile is the least complex : $\mathcal{O}_{AADL}(M_5) \approx 16 \times \mathcal{O}_{AADL}(M_1)$. We propose an additional model $M_6 = M_1 \cup M_2 \cup M_3 \cup M_4 \cup M_5$ which is obviously more complex than the model of the FMS : $\mathcal{O}_{AADL}(M_6) = 9 \times \mathcal{O}_{AADL}(M_5)$.

c) Analyses: We consider the analyses provided by the analysis tools of the toolchain (see Figure 6). We have 14 analyses in total. 7 analyses are provided by MAST, Cheddar, RTaW-Pegase and RTaW-Sim tools to compute tasks schedulability and/or response times and networks traversal times. In addition, we propose 4 analyses that use REAL theorems to check the analyses assumptions are held true in the AADL models. Finally, we consider 3 analyses also using REAL to compare tasks response times and network traversal times against tasks and packets deadlines. We use the latter analyses to conclude about tasks and messages schedulability.

d) Goal: We target a single goal which is to conclude about the schedulability of the tasks and messages used by the systems modeled with AADL.

case study	*LOC*	*NOC*	*NOP*	$\frac{NOP}{NOC}$	$\mathcal{O}_{AADL}$
M_1	148	7	39	5,57	0,06
M_2	337	20	57	2,85	0,25
M_3	395	24	51	2,125	0,27
M_4	464	27	85	3,148	0,5
M_5	753	47	97	2,064	1
M_6	2097	125	329	2,632	9,02

TABLE V
SEVERAL METRICS RELATED TO THE COMPLEXITY OF THE MODELS USED DURING THE EXPERIMENTATION : LINES OF CODE (LOC), NUMBER OF COMPONENTS (NOC), NUMBER OF PROPERTIES (NOP), AVERAGE NUMBER OF PROPERTIES DEFINED PER COMPONENTS (NOP/NOC) AND $\mathcal{O}_{AADL}$.

B. Results

Experimentation results discussed in this part have been processed on a computer with : a processor Intel Core i7-3770 (3,40 GHz), 8.00 GB of RAM ; the version 4.2 of Alloy using MiniSat as solver.

a) Analysis paths: For each case study, the Alloy analyzer successfully found an instance of analysis path : the instances can be used to execute the analyses in a correct order and conclude about the schedulability of the systems modeled with AADL. We do not discuss the produced instances in more details.

b) Contracts processing times: We are interested in the time taken by the Alloy analyzer to find the analysis paths enabling schedulability analysis of the various models : the *contracts processing times*. The contracts processing time (CPT) is the time required by the Alloy analyzer to take into account the contracts and constraints present in the Alloy model and find all the solutions satisfying that model. It encompasses two dimensions : 1) the *generation time* (GT) of the formulas to be handled by the solver and 2) the *resolution time* (RT) of the formulas to provide all the solutions. This is simply summarized :

$$CPT = GT + RT \quad (2)$$

Contracts processing times (CPT) experienced with the various AADL models are outlined on Figure 7. The generation times (GT) increase exponentially with the complexity of the AADL models ($\mathcal{O}_{AADL}$). The best case ($GT = 639ms$) corresponds to the *ravenscar profile* model (M_1). The worst case is attained with the model of the *flight management system* (M_1) with a $GT = 121159ms$ ($\approx 2min$) to generate the formulas to solve. In the scenario where we consider all the models at the same time (M_6), the generation time is multiplied by 20 ($GT \approx 40min$) regarding the scenario involving the FMS only.

We observe that, for all the case studies, almost entire part of the contracts processing times (CPT) is devoted to the generation of formulas to solve (GT). The resolution times of the formulas themselves (RT) never exceeds 1 second ($RT = 856ms$ being the worst-case experienced).

c) Summary: We showed that our approach is applicable on configurations (*i.e.*, sets of models, analyses and goals) of realistic complexity in an affordable time. Despite of the important resolution space to handle, we are able to identify

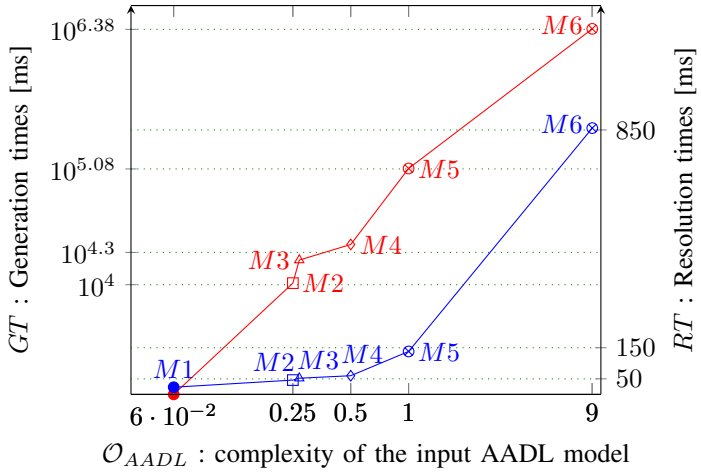

Fig. 7. Contracts processing time $CPT = GT + RT$ dependence of the input model complexity $\mathcal{O}_{AADL}$

all the necessary analysis paths to reach the analysis goal in an reasonable time (the worst processing time is about 2 minutes). In addition, we experienced scalability of our approach : we applied our strategy on a configuration mixing all the models (which represent 5 models, 125 components and 329 properties). Notwithstanding that the strategy is non-optimal, our approach is capable of finding out all the solutions. A better way to handle such a resolution space is to treat the models independently and successively. The processing time is then reduced from 40 to less than 3 minutes.

VI. Conclusions

We presented an approach to organize the execution of analyses in MBE by targeting *goals*. We used *contracts* 1) to capture the properties required and provided by any element (which can be a model, an analysis or a goal) and 2) to identify the precedences between these elements. Our implementation with Alloy is optimal in the sense that if any analysis path towards a goal exists, the Alloy analyzer will always find it. We finally shown that our approach is capable to organize analyses provided by existing tools with complex AADL models in a reasonable time.

Future works can have manifold directions. Firstly, we intend to extend our work to other modeling and analysis concerns than architectural ones. Another upcoming issue will be to semantically interpret the analysis paths before executing the analyses. At last, we think to improve our approach to take into account the quality related to the analysis paths (*e.g.*, rapidity of the executions, precision of the results, etc).

Acknowledgment

The authors would like to thank Loïc Gammaitoni (from the CSC Research Unit) for the support and advices dispensed in the use of Alloy.

References

[1] B. Berthomieu, J.-P. Bodeveix, C. Chaudet, S. Dal Zilio, M. Filali, and F. Vernadat, "Formal verification of AADL specifications in the Topcased environment," in *Reliable Software Technologies–Ada-Europe 2009*, pp. 207–221, Springer, 2009.

[2] Y. Ouhammou, *Model-based Framework for Using Advanced Scheduling Theory in Real-Time Systems Design*. PhD thesis, dec 2013.

[3] V. Gaudel, A. Plantec, F. Singhoff, J. Hugues, P. Dissaux, and J. Legrand, "Enforcing Software Engineering Tools Interoperability: An Example with AADL Subsets," in *IEEE International Symposium on Rapid System Prototyping (RSP)*, (Montreal, Canada), 03-04 October 2013.

[4] I. Ruchkin, D. De Niz, S. Chaki, and D. Garlan, "Contract-based integration of cyber-physical analyses," in *Proceedings of the 14th International Conference on Embedded Software*, 2014.

[5] A. Sangiovanni-Vincentelli, W. Damm, and R. Passerone, "Taming dr. frankenstein: Contract-based design for cyber-physical systems*," *European journal of control*, vol. 18, no. 3, 2012.

[6] D. Jackson, *Software Abstractions: logic, language, and analysis*. MIT press, 2012.

[7] SAE/AS2-C, "Architecture Analysis & Design Language V2 (AS5506A)," Jan. 2009.

[8] D. Redman, D. Ward, J. Chilenski, and G. Pollari, "Virtual Integration for Improved System Design," in *Proceedings of The First Analytic Virtual Integration of Cyber-Physical Systems (AVICPS) Workshop*, (San Diego, USA), November 2010.

[9] M. Walker, M.-O. Reiser, S. Tucci-Piergiovanni, Y. Papadopoulos, H. Lönn, C. Mraidha, D. Parker, D. Chen, and D. Servat, "Automatic optimisation of system architectures using EAST-ADL," *Journal of Systems and Software*, vol. 86, no. 10, 2013.

[10] F. Cadoret, *Génération stratégique de code pour la maîtrise des performances de systèmes temps-réel embarqués*. PhD thesis, Télécom ParisTech, 2014.

[11] F. Singhoff, J. Legrand, L. Nana, and L. Marcé, "Scheduling and memory requirements analysis with aadl," in *ACM SIGAda Ada Letters*, vol. 25, pp. 1–10, ACM, 2005.

[12] M. González Harbour, J. Medina, J. Gutiérrez, J. Palencia, and J. Drake, "MAST: An Open Environment for Modeling, Analysis, and Design of Real-Time Systems," in *1st CARTS Workshop*, (Aranjuez, Spain), October 2002. http://mast.unican.es/.

[13] P. Derler, E. A. Lee, S. Tripakis, and M. Törngren, "Cyber-physical system design contracts," in *Proceedings of the ACM/IEEE 4th International Conference on Cyber-Physical Systems*, pp. 109–118, ACM, 2013.

[14] I. Ruchkin, D. De Niz, S. Chaki, and D. Garlan, "ACTIVE: A Tool for Integrating Analysis Contracts," 2014.

[15] OSATE2 : An open-source tool platform for AADLv2. https://wiki.sei.cmu.edu/aadl/index.php/Osate_2.

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

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

[18] F. Singhoff, J. Legrand, L. Nana, and L. Marcé, "Cheddar: a flexible real time scheduling framework," in *ACM SIGAda Ada Letters*, vol. 24, pp. 1–8, ACM, 2004. http://beru.univ-brest.fr/~singhoff/cheddar/.

[19] M. Boyer, J. Migge, and M. Fumey, "PEGASE - A Robust and Efficient Tool for Worst-Case Network Traversal Time Evaluation on AFDX," in *SAE AeroTech Congress & Exhibition*, (Toulouse, France), October 18-21 2011. http://www.realtimeatwork.com/software/rtaw-pegase/.

[20] RtaW-Sim : Controller Area Network simulation and configuration. http://www.realtimeatwork.com/software/rtaw-sim/.

[21] O. Gilles and J. Hugues, "Expressing and enforcing user-defined constraints of AADL models," in *Proceedings of the 5th UML& AADL Workshop*, (Oxford, United Kingdom), March 2010.

[22] G. Lasnier, B. Zalila, L. Pautet, and J. Hugues, "Ocarina : An Environment for AADL Models Analysis and Automatic Code Generation for High Integrity Applications," in *Proceedings of the 14th Ada-Europe International Conference*, (Brest, France), June 8-12 2009. http://www.openaadl.org/ocarina.html.