Formal Verification of AADL Models Using UPPAAL

Fernando Gonçalves
David Pereira*
Eduardo Tovar*
Leandro Becker

*CISTER Research Centre

CISTER-TR-171101

2017/11/07

Formal Verification of AADL Models Using UPPAAL

Fernando Gonçalves, David Pereira*, Eduardo Tovar*, Leandro Becker

*CISTER Research Centre

Polytechnic Institute of Porto (ISEP-IPP)

Rua Dr. António Bernardino de Almeida, 431

4200-072 Porto

Portugal

Tel.: +351.22.8340509, Fax: +351.22.8321159

E-mail: fersg@isep.ipp.pt, dmrpe@isep.ipp.pt, emt@isep.ipp.pt

http://www.cister.isep.ipp.pt

Abstract

Formal Verification of AADL Models Using UPPAAL

Fernando Silvano Gonçalves*, David Pereira†, Eduardo Tovar†, and Leandro Buss Becker*
*Department of Automation and Systems, UFSC - Florianópolis, SC, Brazil 88040-900
Email: fernando.goncalves@posgrad.ufsc.br,leandro.becker@ufsc.br
†CISTER/INESC-TEC Research Centre, ISEP/IPP - Porto, Portugal
Email: {dmrpe,emt}@isep.ipp.pt

***Abstract*—Cyber-Physical Systems (CPS) are known to be highly complex systems which can be applied to a variety of different environments, covering both civil and military application domains. As CPS are typically complex systems, its design process requires strong guarantees that the specified functional and non-functional properties are satisfied on the designed application. Model-Driven Engineering (MDE) and high-level specification languages are a valuable asset to help the design and evaluation of such complex systems. However, when looking at the existing MDE tool-support, it is observed that there is still little support for the automated integration of formal verification techniques in these tools. Given that formal verification is necessary to ensure the levels of reliability required by safety critical CPS, this paper presents an approach that aims to integrate the Model Checking technique in the CPS design process for the purpose of correctly analyzing temporal and safety characteristics. A tool named *ECPS Verifier* was designed to support the model checking integration into the design process, providing the generation of timed automata models from high-levels specifications in AADL. The proposed method is illustrated by means of the design of an Unmanned Aerial Vehicle, from where we derive the timed automata models to be analyzed in the UPPAAL tool.**

I. Introduction

Applications that integrate complex embedded software systems to control physical processes are widely known as Cyber-Physical Systems (CPS). The CPS design is a multidisciplinary process that involves different teams working cooperatively to address the application's requirements. Adequate tools and methods are of utmost importance to support and guide the teams in order to increase the potential of the project success [1]. To ensure the requirements fulfillment along the design process, strong system analysis is needed [2].

Model-Driven Engineering (MDE) has been considered adequate to support CPS design. By using MDE, complementary models can be created for representing the different system dimensions. At least three different models should be used for CPS design, including the physical behavior representation, the control system design, and the system architecture specification [3], [4], [1].

Considering the typical high complexity of CPS and the need for a strong analysis to ensure the design correctness, formal verification becomes a natural candidate to become part of the overall CPS design process [5]. There exists different approaches therefore, like Model Checking (MC), Theorem Proving, and Runtime Verification (RV). Each method has its pros and cons, namely: MC suffers from the state explosion problem; Theorem Proving requires highly technical knowledge and despite its latest developments it still faces many automation problems (due to foundational limitations on the supporting logical theories); and RV brings overhead to the CPS since monitors have to be coupled with system components and process extra information from events.

Aiming to avoid the state space explosion in Model Checking, different design techniques can be applied. Examples of these techniques include: abstractions and reduction of unnecessary states; the use of symbolic model checking, applying binary decision diagrams and symbolic algorithms; the partial order reduction that concerns on the identification of interleaving sequences, eliminating redundancy and so reducing the state space.

Given this scenario, MC seems to be the more natural approach for our work since it conforms MDE practices and can be fully automated. Well-known MC tools are UPPAAL [6], HyTech [7], Kronos [8], among others. Given that UPPAAL performance is much better than other tools like HyTech, and Kronos in time and space [9]. In this context, on the present work we adopt MC by using the UPPAAL tool [6].

In order to support the formal verification based on MC, a timed automata should be created to express the system behavior, supporting the evaluation of different properties such as safety, reachability, liveness, and deadlock [10]. However, generating these representations is not a simple task and requires sufficient knowledge of the design team to correctly express the system properties. To automate the timed automata construction, it is our claim that a model transformation can be performed using as input the architectural representation.

This paper presents an approach to apply a MC technique on the CPS design process, allowing the timing and error properties evaluation. The proposed approach includes a model transformation process that based on the architectural model in AADL [11], generates a Timed Automata representation that conforms the UPPAAL tool [6]. The tool named *ECPS Verifier* was created in the context of this proposal to support such model transformation.

The reminder parts of this paper are organized as follows. Section II describes the proposed activities to integrate the formal verification method in the CPS design process. Section III details the *ECPS Verifier* tool. Section IV presents the a case study related to design of an Autonomous Aerial Vehicle (UAV) by using the proposed approach. Section V discusses the results obtained by performing the formal verification to the UAV system. Section VI presents the related work. Finally, Section VII presents our conclusions.

II. Design Method and Activities

Aiming to support the evaluation of the CPS properties by using MC a design method was created, allowing designers to incorporate this technique on the CPS design process. In this sense different properties from the system under design can be evaluated, such as liveness, reachability, deadlock freeness, and others. To support this method a tool named *ECPS Verifier* was designed.

ECPS Verifier performs the model transformation from an architectural model in AADL to a network of timed automata devoted to be analyzed via MC. It is assumed the use of AADL components designed in the OSATE tool to represent the architectural model and the timed automata suited with the UPPAAL tool to represent the system behavior.

An important aspect to be highlighted is that the problem under consideration is not restricted to simply representing the CPS behavior and analyzing its properties. It happens that the designer must properly plan how to express the system properties according to the proposed architecture and the set of system devices, as well as to evaluate how correctly it expresses the system properties in order to provide guarantees that the system fulfills its restrictions.

The proposed design method consists in a set of activities to be conducted by the designers to perform the model generation and the evaluation of the system properties. These activities are detailed in the following section.

A. Design Activities

The proposed method defines a sequence of activities to integrate the MC in the CPS design process, allowing to evaluate the system properties and restrictions. These activities are presented in the Fig. 1, and aims to properly guide the designers to define the system behavior and the possible device faults, integrating these information on the architectural model (by the use of AADL annexes) to support the MC evaluation.

In a nutshell, the method starts with an activity that targets analyzing the possible CPS faults (*Fault-trees design*), having as output the a fault-tree definition. Based on the defined faults, a refinement is performed in the architectural model (*Fault properties integration on architectural model*) by integrating the fault's characteristics. This activity is supported by using the AADL Error Annex (EA) [12]. In addition, the threads behavior are also evaluated, refining its properties by using the AADL Behavioral Annex (BA) [13] (*Behavior properties refinement*). The complete architectural model is used as input to the model transformation process (*Model transformation*), performed by the *ECPS Verifier* tool, generating the UPPAAL automata. Then the system requirements are encoded as temporal formulas in the UPPAAL syntax (*Formal properties specification*), generating a complete UPPAAL model. Finally, the system is evaluated (*System analyzis*).

Activity 1. ***Definition of fault-trees*** *- In this activity, the designer analyzes the CPS characteristics aiming to specify the possible failure events. These information include the proposed mission/task for the CPS, its configuration, the set of required devices, and restrictions. The output of this activity is a set of fault-trees to be used by the designers.*

Based on the fault-trees it becomes possible to evaluate the implications its events on the system behavior and define alternatives to mitigate their effects. These representation aims to provide a top view of the possible system failures that should be considered during the design process.

Activity 2. ***Integration of fault properties in the architectural model*** *- By using the EA the fault-trees are added in the architectural model, representing the possible failures, and the associated probability for each error event occurrence. These information is used to evaluate the failures impact in the designed system, as well as to define alternatives to mitigate their effects. The output of this activity is a refined architectural model, integrating the error properties.*

As the error properties, to provide the automata generation the defined behavioral CPS properties needs to be refined.

Activity 3. ***Refinement of behavior properties in the architectural model*** *- Based on the system characteristics in this activity the behavioral properties are refined, by using the AADL threads and the BA. In this way, properties are specified such as execution states, system variables, system transitions, subprograms access, and temporal characteristics. The output of this activity is a complete architectural model, integrating error and behavioral properties.*

Regarding the usual AADL models design, its observed that these models typically contain behavioral information that allow designers to evaluate system properties. However, to the timed automata extraction, some properties need to be refined, adding information related to the automata guards and variables declaration. Such complete AADL model is suitable to be submitted to the transformation process.

Activity 4. ***Formal verification*** *- In this activity the designers perform the system evaluation based on the generated architectural model. This activity is divided into three sub-activities (three last blocks from Fig. 1), that address each part of the formal verification process.*

The verification process is based on the timed automata (UPPAAL model) that is generated by performing a model transformation, which input is an AADL model.

Activity 4.1. ***Model transformation*** *- An UPPAAL model is created by means of a model transformation process from the AADL specification. Such UPPAAL model is composed by a set of templates that describe the AADL threads and devices characteristics. The transformation is based on a set*

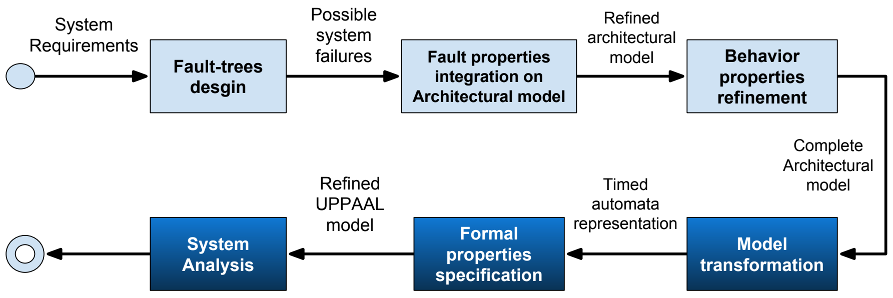
System Requirements
Fault-trees desgin
Possible system failures
Fault properties integration on Architectural model
Refined architectural model
Behavior properties refinement
Complete Architectural model
Model transformation
Timed automata representation
Formal properties specification
Refined UPPAAL model
System Analysis

Fig. 1: ECPS Verifier Workflow.

of rules that map the source and target models, as detailed in the Section III.

Activity 4.2. ***Formal properties specification*** *- In this activity, the designer creates formal expressions that represent model properties that must be evaluated. These properties represent the characteristics and restrictions that the system needs to meet in order to fulfill its objective. In this sense properties like liveness, reachability, safety, and deadlock occurrence can be evaluated.*

However, formally representing the system properties - in this case using the UPPAAL syntax (and the UPPAAL-SMC syntax for those properties that have probabilities associated) - is not a simple task. This comes from the fact that these expressions are dependent of the adopted formal language and, in addition, describing the system restrictions using a formal language is not trivial and requires considerable knowledge.

Activity 4.3. ***System analysis*** *- The system analysis is performed by checking the validity of the formally specified properties carried out in Activity 4.2, against the models derived in Activity 4.1. Depending on the results, system changes may be required (which implies regenerating of, at least, some of the existing automata), therefore adjusting the system to satisfy its intended behaviors.*

During the system properties evaluation it is possible to observe if the system meets its requirements. This implies that the system threads meet their deadlines, that the safety properties are satisfied, and that no deadlocks are found except if an error occurs. The performed analysis on the UAV system is detailed in Section V.

III. Model Transformation Tool *ECPS Verifier*

To automate the UPPAAL model (automata) generation from the AADL model, and to make this process less error prone, we have developed a tool named *ECPS Verifier*. This tool follows the MDE principles, which states that a mapping between the source (AADL) and the target (UPPAAL) models is defined by means of transformation rules. Auxiliary structures are required to provide the automata execution management. The transformation rules make use of metamodels from both source and target models, as further detailed.

A. Related Metamodels

The AADL (source) metamodel is composed by a root *System* that contains a set of subcomponents representing others AADL components, such as *Processes*, *Threads*, *Ports*, and *Connections*. It is important to highlight that the *Thread* component encapsulates the system behavior, so it can contain subcomponents that may represent software calls (black-box component) or it can detail the behavior by means of state machines using the Behavior Annex (BA). Given that our approach is more focused in the AADL software components, the single hardware element under consideration is the *Device*.

Overall, the target metamodel is composed by a set of templates that encapsulate the timed automata, and by a set of queries used for the model evaluation. UPPAAL models are composed by at least one template containing a timed automata, which is decomposed into a set of states and transitions. These transitions can incorporate restrictions (guards and synchronizations points), and can also incorporate actions expressed like in imperative programming language - it allows declaring variables and making function calls. Such actions are defined in the updates definitions. Finally, a set of queries describing the properties to be evaluated can also be attached to the model.

Due to the lack of space, the created metamodels are not illustrated in the paper but, however, they are available in the ECPS Verifier repository[1].

B. Transformation Process

To perform the transformation process, both a parser and a transformation engined were created, all developed in Java as plug-ins from Osate tool. The parser is responsible to mapp to memory the source (AADL) textual model, in accordance to its related metamodel elements.

The transformation engine is responsible to perform the automata generation based on the source AADL model. This engine is composed by the following rules:

- AADL *Thread* components are mapped to UPPAAL templates representing its behavior and characteristics like states, guards, invariants, periodicity, priority, and others.

[1]https://github.com/fernandosgoncalves/ECPSVerifier/

- AADL *Devices* are also mapped as UPPAAL templates, detailing its behavior coupled with its possible failures.
- The set of input and output ports from the AADL model are used as basis to the variables and UPPAAL channels declarations, representing the system communication.

The model for devices are initially composed by two states, an *idle* and an *execution* state. The latter represents the actuation (for actuators) or processing (for sensors). According to its associated fault-tree, if it includes erroneous behavior, additional states are included. Probabilities are associated to each error state, representing the occurrence distribution of the considered errors.

C. Scheduler Component and Auxiliary Functions

To support schedulability analysis using UPPAAL, a scheduler template was designed to be automatically included into the UPPAAL model along the transformation process. Thereby, designers only need to create in the AADL model a device named *Scheduler*. Currently, only the Rate-Monotonic (RM) scheduling algorithm [14] is supported. The template automata model is presented in Fig. 2.

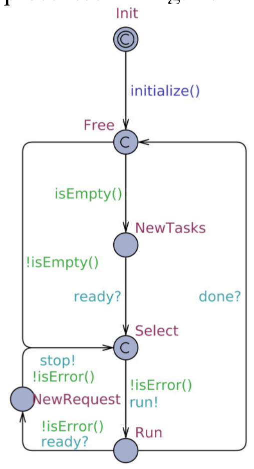

Fig. 2: Scheduler model.

When started (*Init*), the scheduler runs a function (*initialize()*) that performs the tasks inclusion on the scheduling queue according to their priorities and moves to state *Free*. At this state its verified if the scheduler queue is empty (*isEmpty()*). If so, the system waits for the next task activation (*ready?*), on the *NewTasks* state. However, if the queue is not empty, the task on the front of the queue (of highest priority) is selected. So the task execution is started (*Run* state). If a new task is available during another task's execution (*NewRequest* state), the scheduler comes to action and compares the priorities of the running and the recently arrived task. If the running task has higher priority, than the running task is resumed, otherwise the running thread is preempted and the new task is executed.

To allow representing in the AADL model some required UPPAAL properties - and ensuring the mapping between UPPAAL properties and the AADL model, a few design conventions were established. For instance, these conventions involve AADL subprograms, which should be created to represent specific UPPAAL components, such as: (1) invariants; (2) rate expressions; and (3) transition guards. It also requires the addition of (4) functions to support the scheduling mechanism and data types to represent (5) UPPAAL channels (*chan*) and (6) clock variables (*clock*). An example of such set of AADL subprograms is presented in Fig. 3.

Lines 2 to 5 represent the UPPAAL invariants, including their names and invariants expression. Lines 7 to 11 represent the UPPAAL transition guards - this is needed by the fact that guards of the AADL BA do not support the use of functions in the guards expression. Thereby, this subprogram is composed by three inputs describing: the UPPAAL function that should be declared as a subprogram, the guard operator, and the value used in the expression.

```
1  -- ** SUBPROGRAMS **
2  SUBPROGRAM sp_invariant
3   FEATURES
4    state: IN PARAMETER String; inv: IN PARAMETER String;
5  END sp_invariant;
6
7  SUBPROGRAM sp_guard
8   FEATURES
9    function: in PARAMETER String; operator: in parameter String;
10   value: in parameter String;
11 END sp_guard;
12
13 -- ** SCHEDULER FUNCTION **
14 SUBPROGRAM sp_add
15  FEATURES id: IN PARAMETER;
16 END sp_add;
17
18 SUBPROGRAM sp_yield
19 END sp_yield;
20
21 SUBPROGRAM sp_head
22 END sp_head;
23
24 -- ** DATA TYPES ***
25 DATA chan END chan;
26
27 DATA clock EXTENDS Base_Types::Integer END clock;
```

Fig. 3: AADL design conventions.

The scheduler mechanism also requires a set of functions to help its runtime engine, allowing it to: (1) add a new task in the scheduler queue (lines 14 to 16); (2) yield the currently running task (lines 18 to 19); (3) return the currently running task (lines 21 to 22). The data types on lines 25 and 27 represent, respectively, the UPPAL channels - by convention all channels are defined as broadcast - and the UPPAAL clock variables.

Due to the fact that the AADL model can be composed by multiple systems and implementations, it is here defined that the first translated system will be the root. In this sense the designer needs to declare first the root system in the AADL file. Once the transformation is ended, the system becomes suitable for MC. Although typical AADL models make use of multiple source files, our approach only supports the use of a single file.

It is important to highlight that besides some design conventions can be adopted, in order to provide the models mapping, its verified that the AADL model support the representation of required structures to provide automata generation. In this sense, regarding that during the architectural model construction the threads behavior is specified using the BA, and that the model has a high information refinement, its possible to say that only the design conventions needs to be included on the AADL base model to enable the transformation process.

IV. Design of Sensing and Actuation Subsystems of an UAV

The application of the activities presented in the previous section is illustrated here by means of the design - and verification - of the sensing and actuation subsystems of an Unmanned Aerial Vehicle (UAV). Such UAV was conceived in a related project named ProVant[1].

Overall, the UAV design was conducted by the use of the design method proposed in [15]. Such design comprised the creation of a Simulink functional model for the control system, and of an AADL architectural model to represent the UAV embedded system. The designed architectural model is used as basis to the approach presented in this paper.

The top-level view of the UAV architectural model is depicted in Fig. 4. This model integrates the control system (managed by the *pi_control_system* process, line 4) with the sensing and actuation subsystems (managed by the *pi_est_act* process, line 5). Coupled with these processes are the set of system devices (lines 7 to 10) that represent the required UAV sensors and actuators.

```
1 SYSTEM IMPLEMENTATION UAV.impl
2  SUBCOMPONENTS
3    --PROCESS
4    pi_control_system: PROCESS p_control_system.impl;
5    pi_est_act: PROCESS p_est_act.impl;
6    --DEVICE
7    di_esc_r: DEVICE d_esc.impl; di_esc_l: DEVICE d_esc.impl;
8    di_servo_r: DEVICE d_servo.impl; di_servo_l: DEVICE
        d_servo.impl;
9    di_gps: DEVICE d_gps.impl; di_sonar: DEVICE d_sonar.impl;
10   di_imu: DEVICE d_imu.impl;
11 CONNECTIONS
12   C1: PORT di_gps.position -> pi_est_act.position;
... Here goes all others connections (lines 13 to 32)
33 END UAV.impl;
```

Fig. 4: UAV model with sensing and actuation process.

Fig. 5 contains the expansion of the sensing and actuation process (*pi_est_ act*). It is possible to observe the set of threads that are responsible for interfacing with these devices, sending the control references to actuators (thread *ti_signalTrans formation*, line 5) and providing the system behavior estimation (threads *ti_sensing* and *ti_positionEst*, lines 3 and 4).

```
1  PROCESS IMPLEMENTATION p_est_act.impl
2   SUBCOMPONENTS
3     ti_sensing: THREAD t_sensing.impl;
4     ti_positionEst: THREAD t_positionEst.impl;
5     ti_signalTransformation: THREAD t_signalTransformation.impl;
6   CONNECTIONS
7     C1: PORT distance -> ti_behaviorEst.distance;
... Here goes all others connections (lines 8 to 25)
26 END p_est_act.impl;
```

Fig. 5: AADL representation of sensing and actuation process.

To be analyzed using MC, the designed AADL model must be subject of the activities presented in section II, as follows.

Initially, fault-trees are designed to detail the possible failures that can be associated to the systems devices (Activity 1). Regarding the UAV devices, fault-trees were defined for the Servomotors, Electronic Speed Controllers (ESCs), Inertial Measurement Unit (IMU), Global Positioning System (GPS), and Sonar. Fig. 6 presents the fault-tree designed for the GPS, containing information extracted from its datasheet [16].

[1]http://provant.paginas.ufsc.br

Considering our focus on design of the UAV software components, at this time only the logical failures are considered (dark blue blocks of Fig. 6). These failures are flagged by the device, sending in the message package a specific bit to each kind of reported error. The designed faults-trees are also based on the works related to UAV faults presented in [17], [18].

Based on the preliminary AADL model presented in Fig. 4,

The specified fault properties are integrated on the AADL model presented in Fig. 4 by using the AADL EA, increasing the model details (Activity 2). Fig. 7 shows the GPS EA specification according to its fault-tree.

Regarding the behavior specification coupled with the devices error properties, the threads behavior properties are analyzed and extended if required (Activity 3). In this activity the designers detail some properties such as the threads states, transitions, guards, the subprograms access, temporal characteristics, among others. The proposed design conventions can be used at this time, in order to provide a proper model mapping. The behavioral characteristics are presented in the Fig. 8, detailing: the thread periodicity; the execution time; its period; the thread priority; its deadline (lines 3 and 4); the thread variables (line 7); the set of execution states (lines 9 to 10); and the set of system transitions (lines 12 to 31).

Once the AADL model becomes properly complemented, it is subject of model transformation to be further analyzed using MC (Activity 4). In this way, the UPPAAL timed automata is generated by using the proposed transformation tool. As result, the UAV system is mapped into a set of templates. Fig. 9 depicts a template structure representing the UAV position thread that is responsible to provide the GPS interface and to estimate the system position.

It can be observed in this template the set of predefined properties previously detailed on the AADL model. This includes characteristics like the thread's periodicity, the devices interface, the scheduler functions, among others. Regarding the UAV devices, the GPS sensor has an interface with the presented thread. In this sense, the generated template that represents this sensor structure is shown in Fig. 10.

Regarding the GPS model its observed the set of main execution states (*idle* and *processing*), as well as the coupling of the *processing* state with the predefined set of possible failures and its probabilities of occurrence.

Once the automata representations are generated, the designer needs to formally define the set of properties that will be evaluated (Activity 4.2), thus they need to define them as UPPAAL queries. These queries are written by the use of TCTL language and detail properties like reachability, safety, and deadlock freeness for example. A detailed definition of the system queries is presented in Section V.

In terms of the system analysis, the UPPAAL tool performs a state space exploration to validate the designed queries. The system evaluation also includes queries related with defined probabilities, by the use of UPPAAL-SMC. The details related to the performed system analysis on the UAV system are detailed in the following section.

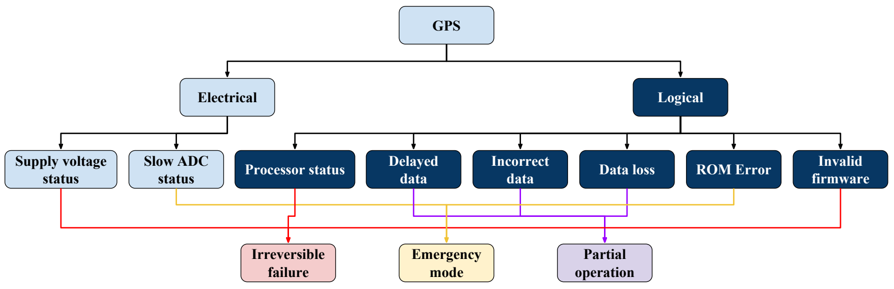

Fig. 6: GPS fault tree representation.

```
ERROR BEHAVIOR gpsError
 EVENTS
  processorError : error event; delayedData : error event;
  incorrectData : error event; dataLoss : error event;
  romError : error event; invalidFirmware : error event;
 STATES
 operational : initial state; partialOperation: state;
 emergencyMode : state; irreversibleFailure : state;
TRANSITIONS
 T1 : operational -[ delayedData ]-> partialOperation;
 T2 : operational -[ incorrectData ]-> partialOperation;
 T3 : operational -[ dataLoss ]-> partialOperation;
 T4 : operational -[ romError ]-> emergencyMode;
 T5 : operational -[ processorError ]-> irreversibleFailure;
 T6 : operational -[ invalidFirmware ]-> irreversibleFailure;
END BEHAVIOR;
```

Fig. 7: AADL GPS error representation.

V. UAV Properties Evaluation

In this section we present the preliminary experiments carried out in the UAV system while formulating the method described in this paper. These experiments involved mostly two efforts: i) the construction of the various timed automata, including those presented in Figs. 10 and 9. They captured, respectively, the GPS properties and the task that is responsible for providing the UAV position; ii) the specification of the relevant properties of the UAV model covering safety, liveness, respect of deadlines, and causes for deadlocks.

In the rest of this section it is presented some examples of the UPPAAL specifications that were developed and checked.

a) **(Example-Spec-1):** All tasks run at least once, and therefore reach a state where they are idle. This liveness property is, as expected, only partially fulfilled since a task can reach an error state (e.g., due to a device or a function failure) during its execution. It is expressed by the following UPPAAL formula, where $T_1 \dots T_k$ denote all the model tasks.

$$\mathsf{E} <> \ (T_1.Idle \text{ and } \dots \text{ and } T_k.Idle), \tag{1}$$

b) **(Example-Spec-2):** Whatever the task we consider, that task in executing only if the scheduler is running or processing a new request. This safety condition imposes therefore that not exits task running out of the scheduler control, and is expressed as:

$$\mathsf{A[\,]}\ \text{not}\ (\phi \text{ and not } (Scheduler.Run \text{ or } Scheduler.NewRequest)), \tag{2}$$

such that ϕ specifies all possible task states that corresponding its execution. The way to state this for a specific T_i is through the term

$$\phi = (\mathsf{T_i.State_1} \ \text{ or } \ \dots \ \text{ or } \ \mathsf{T_i.State_k})\ , \tag{3}$$

where T_i represents a system thread, i represents a task in $[0, N]$ and N define the number of tasks. The $State\_k$ denotes their execution states (which are a subset of task execution states).

c) **(Example-Spec-3):** Considering the set of system tasks, a task is running only if its execution time is smaller than it is deadline. This is expressed by the following formula:

$$\mathsf{A[\,]}\ \text{not}\ (\text{forall}\ (i : \text{int}[0, N])((T_i.State_1 \text{ or } \dots \text{ or } T_i.State_k) \text{ and } (T_i.ax > D[i]))) \tag{4}$$

where T_i represents system threads, $D[i]$ its prescribed deadline, the ax field its current total execution time, and each $State_1, \dots, State_k$ represent task execution states.

d) **(Example-Spec-4):** A system deadlock is possible only if one of the system threads is on error state. This property was checked by specifying the formula scheme:

$$\mathsf{A[\,]}\ \text{deadlock imply}\ (T_1.\text{Error or } \cdots \text{ or } T_i.\text{Error}) \tag{5}$$

with $T_1 \dots T_i$ denoting the tasks of the system, and deadlock is $UPPAAL$'s keyword that denotes that there is a deadlock in the model.

e) **(Example-Spec-5):** As a final example we define a specification that brings statistical analysis of the model. For this, we used the Statistical Model Checking (SMC) facilities that the version of UPPAAL that we have adopted.

This specification refers to the probability of the system to reach an actuator error state due to its execution. This probability condition is expressed by the formula:

$$\text{Pr}\ [\le 12000]\ (<> \text{Actuator}(i).\text{EmergencyMode}), \tag{6}$$

where the bound defined ($\le$ 12000) represents the thread period that interface with this actuator and Actuator(i) denotes the i$^{\text{th}}$ system actuator. For instance, when considering Actuator(0) (named *ESC right* in the UAV model), this property is satisfied with a probability in $[0.107051, 0.206887]$ with confidence 0.95.

Regarding the UAV properties evaluation, overall, forty two properties were analyzed on different categories including, reachability, safety, liveness, and deadlock freeness. Half of them evaluate the probabilities associated with the error states providing estimations with 95% of confidence. The remaining properties cover the more general characteristics where it is observed that 52.38% of these properties are totally satisfied and 47.62% of these properties may be satisfied. These results

```
1  THREAD IMPLEMENTATION t_positionEst.impl
2   PROPERTIES
3    dispatch_protocol => periodic; compute_execution_time => 9000 us .. 10000 us;
4    period => 100000 us; Priority => 3; deadline => 100000 us;
5   ANNEX BEHAVIOR_SPECIFICATION {**
6    VARIABLES
7    thTimer : clock; ttTime : clock; cf : integer;
8    STATES
9     readyState : initial state; error : complete final state; idle : state; blocked : state;
10    msgReceive : state; msgSend : state; gpsInterface : state; positionEstimation : state;
11   TRANSITIONS
12    T1 : readyState -[]-> msgReceive {cf := 0; thTimer := 0; run?; invariant!("msgreceive","thTimer<100000"); guard!("head()", "==","id")};
13    T2 : msgReceive -[errorOccurred!=1 and thTimer>=125]-> gpsInterface {rund_imuimpl!; cf :=1; thTimer:=0; invariant!("gpsInterface","thTimer<100000")};
... here goes others thread transition lines 14 to 31.
32 END t_positionEst.impl
```

Fig. 8: AADL position thread behavior.

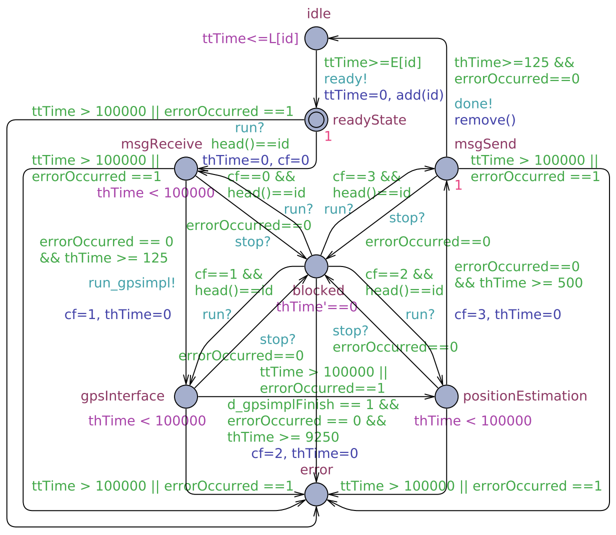

Fig. 9: Position estimation task.

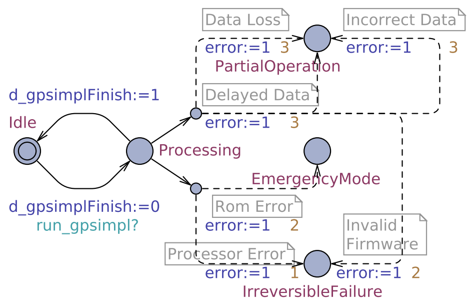

Fig. 10: GPS template.

are indeed according to our expectations since due to the nature of the specifications, these may not be satisfied due to the possibility of leading to an error state of the system.

The obtained results with the UAV model showed that the system meets its requirements. This implies that the system threads meet their deadlines, the safety properties are satisfied, and no deadlocks are founded except if an error occurs.

VI. Related Works

The present section details the works that relate to our proposal, either because they also make use of MDE premises, or because they provide means to support the evaluation of system properties using model checking.

An approach designed to the construction of on-board computer-based aerospace systems named COMPASS is presented in [19]. The authors propose a co-engineering process focused on the specification and analysis of these systems. COMPASS models are designed using the SLIM language, which is a subset of AADL that includes natively on the language some behavioral properties from the AADL BA. The proposed tool provides support for model checking, therefore SLIM model needs to be translated into a Labeled Transition System (LTS). It supports the evaluation of properties such as safety, correctness, and dependability analysis. The major weakness of this approach comes from the fact that AADL specifications are not fully supported (due the use of SLIM).

A model transformation approach to generate Timed Abstract State Machines (TASM) from AADL models is presented in [20]. TASM also uses the UPPAAL tool to make model checking. The major weakness from this approach relates to fact that no hardware elements are taken into consideration (e.g. Devices) and that there is no support for probabilistic model checking.

In [21] the authors propose a set of steps to allow performing the system specification integrated with formal verification. Therefore, AADL models are translated to an intermediary language named Fiacre. The transformation omits the hierarchical information of the AADL syntax and concentrates on the threads execution and communication. In order to support MC, the Fiacre model is then compiled into a format for the Tina tool to allow formal verification using LTL formulations. From the present work perspective, the major weakness from this approach comes from the fact that AADL EA specifications are not covered in the transformation process to Fiacre.

In [5] it is presented another approach that aims to support timed automata generation from AADL, also using the UPPAAL tool for model checking. Therefore an annex to the AADL language is proposed to support properties specification. It is observed that the authors did not detailed the transformation process, i.e., which AADL components are used for the timed automata generation. The transformation rules are also not detailed, including the fact that it is unclear whether the proposal is supported by a tool or should be manually performed.

Analyzing the related works it is observed that they consist mostly in applying MC over AADL models. However, these works provide only partial support regarding the AADL language coverage, to extract the timed automata, and require multiple transformation processes to support the properties evaluation. Some approaches include characteristics that could complement our approach, like the SLIM language [19] and its abstraction capacity for models construction without requiring the usage of AADL annexes. On the other hand, our proposed method can be used to complement approaches like [19], [20], [21], and [5], given that we allow representing the devices characteristics (using the AADL EA).

Approaches such as [20] and [21] are focused on detailing the software components of the AADL model. In [5] the authors propose a language extension to provide the automata properties representation. In our approach, the designers can represent the system threads, the set of required functions, and the possible device failures directly on the native language. This results in more complete models, which are suitable for more detailed analysis.

VII. Conclusions and Future Work

This paper presented an approach to integrate formal verification in the CPS design process. It consists of performing model transformation of AADL models to a timed automata representation that is suitable to be analyzed using model checking (MC). The transformation process is supported by a developed tool named *ECPS Verifier* and MC can be performed using the UPPAAL tool.

Comparing to the related works, it is observed the partial compliance of their transformation process in respect to the designed architectural models. Most of the time the generated models require manual intervention from the designers in order to incorporate additional features such as threads behavior and error properties. In this sense, the present proposal provides means to represent the required system characteristics to support the complete timed automata extraction and its evaluation.

An observed difficulty for using our proposal relates to formally representing the system properties to be evaluated. Currently it must be done using UPPAAL and UPPAAL-SMC syntaxes, which requires a high level of expertise. Our intention is investigating solutions to simplify the properties specification. For instance, we plan to evaluate the integration of the property patterns defined in [22].

Acknowledgment

The authors would like to thank the Brazilian research agencies CAPES and CNPq for their financial contribution to the accomplishment of this work. This work was partially supported by National Funds through FCT (Portuguese Foundation for Science and Technology) and co-financed by ERDF (European Regional Development Fund) under the Portugal2020 Program, within the CISTER Research Unit (CEC/04234); also by FCT and the EU ECSEL JU under the H2020 Framework Programme, within project ECSEL/0006/2015, JU grant nr. 692455-2 (ENABLE-S3).

References

[1] P. Derler, E. A. Lee, and A. S. Vincentelli, "Modeling cyber-physical systems," *Proceedings of the IEEE*, no. 1, pp. 13–28, Jan 2012.

[2] M. Huth and M. Ryan, *Logic in Computer Science*, 2004, vol. 2142.

[3] L. B. Becker, J. Farines, J. Bodeveix, M. Filali, and F. Vernadat, "Development process for critical embedded systems," in *WSE 2010, Gramado. Anais. Porto Alegre: SBC*, 2010, pp. 95–108.

[4] J. C. Jensen, D. H. Chang, and E. A. Lee, "A model-based design methodology for cyber-physical systems," in *IWCMC 2011*, July 2011, pp. 1666 –1671.

[5] Z. Yu, D. Yunwei, Z. Fan, and Z. Yunfeng, "Research on Modeling and Analysis of CPS," in *Autonomic and Trusted Computing: 8th International Conference, ATC 2011, Banff, Canada, September 2-4, 2011. Proceedings*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, no. 60736017, pp. 92–105.

[6] G. Behrmann, A. David, and K. G. Larsen, *Sfm-rt 2004*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2004, ch. A Tutorial on Uppaal, pp. 200–236.

[7] T. A. Henzinger, P.-h. Ho, and H. Wong-toi, "HyTech: The Next Generation*," in *Real-Time Systems Symp.*, 1995.

[8] M. Bozga, C. Daws, O. Maler, A. Olivero, S. Tripakis, and S. Yovine, "Kronos : A Model-Checking Tool for Real-Time Systems *," *Springer International Journal of Software Tools for Technology Transfer*, vol. 1, 1997.

[9] J. Bengtsson, K. G. Larsen, F. Larsson, P. Pettersson, and W. Yi, "Uppaal — a Tool Suite for Automatic Verification of Real–Time Systems," *Workshop on Verification and Control of Hybrid Systems III*, no. 1066, pp. 232–243, 1995.

[10] C. Baier and J.-P. Katoen, *Principles Of Model Checking*, 2008, vol. 950.

[11] P. H. Feiler, D. P. Gluch, and J. J. Hudak, "The architecture analysis & design language (AADL): An introduction," Software Engineering Institute, Carnegie Mellon University, Tech. Rep., 2006.

[12] SAE, *SAE Architecture Analysis and Design Language (AADL) Annex Volume 1: Annex A: Graphical AADL Notation, Annex C: AADL Meta-Model and Interchange Formats, Annex D: Language Compliance and Application Program Interface Annex E: Error Model Annex*, http://standards.sae.org/as5506/1, SAE AS5506/1, 2011.

[13] ——, *SAE Architecture Analysis and Design Language (AADL) Annex Volume 2*, http://standards.sae.org/as5506/2, SAE AS5506/2, 2011.

[14] 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, Jan. 1973.

[15] F. S. Goncalves, L. B. Becker, and G. V. Raffo, "Managing CPS complexity: Design method for Unmanned Aerial Vehicles," in *2016 1st IFAC Conference on Cyber-Physical & Human-Systems*, 2016.

[16] Novatel, *OEMStar Receiver - Firmware Reference Manual*, 6th ed., Novatel Inc., 2 2014.

[17] G. Caswell and E. Dodd, "Improving UAV Reliability," no. 301, p. 7, 2014.

[18] G. Fuggetti, A. Ghetti, and M. Zanzi, "Based on Handy FDI Current Sensor and a Fail- Safe Configuration of Control Surface Actuators," pp. 356–361, 2015.

[19] M. Bozzano, A. Cimatti, J. P. Katoen, V. Y. Nguyen, T. Noll, and M. Roveri, "The COMPASS approach: Correctness, modelling and performability of aerospace systems," *LNCS*, pp. 173–186, 2009.

[20] Z. Yan, K. Hu, J.-p. Bodeveix, L. Pi, J.-p. Talpin, and I. E. Society, "From AADL to Timed Abstract State Machine: A Certified Model Transformation," *Journal of Systems and Software*, pp. 42–68, 2014.

[21] T. Correa, L. B. Becker, J. M. Farines, J. P. Bodeveix, M. Filali, and F. Vernadat, "Supporting the design of safety critical systems using aadl," in *2010 15th IEEE International Conference on Engineering of Complex Computer Systems*, March 2010, pp. 331–336.

[22] M. Dwyer, G. S. Avrunin, and J. C. Corbett, "Patterns in property specifications for finite-state verification," *In Proceedings of the 21st International Conference on Software Engineering*, May 1999.