From AADL to Timed Automaton - A Verification Approach

Mohamed Elkamel Hamdane[1], Allaoui Chaoui[2] and Martin Strecker[3]

[1]Normal High School of Constantine, Algeria
[2]MISC Laboratory, University Of Constantine, Algeria
[3]IRIT Laboratory,University of Paul Sabatier, France

[1]hamdane.med@gmail.com , [2]allaoua.chaoui@gmail.com and
[3]martin.strecker@irit.fr

Abstract

The AADL is considered as one of the most powerful language for modeling the embedded systems. In this work, we propose an approach for the verification of the AADL architecture by using of the timed automata formalism. Indeed, the AADL architecture cannot be directly analyzed by model checking. An alternative for achieving that is to use the model driven engineering technology to extract an analysis model so that the properties can be verified using a model checker toolbox. The goal of this effort is to insure some properties of the AADL models using the Uppaal model checker. We show the application of our approach to an example.

Keywords: *AADL, Timed Automata, Transformation, Verification, Uppaa*

1. Introduction

The AADL [1] (Architecture Analysis and Design Language) language is an ADL (Architecture Description Language) specifically for real-time embedded systems. In this type of system, the temporal properties take an important place in development [2]. On the other hand, the MDE [3] Model Driven Engineering approach provided through its concepts a complete framework to monitor the design of this type of system.

The AADL language focuses on the architectural aspects: it allows the description of components and their connections, but does not deal with their behavior implementation or semantics of the data handled. This requires a step of evaluation of the possible behaviors of the system in order to ensure that the implementation of such architecture meet its specifications. Hence, the need to adopt a formal analysis techniques to verify the behavior of AADL components and their interactions.

In this work, we chose the formalism of timed automate [4] as target formalism for the verification of AADL models. This is justified by their ability to represent temporal constraints [4]. In addition, these models have a great capacity for analysis [5]. Indeed, several verification tools have been developed around this formalism as UPPAAL [6] and KRONOS [7], *etc*.

Related Work

In literature, the consistency of the AADL architecture is interested by several research works. The transformation of the AADL to colored petri net is studied in [11] this work focuses on the validation of the structure of the architecture in order to ensure integrity of the flow execution and data within AADL architecture. The translation AADL to BIP [2] allows

the simulation of AADL models and the application of verification techniques. But, absence of locality notion in BIP makes difficult compositional reasoning. An encoding of AADL in Fiacre is presented in [12] that focuses on an interpretation of the AADL specification, including the behavior annex, in the Fiacre language, which is one of the input language of the Tina [13] toolbox. A specification of the AADL model in Pola is given in [14]. This study presents the integration of a formal language for verification Pola in order to check the scheduling policy between threads in AADL Architecture. In [15] the authors attempt to check the timing constraints of the AADL architecture expressed in AADL modes. An AADL mode describing any configuration of the AADL architecture. The AADL modes are used to specify the configuration of the architecture (topology) through a graph of components and connectors. This information is necessary to determine whether the components and connectors are composed correctly.

In our work we are interested in temporal behavior of components. Indeed, the behavior of each component is defined in behavior annex. Our goal is to extract from these behaviors a formal description expressed in timed automata model. Thereafter, we use UPPAAL toolbox to verify the correctness and time property of the timed automata model. Indeed, we supervise the proposed approach through a set of related tools.

Outline

In Section 2, we give an overview of the AADL language. In the next section we gives some notes about timed automata, MDE approach and model checking technology. Section 4 gives details of the proposed approach. We gives quick overview of the Uppaal model checker in section 5. In Section 6, we demonstrate the efficiency of our system through an example. Finally we present our conclusions in Section 7.

2. AADL: Presentation and Example

AADL [1] was designed as a language to facilitate the design and analysis of complex systems, critical and real-time such as aeronautical, automotive and aerospace. It is derived from the language Meta-H developed by Honeywell and is standardized by the SAE (Society of Automotive Engineers) [1].

AADL has many advantages that justify its growing interest in the embedded industry. This language was designed to be extensible, using either properties or by defining annexes that can be extended according to the specific design. The standard AADL can be expressed in different syntaxes: text format, XML format and graphical presentation. Through its various syntaxes, AADL can be used by many tools, whether graphical or not. Finally through proposed standard precise execution semantics, it allows to define the behavior system modeled by AADL, and a semantic basis for common analysis tools.

The Figure 1 present a simple liquid heating system described in AADL language. This system has two valves V_1 and V_2, a tank, a thermostat and two level sensors: the sensor S_1 monitors the maximum level and the sensor S_2 which monitors the minimum level. The system begins with a filling phase, using the valve V_1. Once the liquid level reaches the maximum level, after an interval of [40.50] tu (time unit), a level sensor emits the event completed, the valve V_1 switches to closed position and the system starts the heating phase lasting 60 tu. Then, the controller controls the discharge of liquid in a tank by opening the valve V_2. The evacuation phase lasts between 20 and 25 tu. It ends when the level sensor S_2 detects that the tray is empty and alerts the controller through the issuance of the event empty. To intercept this event, the controller closes the valve V_2 to start a new cycle of the system.

```
.....
Process sysChauffage
Features
Pin:in event data port Behavior::integer;
End sysChauffage;
Process implementation sysChauffage.imp
Subcomponents
remplissage:thread rempli.imp;
chauffage: thread chaufg.imp;
evacuation: thread evacuer.imp;
Connections
        event data port Pin->remplissage.entree1;
End sysChauffage.imp;
Thread rempli
Features
                Entree1: in event data port Behavior::integer;
                Cal1: server subprogram remplir.imp
        { Behavior_Properties::Server_Call_Protocol=>HSER;};
                Sortie1:out event data port Behavior::integer;
End rempli;
Thread implementation rempli.imp
Connections
Con1: event data port sortie1->chauffage.entree2 ;
Properties
                Dispatch_protocol=>aperiodic;
                Period=>50ms;
        Compute_Execution_Time => 10ms .. 60ms;

Annex  behavior_An2{**
States
        S0:initial state; s1,s2:state;s3:final state;
Transitions
        s0-[entree1?x]->s1{while x<=50  do x:=x+1;cal1?};
        s1-[on (cal1.s<=50 and x=50) sortie1!(x:=0)]-s2;
        s2-[on (x=50) and Timeout 50]->s3;
connections
        con_sub:event data port cal.s->sortie1;
**};
End rempli.imp;
```

```
Thread chaufg
Features
        entree2:in event data port;
        sortie2:out event data port;
End chaufg;
Thread implementation chaufg.imp
Connections
                Con2: event data port sortie2-> evacuation.entree3;
Properties
                Dispatch_protocol=>aperiodic;
                Period=>60ms;
        Compute_Execution_Time => 10ms .. 70ms;

Annex  behavior_specification {**
State
s0:initial state;
s1:state;s2:final state;
transitions
s0-[entree2?x]->s1{if x<=60 then entree2:=x+1;};
s1-[on x<60]->s0;
s1-[on (x=60) ]->s2{x:=0;sortie2!x;};
**};
End chaufg.imp;
Thread evacuer
Features
entree3:in event data port Behavior::integer;
sortie3:out event data port Behavior::integer;
cal2: server subprogram evacu.imp
        { Behavior_Properties::Server_Call_Protocol =>HSER;};
End evacuer;
Thread implementation evacuer.imp
Connections
Con3: event data port sortie3->remplissage.entree1 ;
Annex  behavior_specification{**
States
s0 :initial state ;
s1,s2 :state ;
s3:final state;
transitions
s0-[entree3?x]->s1{if x<=25 then sortie3:=x+1};
s1-[on x<25]->s0;
s1-[on (x>=20 and x<=25)]->s2{cal2!(s1);};
s2-[on cal2.s1!=0and x<25 ]->s1;
s2-[on (cal2.s1=0 and x=25]->s3{x:=0;sortie3!x;};
connections
event data port B2.s1->sortie3 ;**};end evacuer.imp;
```

Figure 1. Example of AADL Component - Thread with Behavioral Annex

3. Preliminaries

Definition 1. Timed Automata:

The timed automaton [4] is considered as classical automata (*i.e.*, graph containing a finite set of nodes or locations and a finite set of labeled edges) equipped with clocks that evolve continuously and synchronously with time. Formally, a timed automaton is composed of six elements (L,l_0,X,T,Inv,A):

- L: is a finite set of control states or localities,
- $l_0 \in$ L: is the initial state,
- X: is a finite set of clocks,
- T $\subseteq$ L × C(X)× A × 2X × L : is a finite set of transitions ; $\boldsymbol{e}$=(l,g,a,r,l') $\in$ T represents a transition from l to l', g is the guard associated to $\boldsymbol{e}$, r is the set of clocks to be reset to 0 and a is the label of $\boldsymbol{e}$.
- Inv: L $\rightarrow$ C<(X) is a function that associates an invariant to each state control,
- A: is a finite alphabet of action.

The Figure 2 represents an example of timed automata in graphical presentation:

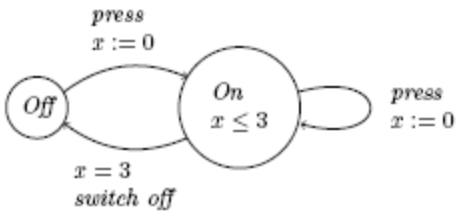

Figure 2. Example of a Timed Automata

Definition 2. Models, metamodels and transformations

The MDE [3] Models Driven Engineering community has been using the concepts of model, metamodel and transformations. A model is a representation of a system; it captures some characteristics and provides knowledge about it. The model is expressed in language called metamodel. Often, metamodels are considered as definitions of the abstract syntax of modeling languages [19].

The development process in MDE is considered as a series of transformations over models. Usually, the transformations are refinement steps over models that decrease the level of abstraction but other forms may also be found in transformation chains. Generally there are two types of transformation: model to model (Model2Model) and model to code (Model2Code). For a precise definition of these concepts based on multi-graphs the reader is referred to [20].

Definition 3. Model Checking

The Model Checking is an automated technique for verification of dynamic systems. It comes to algorithmically verify whether a given model satisfy a specification, often formulated in terms of temporal logic.

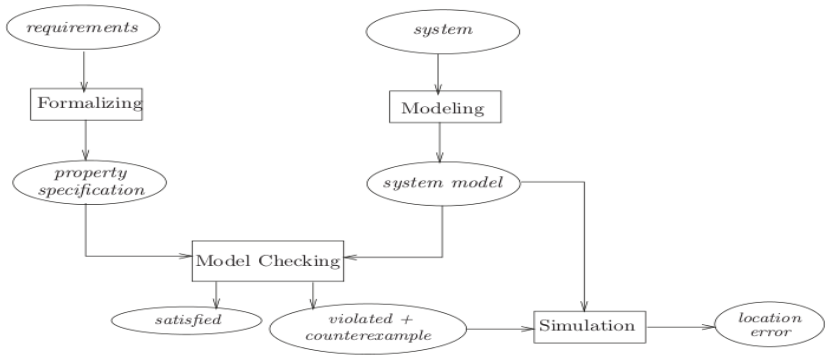

Figure 3. Schematic View of the Model Checking Approach

4. Modeling Approach

The proposed approach consists of transforming the AADL models to timed automata models which will be later an input for the Uppaal model checker. The approach we have implemented follows strictly the approach of MDE. Our approach consists of three phases. In the first phase we define for each model (source and target) theirs metamodel. In the second phase we define a transformation process in two steps:

1. The first step consists in taking an AADL models and transform it (Model2Model transformation) into a timed automata models. This formalism is considered as a formal model for describing real-time systems. They provide a formal support to their analysis. Furthermore, there are many model checker tools around this formalism.
2. The second step in this transformation process is a Model2Code transformation. It consists in taking the timed automata models produced by the previous phase and translate it into a ta-format [6] code (*i.e.*, ta-format is the input code of the Uppaal model checker). The BNF of the ta-fromat is detailed in [6].

The last phase in our approach consists to open the description generated by an Uppaal model checker and begin a verification task in order to evaluate some properties such as: deadlock, reachability, liveness, *etc*. An overview of the proposed approach is given in Figure 4:

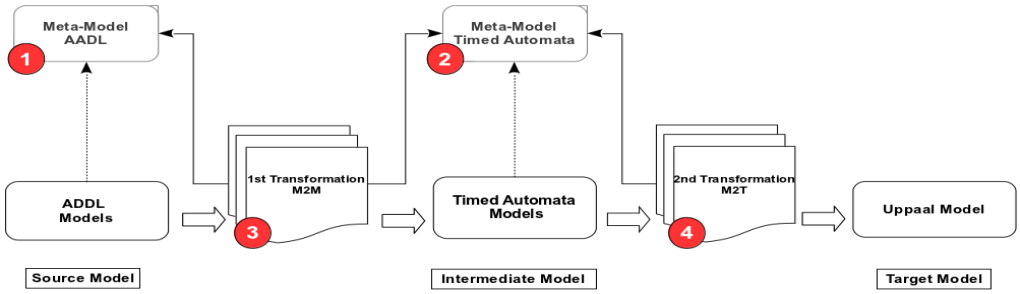

Figure 4. Overview of the Proposed Approach

4.1. The 1st step: Metamodelisation

The metamodel source and target are represent in Ecore format within EMF [8] (Eclispe Modeling Framwork).

4.1.1. Metamodel of AADL: AADL has a great capacity of expression. There are, indeed, a great number of component types which can be used to build hierarchical models. As an indication, the AADL metamodel contains 254 classes including 56 abstract classes. In order to minimize the complexity of the AADL language; the metamodel proposed (see Figure 5) respects the following hypotheses:

– The proposed metamodel supports only software components.
– Only the properties related to the behavioral aspects are considered
– We used only two types of feature: port and subprogram
– The number of data shared between thread component is equal to 1

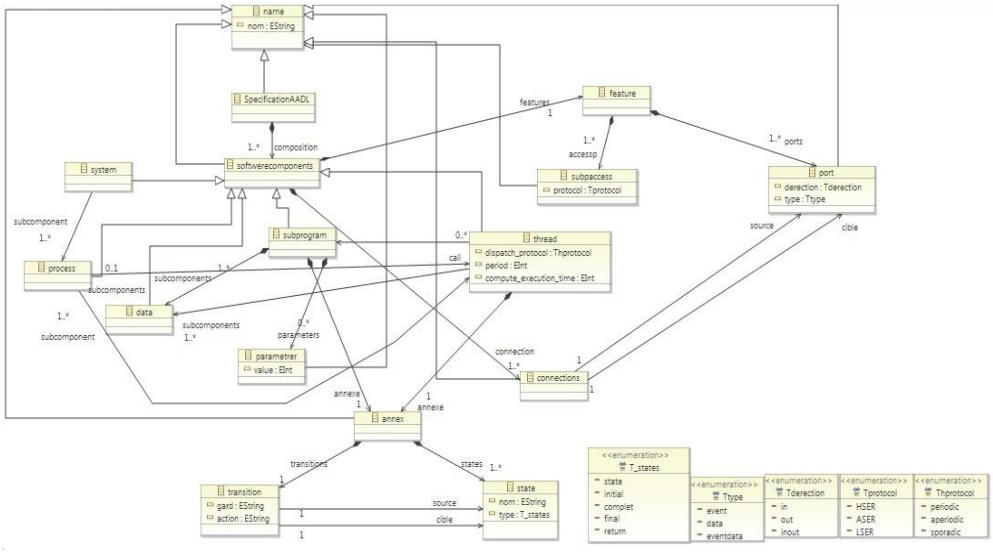

Figure 5. Subset of AADL Metamodel

4.1.2. Metamodel of Timed Automata: The timed automata is an intermediate model in our approach. Inspired from the work [16], we propose a metamodel for timed automaton. A timed automaton is composed of states, clocks and transitions. Each transition has a source state, a target state and a set of clocks to be reset to 0. The metamodel is shown in the following Figure 6:

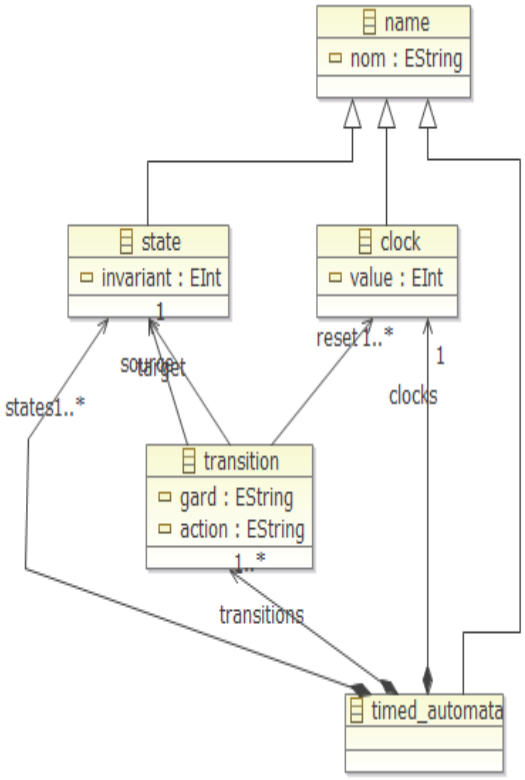

Figure 6. Timed Automata Metamodel

4.2. The 2nd step: Process Transformation

The process transformation is realized in two phases M2M and M2T:

4.2.1. M2M Phase: the aim of this phase is to build a timed automata model from the AADL model. For this, we proposed five rules:

(1) ***Rule1: Process2timedAutomat***: Transform any instance of a component process towards a timed automata such as: (a) The name of the automata receives the process name. (b) SubcComponentt reference that refers to a thread in the AADL metamodel is translated by a reference states that refers to a state in the metamodel timed automata. (d) Reference subcd in AADL metamodel is translated by a reference clocks in a clock pointing timed automata metamodel.

(2) ***Rule 2: Thread2State***: Access to thread component in AADL metamodel and transform it to the state in timed automata metamodel as: (a) The state of timed automata receives the name of thread. (b) The invariant of the state in a timed automata receives the value of the property compute-execution-time

(3) ***Rule 3: Data2clock***: The data implemented inside a process component is translates into a clock in timed automata model

(4) ***Rule 4: Transition2Transition***: Transform the connection between two threads to a transition between two states such as (a) at the behavior annex the guard in a transition translated as the guard into the timed automata (b) at the behavior annex the action between state and a final state is considered as action between two states in timed automata (c) make all clock to 0.

(5) ***Rule 5: Connection2Transition***: A source thread of connection in the AADL model becomes a source state of the transition in the timed automata model. The Target thread of connection in the AADL model becomes a target state of the transition in the timed metamodel.

We choose ATL [9, 19] (ATLAS Transformation Language) mainly for its compliance with the standard QVT (Query View Transformation) of the OMG [17]. In addition, ATL is a plug-in of the Eclipse project.

4.2.2. M2T Phase: The aim of this step is to build a text code in ta-format [6] according to timed automata generated in the first step. The generated code will be interpreted by Uppaal model checker. The next points summarize the different rules to ensure this transformation:

(1) ***Rule 1: Concerns the corp of the generated file***: retrieves the name of the time automata and organize the file generated into three parts: Global Declaration, Process Description, System Configuration

(2) ***Rule 2: Concerns the states:*** retrieves for each state the name and invariant. Put this information in Process Description Part.

(3) ***Rule 3: concerns the initial state***: retrieves the name and invariant of the initial state and put it in the Process Description Part.

(4) ***Rule 4: Concerns the transitions:*** retrieves for each transition the source state, the target state, guard, action and the reset. Put it in the Process Description Part.

(5) ***Rule 5: Concerns clocks***: retrieves all clocks from the timed automata and put it in the Global Declaration.

We choose the Xpand [10] tool to implement these rules. This tool is selected because it follows MDE approach, including increased reliability and quality of code. In addition, Xpand is a plug-in of the Eclipse project.

4.3. The 3rd Step: Verification with the Uppaal

The next step in our approach was to verify the timed automata represented in the ta-format code using model-checking technology. This textual description can be opened with the Uppaal [8] model checker in order to verify certain properties as: deadlock, safety, liveness, reachability, *etc*.

5. Uppaal Model Checker: An Overview

Uppaal [18] is a tool box for simulation and verification of timed automata networks. It consists of two main parts: a graphical user interface (GUI) and a model-checker engine [18]. Since its first release in 1995, the tool has acquired a strong maturity and has been applied in many case studies. To describe the network of timed automata in Uppaal it is possible to use a

GUI or the model can be directly written in the ta-format code. After the description of the network, we can apply a series of simulation and verification. A complete presentation of the tool can be found in [6, 18]. The following figure was produced using the version 2.04 of Uppaal:

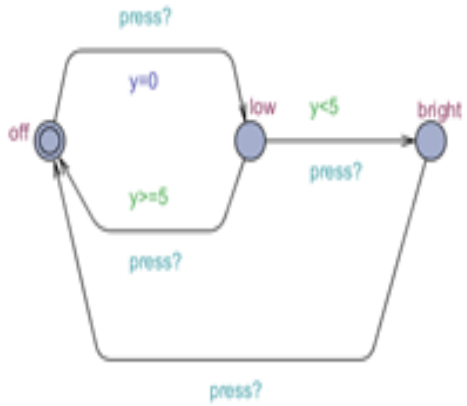

Figure 7. Timed Automata in Uppaal

The user can simulate any finite trace with the simulator, such as counter-examples produced by the verification engine. The user defines the queries expressed in temporal logic CTL (Computation Tree Logic) [6, 18] and launches the verification engine. The notation used to express the rules are:

- S.r defines that the timed automata S is in state r
- V α n (V being a variable) or h ~ n (h is a clock) with n ∈ N and α ∈ {=, >, <, ≥, ≤} a boolean combination (not, and, or, imply) of atomic propositions P. The properties are expressed by two types of formulas using the combiner AF, G and F of the CTL :
  - AGP " all reachable configuration check satisfying P " written in Uppaal by A[]P
  - EFP " it is possible to expect a configuration satisfying P " written in Uppaal by E<>P

6. Applications

We have implemented the approach of Section 4 on the example of the liquid heating system (see Section 2). The Figure 8 present the model of the liquid heating system described in EMF according to the AADL metamodel. This model has one process called “Chauffage”. This process has two features input and output, this process is implemented with three thread component: rempli, chaufg and evacuer. The behavior of each thread component is detailed in the annex behavior.

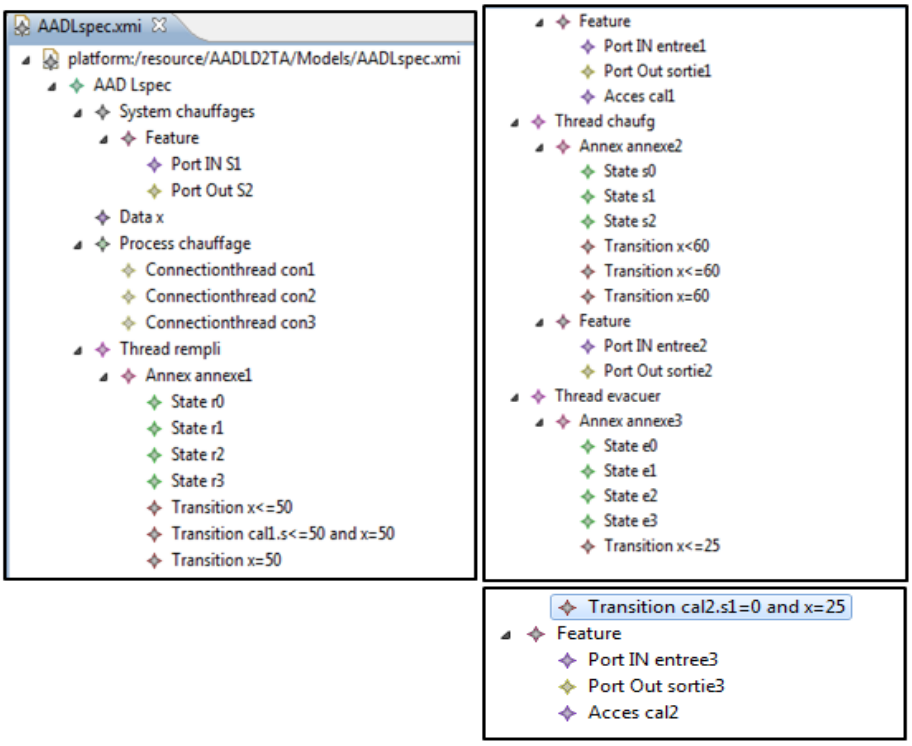

Figure 8. The AADL Model of the Liquid Heating System (Source Model)

From the model of Figure 8 we applied the first step of transformation to produce the corresponding timed automata model. The result of this transformation is described in the Figure 9.

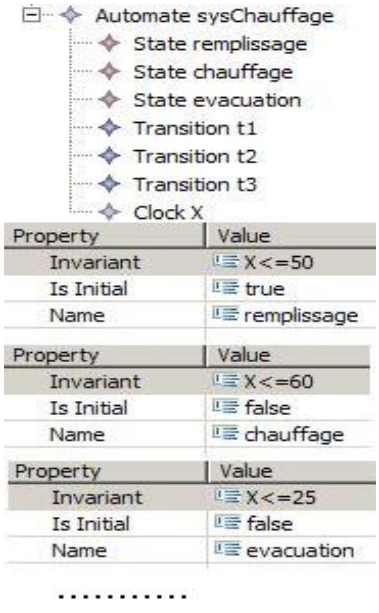

Figure 9. The Timed Automata Model of the Liquid Heating System

The next step is to generate the ta-format code according to the model of Figure 9. This second transformation is assured by the rules of the second step int the process transformation.

```
//Global Declaration
clock X;
chan next;
// Process Description
process sysChauffage {
state remplissage {X<=50},chauffage {X<=60},
evacuation {X<=25};
init remplissage;
trans
remplissage -> chauffage {
guard X>=40 and X<=50 ;
sync next!;
assign X:=0;
},
chauffage -> evacuation {
guard X==60 ;
sync next!;
assign X:=0;
},
evacuation -> remplissage {
guard X>=20 and X<=25 ;
sync next!;
assign X:=0;
};
}
process user {
state idl;
init idl;
trans idl->idl {sync next?;};
}
// System Configuration
system sysChauffage, user;
```

Figure 10. The ta-format Code of Liquid Heating System

The code we have obtained (Figure 10) represents the model timed automata of liquid heating system accepted by Uppaal model checker. The Figure 11 represents the interpretation of the previous code with the Uppaal model checker.

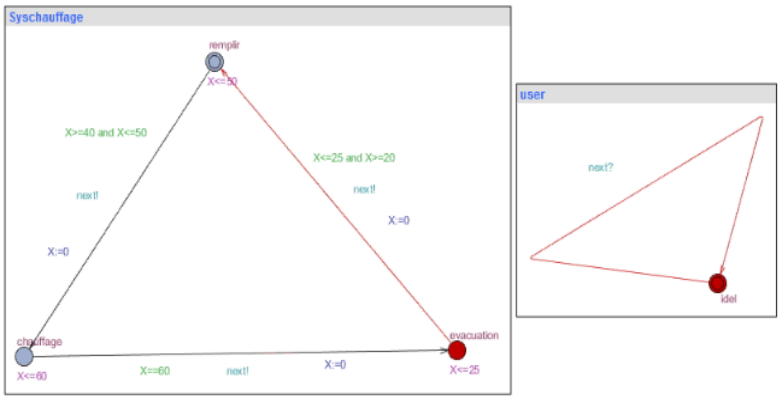

Figure 11. Interpretation of the Code ta-format with Uppaal

6.1. Verification with Uppaal Model Checker

Now the equivalent model of the source model (AADL model) is represented by the timed automata model in Uppaal model checker. The analyses of the model with Uppaal can checked properties such as reachability, deadlock, liveness. The Uppaal model checker

require that the request (the property to check) will be expressed on temporal logic CTL. The series of queries are detailed in the following:

Checking the deadlock property

- The system cannot be a deadlock situation ?
  - Query: A[] not deadlock
  - Result: Property is satisfied

Checking the reachability property

- The system can returns to the initial state (complete cycle) ?
  - ***Query*****:** A<> Syschauffage.remplir
  - ***Result*** : Property is satisfied
- Is that the system passes obligatorily by a evacuation phase ?
  - ***Query****:* A<> Syschauffage.evacuation
  - ***Result:*** Property is satisfied
- If the system terminates the heating phase can it reach the evacuation phase ?
  - ***Query:*** Syschauffage.chauffage−− >Syschauffage.evacuation
  - ***Result*****:** Property is satisfied.

Checking the liveness property

- Can I have a situation where the system is in the phase of heating but the clock did not exceed 50 ?
  - ***Query****:* Syschauffage.chauffage −− > X < 50
  - ***Result***: Property is not satisfied.
  - ***Comments***: the clock X should be superior to 50 tu
- Is that the system will complete the filling phase in 30 tu ?
  - ***Query***; Syschauffage.remplir −− > X <= 30
  - ***Result***: Property is not satisfied,
  - ***Comments***: because this phase is completed after 50 tu.
- Is that the system will complete the filling phase whose X is greater to 50 tu ?
  - ***Query***; Syschauffage.remplir −− > X >= 50
  - ***Result***: Property is satisfied

7. Conclusion and Future Work

In this paper, we have presented an approach to derive a formal descriptions from the AADL model which will be verified using Uppaal model checker. We are particularly interested in the temporal behavior of the AADL architecture. The general idea is to extract a timed automaton for capturing the behavior of AADL architecture, and then verify using the Uppaal model checker that it respects the constraints of the specification. The proposed approach strictly follows the principles of the MDE approach. So, we have defined a metamodel for AADL (source model) and another metamodel for timed automaton (target model). Then, we defined a transformation process in two steps that takes as input an AADL model and produces as output a code accepted by Uppaal model checker. We validated the presented approach through a example of liquid heating system.

As further work, this approach is going to be extended to consider other aspects of AADL as dispatch protocol in order to widen its applicability. The tool is also going to be extensively applied to case studies, particularly in the automotive applications domain. Moreover, all tools used during this work can be integrated in only one plug-in under Eclipse project.

References

[1] SAE International, "Architecture Aanalysis & Designe Language (AADL)", Standard Version 2, **(2009)**.
[2] M. Chkouri, A. Robert, M. Bozga and J. Sifaksi, "Translating AADL into BIP -application to the verification or real-time systems", Models in Software Engineering, Springer-Verlag Berlin, Heidelberg, **(2009)**.
[3] R. Lämmel, J. Saraiva and J. Visser, "Generative and Transformational Techniques in Software Engineering", Lecture Notes in Computer Science, Springer, vol. 4143, **(2006)**, pp. 36-64.
[4] R. Alur and D. L. Dill, "A Theory of Timed Automata", Theoretical Computer Science, no. 2, vol. 126, **(1994)** April, pp. 183-236.
[5] J. Bengtsson and W. Yi, "Timed Automata: Semantics, Algorithms and Tools", Lecture Notes in Computer Science, Springer-Verlag, vol. 3098, **(2004)**, pp. 87-124.
[6] K. G. Larsen, P. Pettersson and W. Yi, "Uppaal in a Nutshell", International Journal on Software Tools for Technology Transfer, **(1997)**.
[7] D. A. Olivero, S. Tripakis and S. Yovine, "The tool KRONOS, Hybrid Systems III: Verification and Control", Lecture Notes in Computer Science, Springer-Verlag, vol. 1066, **(1996)**, pp. 208-219.
[8] D. Steinberg, F. Budinsky, M. Paternostro and E. Merks, "EMF: Eclipse Modeling Framework -chapter Ecore Modeling Concepts", Second Edition, Addison-Wesley Professiona, USA, **(2008)**.
[9] F. Jouault and I. Kurtev, "On the architectural alignment of ATL and QVTs", Proceedings of the ACM Symposium on Applied Computing (SAC'06), Dijon, France, **(2006)** April 23-27.
[10] Xpand Documentation, available in http://ditec.um.es/ssdd/xpand_reference.pdf, **(2010)**.
[11] T. Vergnaud, "Modélisation des systèmes temps-réel répartis embarqués pour la génération automatique d'applications formellement verifies", Ph.D thesis, École nationale supérieure des télécommunications, **(2006)**.
[12] L. Pi, J. P. Bodeveix, M. Filali and M. K. Dianfum, "A comparative study of FIACRE and TASM to define AADL real time concepts", Proceedings of ICECC'09–14th IEEE International Conference on Engineering of Complex Computer Systems, Potsdam, Germany, **(2009)** June 2-4.
[13] B. Berthomieu, P. O. Ribet and F. Vernadat, "The tool TINA construction of abstract state spaces for Petri nets and time petri nets", J. of Production Research, **(2004)**.
[14] P. E. Hladik, F. Peres and X. Shi, "Analyse d'un modèle AADL à l'aide de Pola", Proceedings of AFADL'2010, Approches Formelles dans l'Assistance au Développement de Logiciels, Poitiers, France, **(2010)**.
[15] Y. Zhang, Y. Dong, Y. Zhang and W. Zhou, "A Study of the AADL Mode Based on Timed Automata", Proceedings of ICSESS'11, 2nd IEEE International Conference on Software Engineering and Service Science, Shanghai, China, **(2011)** July 15-17.
[16] M. E. Hamdane and A. Chaoui, "Specification and verification of timed automaton using metamodeling and graph grammars", Proceedings of ICADIWT'11, 4th IEEE International Conference on the Applications of Digital Information and Web Technologies, Wisconsin, USA, **(2011)** August 4-6.
[17] F. Jouault and I. Kurtev, "On the Architectural Alignment of ATL and QVT", Proceedings of the SAC'06 ACM Symposium on Applied Computing, France, **(2006)**.
[18] G. Behrmann, A. David, and K. G. Larsen, "A Tutorial on Uppaal", Department of Computer Science, Aalborg University, Denmark, **(2004)**.
[19] F. Jouault, F. Allilaire, J. Bézivin and I. Kurtev, "ATL: A model transformation tool", Science of Computer Programming, Elsevier, vol. 72, no. 1–2, **(2008)**, pp. 31-39.
[20] F. Jouault and J. Bézivin, "KM3: A DSL for metamodel specification", Proceedings of the FMOODS'06, 8th IFIP International Conference on Formal Methods for Open Object-Based Distributed Systems, **(2006)** June 14-16, Bologna, Italy.