A static approach to estimation of execution time of components in AADL models

A.M. Troitskiy <troitskiy@ispras.ru>
D.V. Buzdalov <buzdalov@ispras.ru>
Institute for System Programming of the Russian Academy of Sciences,
25, Alexander Solzhenitsyn st., Moscow, 109004, Russia

Abstract. During development of modern avionics systems and other mission-critical systems modelling is vitally used. Models can be used for checking and validation of developed system, including early validation. Early validation is very important because the cost of errors is raising exponentially depending on the development stage. For modelling of such systems, Architecture Analysis and Design Language (AADL) is widely used. It allows to model both architecture of a developed system and some of behavioral characteristics of its components. In the paper, the task of automated model checking for consistency of some behavioral properties is considered. In particular, we focus on the problem of estimation of working time of model components and corresponding between this time and other properties in a model. This problem is close to the worst-case execution time problem (WCET) but it has its own specific in this application. We considered a static approach allowing to work with standard specification of components behaviour in AADL-models with specialized extended finite automata. In the paper, peculiarities of used behaviour model (specialized finite automata) were considered including work with time and external events. We considered the problem of working time estimation for such models connected with non-local characteristic of this property. We propose an algorithm for time estimation for such behaviour models. This algorithm was implemented in MASIW framework, a tool for development of AADL-models.

Key words: AADL; avionics design; static analysis.

DOI: 10.15514/ISPRAS-2016-28(2)-10

For citation: Troitskiy A.M., Buzdalov D.V. A static approach to estimation of execution time of components in AADL models. *Trudy ISP RAN/Proc. ISP RAS*, vol. 28, issue 2, 2016, pp. 157-172. DOI: 10.15514/ISPRAS-2016-28(2)-10

1. Introduction

Modern avionics is responsible for control of almost all aspects of aircraft operation. As a result, the complexity of such systems is really high. Thus making sure that developed system is correct is a challenging task.

Nowadays problems and their solution bring additional complexity to avionics systems. To satisfy models requirements for weight and power consumption,

integrated modular avionics (IMA [1]) approach is used. It means that several resources (e.g. universal processor modules and network) are shared between several pieces of software. The approach leads to appearing of step of the integration of the whole system, i.e. deployment of software on different hardware, network configuration and etc.

This approach solves weight and power consumption problems, but leads to potential problems of interfering of applications. It means that the whole system correctness must be checked and this problem is not solvable by checking of correctness of each part of the system.

The model-driven approach of development allows to manage with the complexity of a system being developed. In particular, models are needed to perform different kinds of analysis of the modelled system though analysis of appropriate models. Such analyses are intended to be performed on different stages of development, in particular, to eliminate errors at early steps of development.

One kind of checks that are needed to be performed is check of *timing properties* of software components.

In particular, during design and deployment stages, each particular application is bound to a processor module. Appropriate timing properties are assigned to them, for example

- *dispatch protocol*, i.e. whether an application is fired periodically, eventually (sporadically) or both;
- *period* of execution for periodic applications;
- *compute deadline*, i.e. time interval in which an application has to finish its work after it was given an ability to execute;
- *recover deadline*, i.e. time interval in which an application has to recover from recoverable errors;
- *process time*, i.e. the time between sending a processed output data after getting some input data;
- *output rate*, i.e. rate at which an application has to produce its output, when it is periodic;
- *output jitter*, i.e. maximum deviation of time for periodic output and etc.

Being assigned to some particular application, these properties can be used in schedulability analysis, data flow timing analysis, worst case execution time (WCET) analysis and etc. Some desired or expected values can appear before implementation of particular software.

During the system development, models of it are refined. In particular, for software some behaviour specifications can appear. Such behaviour specifications can be purely functional (i.e. containing only information about which outputs will be produced in particular inputs at the given state).

Also such specifications can contain how much time will be consumed in this or that situation. The addition of this information can lead to inconsistency in the model, because some assumptions about timing properties of software can already exist in the model and these assumptions can contradict with behavior specification.

To check the consistency of a model, it is important to estimate timing properties of particular behaviour specifications.

Compute deadline consistency example

Consider a periodic software component with some particular *period* set in the model. Consider also that this component has *compute deadline* property bounds set to a range $p$from $p_1$to $p_2$ms.

This property can be used in the schedule building: e.g. a time frame of $p_2$ms can be reserved each period to ensure this software component has enough time to compute. This can be done on early stages of system development when no particular behavior is known yet.

Consider the case when after development this software component is refined: now its behaviour is specified with automaton with transitions containing how much time is consumed by computations assigned to them. We can estimate general time consuming of an application each period as a range $h$from $h_1$to $h_2$ms.

After getting estimations $h$we can compare it with bounds $p$from the model and there are several decisions we can take:

- when $h = p$, behavior corresponds to property and the model is consistent;
- when $h \not\subset p$, the model can be inconsistent because real execution time may miss the bounds;
- when $h \subset p$, $p \neq h$, the behaviour specification corresponds to the property; also, we can say that the property in the model can be refined to a more precise value;
- when $p \cap h = \emptyset$, the model is inconsistent.

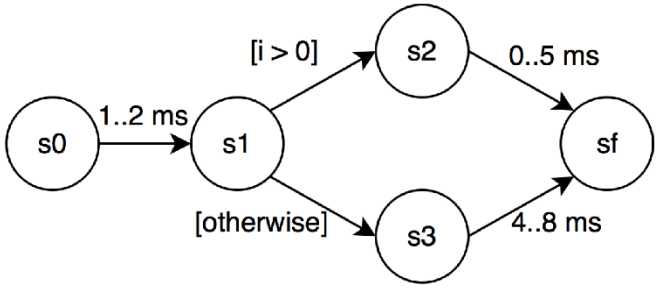

Fig. 1. Example of behavior specification

Example of consistent case

Consider an example when the model has bounds for *compute deadline* property set to be from 3 to 10 ms. Consider also that this application has behavior specification

with automaton shown on the *fig.1*. Each period this application begins in state s_0 and finishes in s_f.

In this example we can estimate execution time of the application to be between 5 and 10 ms. This value is consistent with property set in the model.

There is another case when such estimations are useful. Consider a situation when some software component in the model did not have any timing properties set. Consider then, that later it was refined and some behavior specification has appeared for it. The model still needs to be checked for schedulability and other timing-aware properties. So, we need to derive these timing properties for a component with some behaviour specification. Again, we run into an issue of estimation of timing properties having a particular behavior specification.

So, generally we can resume that there is an important issue of estimation of timing properties in responsible systems' models with behavior specifications.

2. AADL and BA

We use AADL (Architecture Analysis and Design Language, [2]) as a modelling language. It allows to describe both physical and logical parts of the modelled system, connections between components and bindings between layers of the system. AADL has a mechanism of the language extending though special language annexes and it has a number of standard annexes.

One of such extensions is called Behavior Model Annex [3] (BA). It allows to specify behavior of AADL-components using extended time-aware finite-state machine.

Behaviors are set to components of a modelled system. The basic elements used in BA behavior specifications are

- automaton states change;
- internal computations;
- accessing and assigning to internal or external variables (data components);
- interaction with the outer world using input/output ports; depending of behavior, input ports can be managed both by pulling data and by waiting for data to come;
- handling *dispatch* events, i.e. a situation when software component is allowed to perform its execution (*e.g., an operating system signals a thread to start*).

Behavior Annex automaton must contain a single initial state. When the automaton goes out from the initial state, its internal variables are being initialized. The automaton can contain several final states, in these states automaton can stop its execution.

Each state of the automaton belongs to one of the classes of *complete states* or *execution states*.

Transitions from execution states occur immediately after automaton comes to such state. In complete states automaton waits for external events (data for input ports or dispatch event). Transitions going out of complete states are fired as soon as corresponding event happens.

In BA each state transition is assigned with a list of actions which is run when automaton performs this transition.

There are actions that appear in the list of actions in BA behavior specification:

- actions with ports: reading, writing, getting of messages count in ports;
- actions with local and accessible external variables: reading and assignment;
- locking on resources: getting and releasing;
- action for modelling of time consumption $computation(t_{min}..t_{max})$;
- *stop* action for automaton interruption;
- composite actions (loops, conditionals);
- computation of arithmetical expressions.

3. Problem

We focus on AADL models with behavior specifications set using Behavior Model Annex language.

We consider a BA behavior specification of a single component in a model. Also, we consider two states s_{start} and s_{end} of the automaton are given.

We want to estimate the maximum and minimum model time the BA automaton will consume to go out from state s_{start} and to come to s_{end}.

4. Solution

Automaton can reach a given state starting from another given state in several ways depending on variables state, external events and nondeterminism. We will call an interleaving sequence of states and transitions as a *path* in automaton.

Thus we divide the original problem to considering a single path in automaton and then considering the automaton itself as a source of paths.

4.1. Estimation for a path

First, let us look at a finite path starting and ending at given states s_{start} and s_{end}, and going through states $s_1, s_2, \ldots, s_n$, which could be equal to each other and to states s_{start} and s_{end}. We would designate it as $s_{start} \to s_1 \to s_2 \to \ldots \to s_n \to s_{end}$. The question is how long does it take to go along this path out from s_{start} to s_{end}.

Some of states in the path may be complete. An automaton is waiting for external events in these states while going through them. It is a hard task to estimate how much

time would it take because it is not a local property, i.e. it depends on other components in the model.

Execution states do not consume any time by definition, thus there is no such problem for them.

Also, in BA actions assigned to transitions can take some time (*e.g. computation action takes time, which is specified with its argument; input/output operations may take time too*). Time taken by composite actions (loops and conditionals) depend on very actions inside them and external conditions (state of variables and ports). Having dependency on external conditions, estimation of time consumption by conditionals it a tricky task (undecidable in the general case).

Thus, task of estimation of time, taking by execution of a finite path, can be split into two tasks: time estimation for each complete state in the path and for each list of actions assigned to a transition in the path.

4.2. Estimations for an automaton

The whole automaton containing both execution and complete states is a challenging object. Let us at first consider simpler kind of automata containing only execution states and then to consider the general case.

4.2.1 Automata with execution states only

In this case, automaton is not waiting for external events and goes through states right away. We can represent such automaton as a weighted graph. Vertexes of the graph are states of the automaton, and edges of the graph are transitions of the automaton. Weight of each edge is time estimation for the actions of corresponding transition.

We can use all known algorithms for finding minimum and maximum times (*e.g. for finding minimum time we can use Dijkstra's algorithm [4]*).

However, when the graph is cyclic these estimations can be inaccurate. *For example, we have a loop of the automaton, which is executed exactly 50 times. If this fact is not used, estimation of the time consumption of this loop may be too imprecise, up to* $+\infty$ *for the higher bound and to 0 for the lower bound. Considering information of the number of loop iterations, we can estimate the time to be* $50t_{body}$ *where* t_{body} *is an estimation of the time consuming by the loop body, or even more precise if* t_{body} *depends on the loop iteration number in a known way.*

Despite inaccuracy in some cases, time estimation for this kind of automaton is a pretty studied problem.

4.2.2 Automata with complete states too

Approaches with simple weighted graphs with weights only on edges do not model the fact that automaton can wait some time in a complete state during its execution. But we work with automata having complete states. Thus, we need to manage with it somehow while estimating automata execution time.

It seems that this problem can be reduced to the previous one, e.g. though replacing a single complete state with two connected execution states with a transition consuming the same time as automaton waits in this complete state.

But what we realized trying to implement such approach is that time of waiting in a complete state is not local and cannot be represented by some constant. This time actually depends both on the way this state was reached and on how regular external events occur. So, automata with complete states need special treatment, one variant of which will be discussed below.

4.2.3 Solution structure

So, to solve the original task we have divided the original problem to the following subtasks:

- estimation of time consumption of paths in automaton:
  - estimation of execution time for transitions;
  - estimation of time of waiting in complete states;
- estimation of time consumption by automaton itself:
  - in a particular case, when the automaton contains only execution states;
  - in the general case, when automata with both complete and execution states are considered.

The rest of the paper follows this division.

5. Estimation of time for paths

5.1. Estimation of time for transitions

Let us estimate how much time can take different Behavior Annex actions. At first, look at simple actions.

The action *computation* has a time as an argument, which is the execution time of this action.

Also, the action *get resource* can take some time, because at the moment when this action is executed, needed resource can be used by some other component. And so it will be necessary to wait for some time until the resource can be used. We will estimate this time from 0 to $+\infty$.

If action *stop* occurs at some point, then the execution of automaton became interrupted and it does not go to the next state. The action does not take time. However, since we are interested in the time between the states of the automaton, it is convenient to assume that the time of this action is $+\infty$. Indeed, if the transition from s to q with action *stop* exists, it means that automaton will not ever be in state $q$after this transition.

Now let us consider composite actions. Loops which contains the actions occupying some time, we will estimate with time from 0 to $+\infty$. Making this estimation to be

more accurate is possible but it is not considered in this paper. Other loops do not take any time.

We will estimate conditional constructs in the following way. Time of actions in if-block is from t_1 to t_2, time of actions in else-block is from τ_1 to τ_2 (if there is no else-block $\tau_1 = \tau_2 = 0$). Then the estimation is the time range from $\min(t_1, \tau_1)$ to $\max(t_2, \tau_2)$.

In this way, estimations for transitions of the automaton can be performed. Now let us estimate time, that automaton is waiting in complete states.

5.2. Estimation of time for complete states

Behavior Annex allows to handle two types of external events: receiving a message to input port and a dispatch signal.

At first, look at the first type of events. Since the expectation of the receiving message can take arbitrarily much time, we will estimate this time with 0 to $+\infty$. So, this is the estimations of time of waiting in the complete states for the external event of the first type.

Estimations of time waiting for events of the second type can be performed in same way. But the estimations can be more accurate when the component is a thread. This is due to the fact, that AADL allows to set properties for the thread, which determined how often dispatch signal arrives to the thread (*such properties are Dispatch Protocol and Period*).

These properties determine the time between neighboring complete states in automaton. Consider any path in an automaton, which starts and ends in complete states, all other states are execution states, and the transition from the first complete state is the transition of the second type. Above AADL-properties can determine the execution time of this path from going out from the first complete state to going out from the second complete state. This time is determined by time range with possibly infinite bounds.

In this way, when automaton comes to complete state, the waiting time in this state is determined by the time elapsed from going out from the previous complete state and by AADL-properties.

6. Estimation of time for the whole automaton

6.1. Particular case, execution states only

6.1.1 Problem

The weighted oriented graph $G = \{V, E\}$ and two vertices s_{start}, s_{end} are given. The weights of the edges are determined by the function $w: E \to \mathbb{R}^2$.

Weight of each edge is a range of two real numbers $[r_1, r_2]; r_2 \geq r_1$, where r_1 is the lower bound, r_2 is the upper bound of the range. Weights are partially ordered in the following way:

$$[r_1, r_2] < [q_1, q_2] \Leftrightarrow r_2 < q_1.$$

Also, the addition function for weights is determined:

$$[r_1, r_2] + [q_1, q_2] = [r_1 + q_1, r_2 + q_2].$$

The task is to find the maximal and minimal weight of paths from s_{start} to s_{end}, where weight of a path is a sum of weights of path's $s_{start} \to \cdots \to s_{end}$ transitions counted with multiplicity.

For example, we will consider the graph on the fig. 2 and vertices s_0 and s_6 as s_{start} and s_{end} respectively.

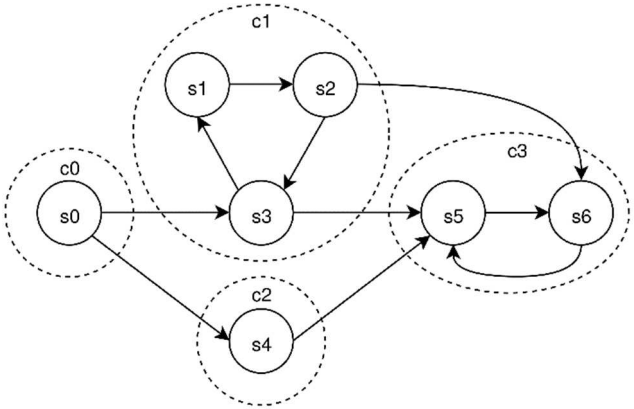

Fig. 2. Graph G and strongly connected components

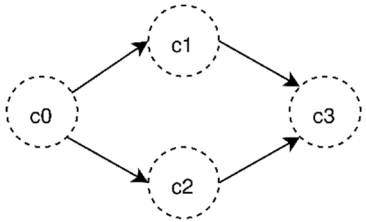

Fig. 3. Graph E

6.1.2 Algorithm

1) We find strongly connected components (SCC) in graph G with Tarjan's algorithm [5]. *Strongly connected components of the graph G are highlighted by a dotted line on fig. 2.*
2) We build acyclic graph E from strongly connected components of the graph G (*fig. 3*).
3) Let vertices s_{start} and s_{end} belong to strongly connected components c_{start} and c_{end} respectively. Then we find all paths in acyclic graph E (we call them SCC-paths) from c_{start} to c_{end}. *In the example, all paths from c_0 to c_3 are $c_0 \to c_1 \to c_3$ and $c_0 \to c_1 \to c_3$.*
4) For each SCC-path $c_{start} \to c_1 \to \ldots \to c_{n-1} \to c_{end}$ we pick vertices from each SCC and consider the following path through them: $(s_{start} \Rightarrow s_0^{out}) \to (s_1^{in} \Rightarrow s_1^{out}) \to \cdots \to (s_{n-1}^{in} \Rightarrow s_{n-1}^{out}) \to (s_n^{in} \Rightarrow s_{end})$, where $s_{start} \in c_0$, $s_{end} \in c_n$, $s_i^{in}, s_i^{out} \in c_i$, $i = 1, 2, \ldots, n$, and edges $(s_j^{out} \to s_{j+1}^{in}) \in E$, $j = 1, 2, \ldots, n-1$. We will designate such paths as p_{picked}. Designation $s_i^{in} \Rightarrow s_j^{out}$ represents an automaton path from state s_i to state s_j inside a single SCC-component. Vertices s_i^{in} and s_i^{out} can be the same. *On the fig. 4 all paths are presented.* Notice that number of such paths is finite because each SCC-path is finite.
5) Let us find the weight of each path p_{picked}. Weight of each transition $s_i^{out} \to s_j^{in}$ is equal to weight of edge (s_i^{out}, s_j^{in}) of graph G. To estimate weight of transitions $s_i^{in} \Rightarrow s_i^{out}$, $i = 1..n-1$, we consider two cases.
   Case 1: c_i is acyclic (thus containing a single vertex), then weight of the transition $s_i^{in} \Rightarrow s_i^{out}$ is 0.
   Case 2: c_i is cyclic, then upper bound of weight of the transition $s_i^{in} \Rightarrow s_i^{out}$ is positive infinity, and the lower bound is calculated using Dijkstra's algorithm [4].
6) For possibly infinite set of paths between s_{start} and s_{end} we have considered finite set of p_{picked} paths. We calculated weight of each p_{picked} path, got a finite set of weights. Thus, we can pick maximal and minimal ones.

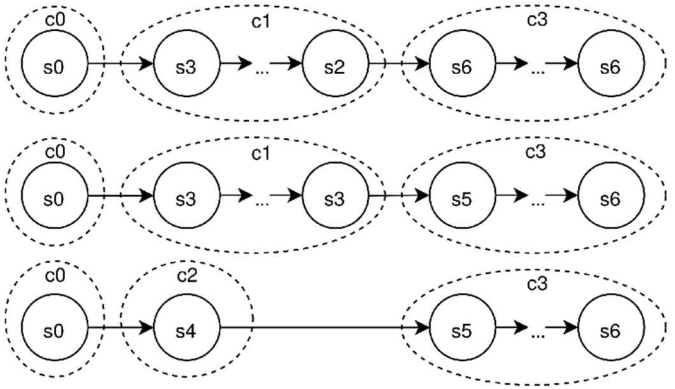

Fig. 4. Paths in graph G from s0 to s6

6.2. General case, both execution and complete states

6.2.1 Problem

The Behavior Annex automaton and two states of the automaton are given. The problem is to find estimation of the execution time of the automaton between leaving the state s_{start} and entering the state s_{end}.

We designate the set of states of the automaton as S. The set of execution states of the automaton is $Exec \subset S$, the set of complete states of the automaton is $Comp \subset S$.

For example, let us consider the automaton on fig. 5. Complete states are marked by white color, execute states are gray. The goal is to find time between state e2 and state c2.

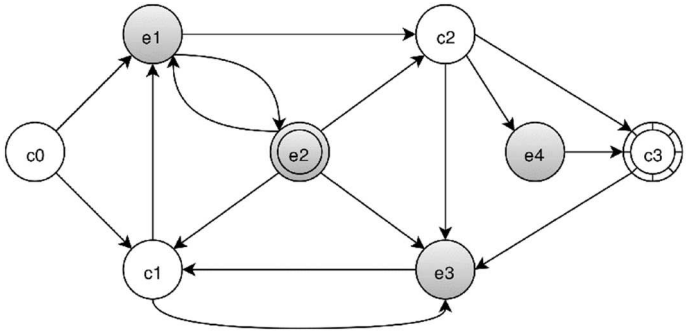

Fig. 5. Graph with complete states and execution states

6.2.2 Solution idea

Two different states types are determined in Behavior Annex. So we consider two different graphs.

We consider graph of the complete states and the graph of the execution states separately. Then if we need to find time between exit from one complete state to exit from other complete state, we use graph of complete states. In other cases we use the graph of execution states.

6.2.3 Algorithm

At first, we introduce few functions.

Function $PREV: S \to Comp$ computes all *previous complete states* for a state of the automaton, i.e. those complete states starting with which it is possible to reach the state through only execution states. More formally, $\forall s \in S\ \forall c \in Comp$: $c \in PREV(s) \Leftrightarrow \exists (c \dashrightarrow s)$, where $c \dashrightarrow s$ means $(c \to e_1 \to e_2 \to \cdots \to e_n \to s)$, with $n \geq 0,\ e_1, e_2 \ldots e_n \in Exec$.

Function $NEXT: S \to Comp$ computes all possible *next complete states* for a state of the automaton, i.e. those complete states, which can be reached from the state through only execution states. More formally, $\forall s \in S\ \forall c \in Comp: c \in NEXT(s) \Leftrightarrow \exists (s \dashrightarrow c)$. It is easy to see that $\forall c_1, c_2 \in Comp: c_1 \in PREV(c_2) \Leftrightarrow c_2 \in NEXT(c_1)$.

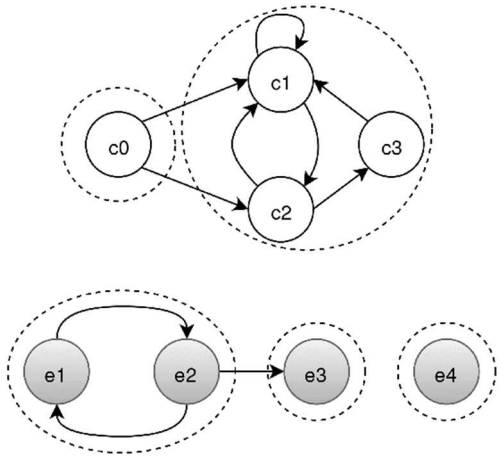

Fig. 6. Graph G_e and graph G_c

6.2.3.1 General scheme

We have two states $s_{start}, s_{end} \in S$. The aim is to find the minimum and the maximum possible time between leaving the state s_{start} and entering the state s_{end}. We will do this by estimation of time for each path $s_{start} \to \cdots \to s_{end}$. The problem is that execution time of the path depends on complete states before state s_{start}, if s_{start} is execution state.

We will consider two cases: when s_{start} is complete state, and when s_{start} is execution state.

When s_{start} is complete state, each path $s_{start} \to \cdots \to s_{end}$ can be divided into smaller paths: $s_{start} \to \cdots \to c_{in}$ and $c_{in} \dashrightarrow s_{end}$, where $c_{in} \in PREV(s_{end})$. For each $c_{in} \in PREV(s_{end})$ time of the path $s_{start} \to \cdots \to c_{in} \dashrightarrow s_{end}$ is $T(s_{start} \to c_{in}) + t(c_{in} \dashrightarrow s_{end})$, where $T(s_{start} \to \cdots \to c_{in})$ is time between leaving s_{start} and leaving c_{in}, and time $t(c_{in} \dashrightarrow s_{end})$ is time between leaving c_{in} and entering s_{end}. Notice that times T and t can be different for the same path, when the last state of the path is complete state. The ways of estimation of time $T(c_i \dashrightarrow c_j)$ were described in section 5.2.

When s_{start} is execution state, each path $s_{start} \to s_{end}$ is a part of path like $c_{out} \dashrightarrow s_{start} \dashrightarrow c_{med} \to \cdots \to c_{in} \dashrightarrow s_{end}$ where $c_{med} \in NEXT(s_{start})$, $c_{out} \in PREV(s_{start})$, $c_{in} \in PREV(s_{end})$. Time of the path $s_{start} \to \cdots \to s_{end}$ can

be computed as $T(c_{out} \dashrightarrow c_{med}) - t(c_{out} \dashrightarrow s_{start}) + T(c_{med} \to \dots \to c_{in}) + t(c_{in} \dashrightarrow s_{end})$.

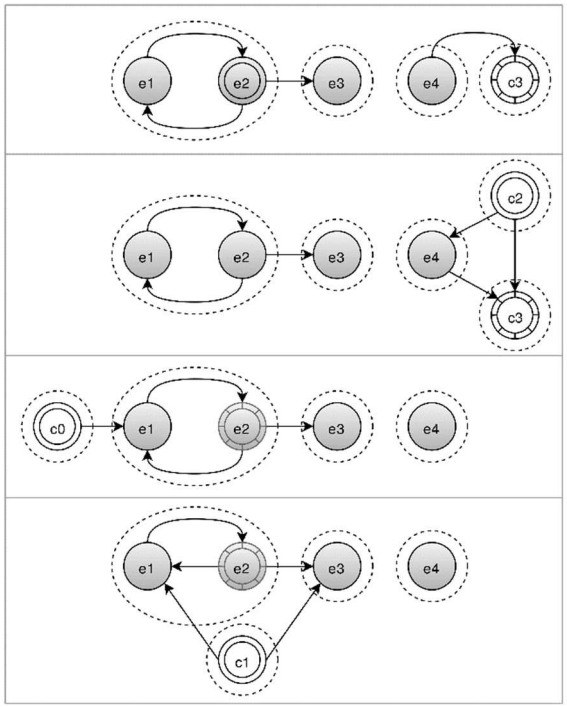

Fig. 7. Usage of graph G_e: graphs $G_e'(e_2, c_3), G_e'(c_2, c_3), G_e'(c_0, e_2), G_e'(c_1, e_2)$.

6.2.3.2 Calculation of T

Let us focus on the function T. Value of T is described in section 5.2 for paths $c_i \dashrightarrow c_j$, where $c_i, c_j \in Comp$. To find time T for arbitrary paths $(c_i \to \dots \to c_j)$ we build weighted oriented graph G_c. The vertices of the graph G_c are complete states of the automaton. We build edge (c_i, c_j), if a path $c_i \dashrightarrow c_j$ exists in the automaton. Weights of edges are determined with AADL-properties of the component as described in section 5.2, i.e. weight of an edge (c_i, c_j) equals to $T(c_i \dashrightarrow c_j)$. *Graph G_c for the considered example is presented on fig. 6.* To find time $T(c_i \to \dots \to c_j)$ we execute the algorithm described in section 6.1 on graph G_c.

6.2.3.3 Calculation of t

To find time $t(s_1 \dashrightarrow s_2)$ we build weighted oriented graph G_e. The vertices of the graph G_e are all execution states of the automaton. For each transition $e_1 \to e_2$ of the

automaton we build edge (e_1, e_2) in graph G_e. The weight of this edge is time estimation for transition's actions (see section 5.1). Graph G_e can be not connected. *Graph G_e is presented on the top of fig. 6.*

With graph G_e we can estimate time $t(s_1 \dashrightarrow s_2)$. To do this we build new graph $G_e'(s_1, s_2)$. Vertices set of graph $G_e'(s_1, s_2)$ is union of states set of G_e and $\{s_1, s_2\}$. It contains all edges from G_e. Additionally, it contains all edges, which are corresponding to outgoing transitions of automaton from state s_1 to vertices from $G_e'(s_1, s_2)$ and incoming transitions from vertices of $G_e'(s_1, s_2)$ to s_2. To find $t(s_1 \dashrightarrow s_2)$ we execute the algorithm from section 6.1 on graph $G_e'(s_1, s_2)$.

On the second line of fig. 7 the graph G_e for calculating the time between exit from complete state c_2 to enter to complete state c_3 is presented.

6.2.3.4 Calculation of the result

For each path $s_{start} \to \dots \to s_{end}$ we calculate time estimation. The result of the algorithm is the smallest time range, that contains all these time ranges.

7. Related works

One close problem to the problems, considered in this paper, is WCET problem. This problem is well-known, and a lot of algorithms solving WCET exist. But these algorithms cannot be applied to our problem directly, due to considered specific object class, defined by Behavior Annex language. As Behavior Annex describes behavior based on timed automata, consider WCET algorithms working on timed automata.

The WCET problem for timed automata was considered in the paper [6]. This paper has a description of the algorithm using the difference-bound matrix data structure to represent zones (heuristic). This algorithm can be applied in the particular case, which was described in section 6.1.

The main specific construct in Behavior Annex is complete states. In the particular case we consider automata with only execution states. These automata are very similar to timed automata from the paper [6]. It means that algorithms from the paper can be applied to the particular case. We are thinking about applying it, but currently we have chosen simpler algorithm.

But to use it in the general case from 6.2, it should be adapted. We have decided that the adaptation of the algorithm would be harder, than to develop the new algorithm applied to a needed object class.

8. Conclusion

In this paper, the development of mission-critical systems is considered. In this context, we have considered the task of correct integration of the whole system. System modelling with language AADL and analysis of models are using to solve the task.

The problem is that a component of an AADL model can have behavioral properties set. At the same time the behavior of the component can be set with Behavior Model Annex. That can lead to inconsistency of the model. So, we considered a task of automated analysis of behaviors in AADL-models.

In this paper, one static approach for analysis of timing properties is proposed. An algorithm for finding of execution time estimation of behaviour of AADL-components was offered and described in the paper. This algorithm was implemented in MASIW, a framework for development and analysis of AADL models [7].

Characteristics of behaviors, acquired using proposed algorithm can be used for checking of model consistency and for model refinement, when AADL-properties are not set.

References

[1]. B. C. Watkins, “Transitioning from federated avionics architecture to Integrated Modular Avionics”, AIAA 26th Digital Avionics Systems Conference, 2007.

[2]. Architecture Analysis & Design Language (AADL), SAE International standard AS5506B, SAE International, 2012, http://standards.sae.org/as5506b/.

[3]. Architecture Analysis & Design Language (AADL), Annex Volume 2, Behavior Model Annex, SAE International, 2011, http://standards.sae.org/as5506/2/.

[4]. E.W. Dijkstra, “A note on two problems in connexion with graphs”, Numerische Mathematik, 1959.

[5]. R.E. Tarjan, “Depth-first search and linear graph algorithms”, SIAM Journal on Computing, 1972.

[6]. O. I. Al-Bataineh, “Verifying worst-case execution time of timed automata models with cyclic behaviour”. Ph. D. dissertation, School of Computer Science & Software Engineering, 2015.

[7]. D. Buzdalov, S. Zelenov, E. Kornykhin, A. Petrenko, A. Strakh, A. Ugnenko, and A. Khoroshilov, “Tools for system design of integrated modular avioics”. *Trudy ISP RAN/Proc. ISP RAS*, vol. 26, issue 1, 2014, pp. 201-230 (in Russian). DOI: 10.15514/ISPRAS-2014-26(1)-6

Способ статической оценки времени работы компонентов AADL-моделей

А.М. Троицкий <troitskiy@ispras.ru>
Д.В. Буздалов <buzdalov@ispras.ru>
Институт системного программирования РАН,
109004, Россия, г. Москва, ул. А. Солженицына, д. 25.

Аннотация. При проектировании современных систем авионики, а также других ответственных систем, неотъемлемой частью разработки является моделирование этих систем. Модели могут использоваться для проверок и валидации системы, в том числе на ранних этапах разработки. Ранняя валидация важна из-за того, что стоимость исправления ошибок растёт экспоненциально от времени внесения этой ошибки. Для

моделирования такого рода систем широко используется язык моделирования AADL, позволяющий моделировать как архитектуру разрабатываемых систем, так и некоторые поведенческие характеристики компонентов модели. В статье рассматривается задача автоматизированной проверки модели на консистентность некоторых поведенческих свойств. В частности, рассматривается проблема оценки времени работы компонентов моделей и соответствия этого времени другим свойствам в модели. Эта проблема близка к проблеме худшего времени выполнения (WCET), но имеет свою специфику в данном приложении. Рассмотрен статический подход, работающий со стандартной спецификацией поведения компонентов AADL-моделей специализированными расширенными конечными автоматами. В статье были рассмотрены особенности используемой модели поведения (специализированных конечных автоматов), в частности, за счёт работы автомата со временем и внешними событиями. Были рассмотрены проблемы оценки времени работы таких моделей поведения, связанные с нелокальностью этой характеристики в ряде случаев. Был рассмотрен важный частный случай, а также общий случай этой проблемы. В статье предлагается алгоритм, позволяющий оценить время работы таких моделей поведения в этих случаях. Данные алгоритм реализован и используется в среде разработки AADL-моделей АРМ СИ (MASIW).

Ключевые слова: AADL; авионика; статический анализ.

DOI: 10.15514/ISPRAS-2016-28(2)-10

Для цитирования: Троицкий А.М., Буздалов Д.В. Способ статической оценки времени работы компонентов AADL-моделей. Труды ИСП РАН, том 28, вып. 2, 2016 г., стр. 157-172 (на английском). DOI: 10.15514/ISPRAS-2016-28(2)-10

Список литературы

[1]. B. C. Watkins, “Transitioning from federated avionics architecture to Integrated Modular Avionics”, AIAA 26th Digital Avionics Systems Conference, 2007.

[2]. Architecture Analysis & Design Language (AADL), SAE International standard AS5506B, SAE International, 2012, http://standards.sae.org/as5506b/.

[3]. Architecture Analysis & Design Language (AADL), Annex Volume 2, Behavior Model Annex, SAE International, 2011, http://standards.sae.org/as5506/2/.

[4]. E.W. Dijkstra, “A note on two problems in connexion with graphs”, Numerische Mathematik, 1959.

[5]. R.E. Tarjan, “Depth-first search and linear graph algorithms”, SIAM Journal on Computing, 1972.

[6]. O. I. Al-Bataineh, “Verifying worst-case execution time of timed automata models with cyclic behaviour”. Ph. D. dissertation, School of Computer Science & Software Engineering, 2015.

[7]. Д.В. Буздалов, С.В. Зеленов, Е.В. Корныхин, А.К. Петренко, А.В. Страх, А.А. Угненко, А.В. Хорошилов, “Инструментальные средства проектирования систем интегрированной модульной авионики”, Труды ИСП РАН, том 26, выпуск 1, 2014 г., стр. 201-230. DOI: 10.15514/ISPRAS-2014-26(1)-6