Visualization of Behavioral Aspects of AADL-Models

Sergey Zelenov, Denis Buzdalov
Software Engineering Department, Institute for System Programming of Russian Academy of Sciences, Moscow, Russia
e-mail: {zelenov, buzdalov}@ispras.ru

ABSTRACT

In the paper, we describe a metamodel to represent dynamics of AADL-models. The metamodel allows to visualize step-by-step a process of models' behavioral aspects analysis. In particular, it essentially helps to debug behavioral specifications. In the paper, we consider examples of the metamodel usage to represent dynamics for the following two aspects of AADL-models:

1. simulation of a modelled system on the basis of a Behavior Annex specification;
2. failure modes and effects analysis on the basis of an Error Model Annex specification.

We also present a stepwise visualization tool (engine and applications for both mentioned examples) that is implemented in MASIW framework for design of modern avionics systems.

Keywords

model-based development, behavior specification, specifications validation, visualization

1. INTRODUCTION

Nowadays, the development process of large mission-critical systems tends to become standardized. In particular, it requires to perform system analysis and certification stages in order to check that a system under development meets all specified requirements. Modern systems are very large and complex. Manual analysis of such systems is very labor-consuming and expensive. In some cases, it is simply impossible. Thus, it is necessary to use automated tools and, consequently, to develop formal models.

Architecture Analysis & Design Language (AADL [7]) is used to model avionics systems. In fact, AADL is a standard in this area. The main purpose of AADL is to describe the system architecture and requirements. Model of a system looks like a hierarchy of hardware and software components. Some of the components are linked to each other. System requirements are specified as properties of corresponding components. AADL allows to represent models both in textual and in graphical form. Note that the textual form of AADL-models is much better than, for instance, UML's one, because it is easy to read and edit.

AADL itself does not support any means to specify algorithms. Nevertheless, there are a number of independent extensions of AADL that provide suitable ways to analyze AADL-models. In this paper, we consider two extensions: Behavior Annex [5] that specifies the behavior of a system, and Error Model Annex [6] that specifies how local failures in components impact on environment of those components.

Behavior Annex allows to monitor how a system changes its state during the given time period. In other words, it provides a way to simulate the behavior of a system.

Error Model Annex allows to monitor how failures in components propagate their effects to a system as a whole. In other words, it provides a way to perform a failure mode and effect analysis — a kind of risk analysis.

In practice, both behavior simulation and failure effect analysis may give incorrect results that does not meet the expected ones. There are different reasons of such phenomenon: errors in specifications, partially unsupported protocols, etc. In order to search for a cause of the problem, one needs convenient tools (like debugger for programming languages) that allows to perform the simulation/analysis step-by-step and show the current state of model elements at each step.

All existing tools for development and analysis of AADL-models use unified graphical representation and are compatible. Initially, this graphical representation was intended to be used statically. Given the developed graphical architectural model, one can launch various analytical applications in the context of the model as a whole or a part of it. But always the launched analysis saves results in separated tables, diagrams, traces, etc. Thus, one cannot see the intermediate steps of the analysis.

Consequently, firstly, the required debugger must be graphical and bound to source architectural diagram. In other words, steps of simulation/analysis performed in components must be shown over the corresponding elements of the diagram. Secondly, representation of AADL-model dynamics must be standardized in order to use different feasible visualization tools to show different behavioral aspects of models.

This paper concerns the problem of building standard representation of dynamics aspects of AADL-models. We suggest a unified abstract metamodel of internal representation of stepwise components interaction process.

In a separate section of the paper, we present a mapping of Behavior Annex and Error Model Annex into the suggested metamodel. It proves the adequacy of metamodel to the discussed problem.

2. RELATED WORKS

2.1 OSATE toolset

OSATE is an extensible AADL-based modelling tool used for creation and analysis of AADL models [3]. Besides other abilities, it has an instance model viewer plugin [2]. This plugin allows to display the graphical representation of AADL models in a static form:

- show components and links between them;
- show data flows: given a data flow f, highlight all links used by f;
- show error flows: given a component A, highlight all components that may be affected by (any) failure in A.

Showing data flows is closely related to Behavior Annex specification and showing error flows is related to Error Model Annex specification. But such displayed flows are fetched *statically* and, thus, all *possible* flows are displayed, but *actually* used flows (*dynamically*, under the given initial conditions) cannot be displayed. In particular, it does not display different important information like the state of each component and data/errors that are going through links.

Besides, these two seamingly connected facilities are not connected and are implemented independently.

2.2 AADL Inspector

AADL Inspector is an AADL-model processing framework [1] supporting different actions for development and analysis of AADL-models. In particular, it supports editing and checking of Behavior and Error Model Annex specifications.

AADL Inspector is integrated with the MARZHIN simulator [9]. It allows to analyze schedulability issues and visualize different states, messages and events of given components on the time scale. The bunch of displayed information is tuned easily.

Similar to OSATE, AADL Inspector does not have facilities to depict different behavioral characteristics of a single model got both from Behavior Annex and Error Model Annex.

2.3 SCADE System with medini

SCADE System [4] is a well-known and popular tool for creation of architecture models of complex systems. Recently an AADL support plugin has been developed [11]. Also, some integration with the medini approach was performed for fault-analysis [12].

These instruments can be connected into an automated tool chain for representing and visualization of AADL models with data flows and error propagation paths. It is still under heavy development but at the moment, visualization functionality allows to analyze only a single step of data or error propagation over the model.

3. SYSTEM BEHAVIOR METAMODEL

AADL core language gives an ability to define architecture of a modelled system. The AADL standard contains a metamodel definition, i.e., a bunch of terms and abstractions that are used in this language and relations between these abstractions. This metamodel contains main architectural notions: components and their categories, component interface elements (so called, features), connections and bindings (allocations), etc.

Specifications for Behavior and Error Model Annexes are applied to certain model objects. Our goal is to visualize intermediate states and dynamics of evolution of an architecture model according to its behavioral specification.

To visualize the evolution process of an architecture model, we introduce a metamodel for behavioral specifications, which consists of two parts:

- one for system architecture (which is an abstraction of the core language metamodel);
- behavior metamodel.

3.1 System architecture metamodel abstraction

We consider a modelled system as a bunch of interacting *components*. These components can represent software, hardware, composition of both, etc. To depict these interactions, we introduce the notion of *interaction channel*.

In practice, interaction channels can be of very different natures, for instance:

- relation of a container and contained;
- connection for data transmission;
- logical allocation (binding), e.g., execution of a software component on a given platform;
- close physical collocation, when, for example, heating or vibration of one component can influence the behavior of another one.

Thus, the system architecture metamodel abstraction includes the following notions:

- individual elements:
  - model component (possibly, including other components);
  - channel of interaction of two components;
- model itself, a bunch of several interacting components.

3.2 Behavior metamodel

The base of the behavioral specification of a component is its reaction on current conditions.

Behavior of a component depends on the influence of other components and on some internal events that happen inside the components themselves. We consider that each component at each step of its behaving can be in one of several states which influence the behavior. So, for a single step of behaving of a component we need

- the current state of a component;
- triggered internal events;
- influence of other components through interaction channels.

We consider the reaction of a component on each step to consist of

- change of the component state;
- influence of this component on the others through interaction channels.

Behavior of a component can be non-deterministic, so, in the same conditions a component can define several reactions.

Thus, behavior metamodel contains the following notions:

- elements describing a situation:

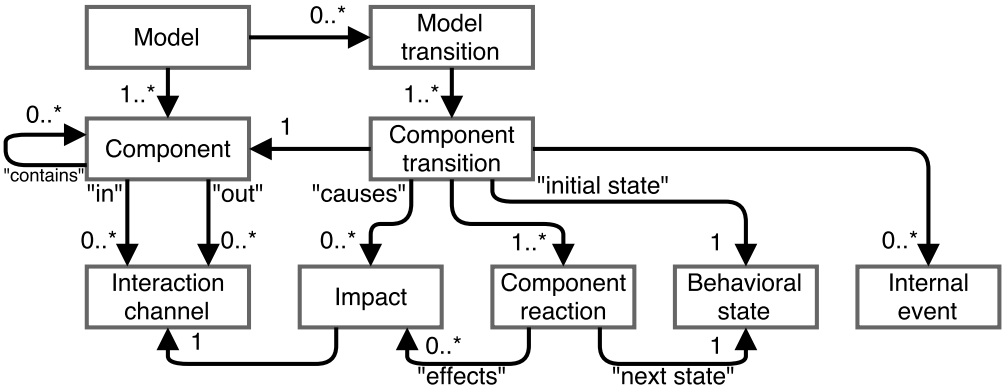

Figure 1. Suggested metamodel

- component state;
- internal events that can happen inside a component;
- type of impact of one component on another one (through the interaction channel);

- a step of component behavior, putting a set of possible reactions in current situation;
- component reaction consisting of new component state and impacts on the other components;
- a step of model behavior containing steps of component behaviors for all relevant components.

All notions of the suggested metamodel and their relations are depicted in Figure 1. On the left there are abstract system architecture notions, and on the right — behavior specification metamodel notions.

4. EXAMPLES OF THE METAMODEL APPLICATION

In this section, we show how Behavior Annex and Error Model Annex of AADL map onto the suggested metamodel. We describe which elements of these two specifications correspond to the behavior metamodel.

4.1 Behavior Annex

4.1.1 Preliminaries

Behavior Annex specification at a glance is a state machine with specific transition conditions and transition actions. Extended state machines are used, i.e., variables can be the part of the machine state in addition to the usual named states similar to finite state machines.

This state machine can define reactions on incoming signals and messages from other components. In different states and depending on variables values, the machine can differently react on the same impacts. As a reaction, component can send messages and generate events towards the other components and also change its own state and variables values.

Declaration allows explicitly declare time which it takes to execute this or that state machine transition. This allows to model time-consuming calculations and also to model and analyze latencies that are brought by components into signals processing.

In general case, Behavior Annex state machines can be non-deterministic. In particular, the following aspects can be non-deterministic:

- selection of a transition for the given state and input events;
- time for calculations during transition execution;
- order of actions on transitions;
- particular data during assignments.

Behavior Annex specifications are closely related to the architecture declarations. In particular, this annex does not have any facilities for data modelling. AADL core declarations are used for this. Also, for interactions between components facilities like ports and bindings from the AADL core are used.

4.1.2 Mapping to behavioral metamodel

Behavior Annex entities are mapped to the behavioral metamodel in the following way.

AADL components are used as behavioral model's components as they are. Components of categories like threads, processes, devices and subprograms can be used. System components can be also used for behavioral modelling of abstract subsystem in the model without precise modelling of its parts.

The main type of interaction channels used for Behavior Annex specifications is *port connection* type. AADL ports allow to transmit both data and control (events). Also, when subprograms are modelled, control can be transferred along so called *subprogram access connections*. Besides, binding relation can be used as an interaction channel (mainly, for transmitting events of control of software components by hardware ones).

Internal state of a component consists of the state machine's state and a composite state of all variables accessible by Behavior Annex specification. This state determines which transitions are available on different incoming events and data arrival.

When sources of events or data are modelled to be inside a component, AADL's *internal port* notion is used. Such ports are represented as internal events in terms of behavioral metamodel.

Impact of one component on another in behavioral model for Behavior Annex is a data message and/or an event. They can be

- a service call for different components (e.g., subprogram call);
- message transmission through some net or ports (including OS software ports);
- hardware signal (e.g., a signal from a sensor to another hardware component or control software);

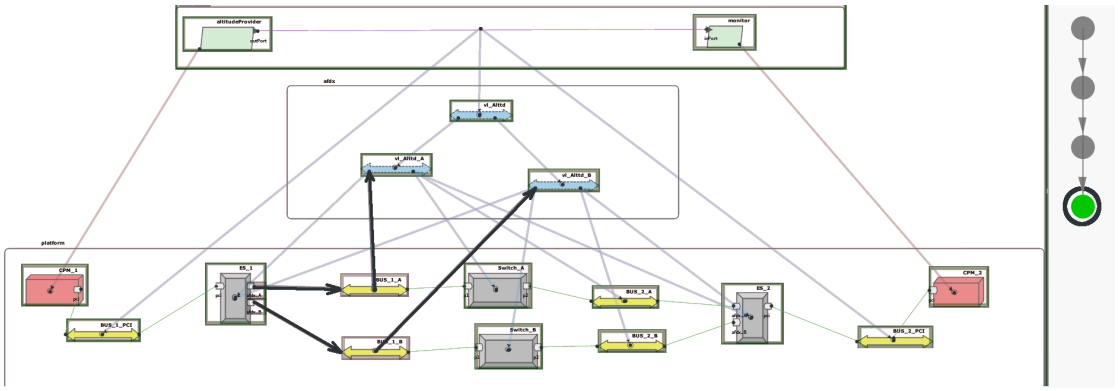

Figure 2. Behavior visualization mode

- controlling one of the components by another ones (e.g., execution or context switch events).

During execution step, component with Behavior Annex specification can

- change the state of the machine and variables;
- perform arbitrary calculations;
- spend arbitrary amount of the model time for calculation;
- send messages and generate events going out to other components through ports.

4.2 Error Model Annex

4.2.1 Preliminaries

Error Model Annex specification mainly consists of description of impact of failures in components to neighbour components. If a component A has some failure (this failure may be the result of either internal event or another failure propagated from neighbours), then it may impact the neighbour component B: as a result, B also has failure. For example, if a processor fails, then all processes that run on it immediately fail as well. Note that failure propagates from component A to component B *instantly*. Error Model Annex specification of a component declares relevant error states of the component: operational, failed, and in some cases one or several intermediate states when the component is partially failed. If a failure in a component A impacts a component B, then B may change its state. Depending on its state, the component B propagates failures to neighbour components.

4.2.2 Mapping to behavioral metamodel

In this subsection, we describe correspondence of Error Model Annex entities to elements of suggested behavioral metamodel.

Components of the behavioral metamodel are components of source AADL-model. Interaction channel between two components is a cause-effect link that represents the impact of failure of one component on another. Component state is an error state. Internal event is internal independent failures in a component. Kind of impact of one component on another is a kind of failure that one component propagates to another.

5. VISUALIZATION OF AADL-MODEL BEHAVIOR

We used the behavioral metamodel described in the paper in order to develop an engine for visualization of AADL-model behavior. The implementation base for it is MASIW [8, 10] which is a framework for development and analysis of AADL-models. Based on the developed visualization engine, we also developed plug-ins that visualize Behavior Annex and Error Model Annex.

The visualization engine provides the following support of interactive dynamic analysis process:

- choose initial events;
- perform one step of dynamic analysis;
- "run" dynamic analysis steps until reach non-determinism;
- in case of non-determinism, choose a branch for the next step;
- show history of steps and visited branches;
- navigate the history:
  - return to any previous step;
  - jump to any step of already visited branch;
- at any step, show state of any component;
- at any step, show current causes and effects.

The behavior visualization mode has two main view areas (Fig. 2):

- left one — an architecture diagram of a system under analysis (hierarchy of components, some of them are linked to each other);
- right one — a history.

The history view consists of points and arrows. Points represent different system states. Arrows represent transitions between states. Selected point is a current system state that is displayed in the architecture diagram view. Black arrows over an architecture diagram represent impacts between components in current system state.

6. CONCLUSION

In this paper, a metamodel was suggested that allows to describe behavioral aspects of modelled systems for visualization upon static diagram of architecture. Such visualization allows, in particular, to perform validation of specifications of behavioral aspects of models.

The metamodel we suggest allows to depict dynamics of AADL-model behavior specified with Behavior Annex and error propagations between the components in AADL-models with Error Model Annex specifications.

Appropriate visualizer of behavior based on the suggested metamodel has been implemented in the MASIW framework with modules for visualization of Behavior and Error Model Annexes.

We believe that this paper is the first step in the standardizing of representation of dynamic aspects of AADL-models.

REFERENCES

[1] *AADL Inspector*. http://www.ellidiss.com/products/aadl-inspector/.

[2] *OSATE IMV plug-in*. http://aadl.info/aadl/osate/osate-doc/osate-plugins/imv.html.

[3] *OSATE: Open Source AADL2 Tool Environment*. http://osate.org/.

[4] *SCADE System*. http://www.esterel-technologies.com/products/scade-system/.

[5] *SAE International standard AS5506/2, Architecture Analysis & Design Language (AADL), Annex D: Behavior Model Annex*, 2011. http://standards.sae.org/as5506/2/.

[6] *SAE International standard AS5506/1A, Architecture Analysis & Design Language (AADL), Annex E: Error Model Annex*, 2015. http://standards.sae.org/as5506/1a/.

[7] *SAE International standard AS5506C, Architecture Analysis & Design Language (AADL)*, 2017. http://standards.sae.org/as5506c/.

[8] D. V. Buzdalov, S. V. Zelenov, E. V. Kornykhin, A. K. Petrenko, A. V. Strakh, A. A. Ugnenko, and A. V. Khoroshilov. Tools for system design of integrated modular avionics. In *Proceedings of the Institute for System Programming of RAS*, volume 26, pages 201–230, 2014.

[9] Pierre Dissaux and Olivier Marc. Executable aadl: Real-time simulation of aadl models. In *ACVI 2014 — Architecture Centric Virtual Integration Workshop Proceedings*, pages 59–68, 2014.

[10] Alexey Khoroshilov, Dmitry Albitskiy, Igor Koverninskiy, Mikhail Olshanskiy, Alexander Petrenko, and Alexander Ugnenko. AADL-based toolset for IMA system design and integration. *SAE Int. J. Aerosp.*, 5:294–299, Oct 2012.

[11] Thierry Le Sergent. System design with SCADE avionics package. In *ANSYS Model-based Systems Engineering (MBSE) Seminar*, May 2017.

[12] Michael Soden. Fault analysis with medini. In *ANSYS Model-based Systems Engineering (MBSE) Seminar*, May 2017.