

Modelling Composite End-to-End Flows with AADL

Naeem Muhammad¹, Yves Vandewoude¹, Yolande Berbers¹, Sjir van Loo²
¹*Department of Computer Science, Katholieke Universiteit Leuven, Belgium*
{Naeem.Muhammad, Yves.Vandewoude, Yolande.Berbers} @cs.kuleuven.be
²*Embedded Systems Institute, Netherlands*
Sjir.van.Loo@esi.nl

Abstract

AADL is a modelling formalism that supports modelling and analysis of functional and non-functional properties of embedded systems. End-to-End (EtE) flow latency is the amount of time consumed by the contributing components for a specific flow of information. Currently, AADL requires that flow specifications of the contributing components are connected through the AADL connector element and does not provide support for composite EtE flows: flows that themselves consist of multiple discrete EtE flows. In this paper we discuss how this issue can be overcome and we propose two possible extensions that allow AADL to model composite EtE flows. We also developed a tool to analyse AADL models for composite EtE flows. We will describe results of the analysis performed with the tool.

Keywords—Performance Modelling, AADL, End-to-End Flow, Embedded Systems.

1. Introduction

Non-functional constraints are typically linked to many different functional blocks of the system and the impact of each block is not always clear [13]. In addition, properties of the individual subsystems are not always known in detail until the system is completely finished. Nevertheless, despite such difficulties, evaluation of non-functional properties at the level of the architecture is cost effective since it provides insights that help to understand the realization of the different requirements before the system is

implemented. Embedded systems in particular have many different soft or hard real-time constraints such as efficiency or response time (the property that represents the total time consumption for a particular task).

Much focus has been placed on modelling such properties in the last few years which has resulted in the addition of support for such features in some architecture description languages (ADLs). One of the best known and most actively used architecture description language for embedded systems is the Architecture Analysis and Design Language (AADL) [1]. Originally developed for modelling and analysis of systems in the domain of avionics [5], it has been standardized by the Society of Automotive Engineers. Because of its rich modelling and analysis capabilities it is widely used for embedded systems in other domains as well. AADL provides a formalism for modelling response times in terms of End-to-End (EtE) flows, the description of these flows is used to perform the flow latency analysis. EtE flows are logical flows through a sequence of system components for a certain flow of information [2]. In general, every system consists of multiple flows. Each flow has a single starting and ending point but may internally consist of many distinct flows. Such composite flows are most interesting from an architecture abstraction point of view since they often represent more abstract flows of information through the system.

Currently, however, AADL has only limited support for modelling top-level EtE flows: only those flows that can be directly mapped to distinct internal flows can be modelled. All other flows of information, such as those interrupted by asynchronous communication, are currently not supported.

In this paper we describe how we have overcome this shortcoming by extending AADL with support for top-level EtE flows. The remainder of this paper is

This work has been carried out as a part of the Condor project (<http://www.esi.nl/Projects->Condor>) at FEI company under the responsibilities of the Embedded Systems Institute (ESI). This project is partially supported by the Dutch Ministry of Economic Affairs under the BSIK program.

structured as follows. In section 2 we briefly introduce AADL. We describe how AADL can be used to model EtE flows and discuss the problems with the current approach in section 3. In section 4 we present our own solution and apply this solution on an industrial case in section 5. Related work is discussed in section 6; conclusion and future work is given in section 7.

2. Architecture Analysis and Description Language (AADL)

Architecture Description Languages are modelling languages that provide support for describing system architectures through their formal notations. A concise definition of an ADL according to Medvidovic et al. [4] is “an ADL must explicitly model components, connectors, and their configurations; furthermore, to be truly usable and useful, it must provide tool support for architecture-based development and evolution”. In compliance to the definition of ADL, AADL provides a modelling formalism accompanied by a toolset to support modelling activities and analysis. In this section a brief introduction is given on the AADL architectural notations, its analysis and its tool support.

a. AADL Architectural Elements

AADL consists of a rich set of architectural elements for modelling components, their interactions and their configurations. Architectural elements and the core concepts of the language (based on the AADL standard V1) are given in figure 1. Components in AADL are used in terms of component types and component implementations. A component

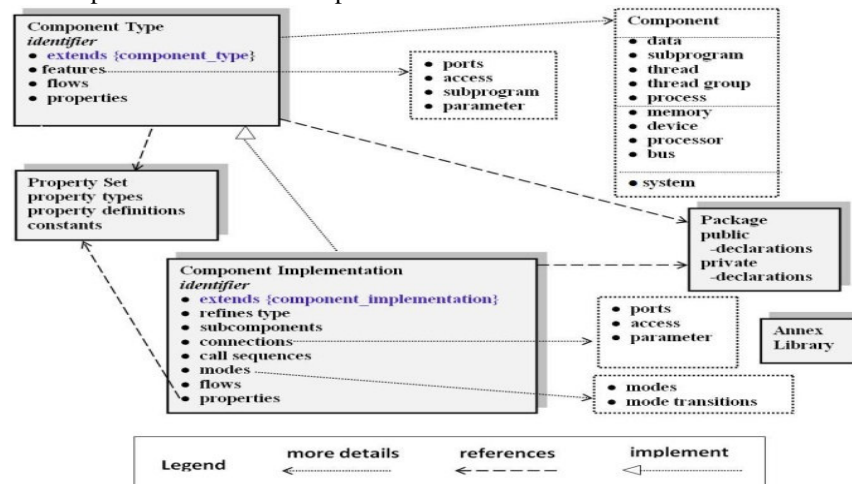


Figure 1 AADL Architectural Elements [3]

Interactions between components can be realized by using *features* and *connections*. *Features* are interaction points of components, which are used for

type defines the externally visible characteristics of a component usually by using features, flows and properties. Whereas, a component implementation models the internal structure of the component; an internal structure may consist of subcomponents, connection among them, flows across them and their operational behaviour. AADL distinguishes between three types of components: software components, hardware components and composite (system) components:

Software Components:

Thread: Represents a unit of sequential execution through source code.

Thread group: Represents a logical grouping of threads.

Process: Represents a protected address space.

Data: Represents static data and data types.

Subprogram: Represents a callable part of a source code.

Hardware Components:

Processor: Hardware that is responsible for executing threads.

Device: Hardware that interacts with the external environment.

Bus: Hardware that provides access to the other execution platform components.

Memory: Hardware that stores digital data.

Composite (System) Component: A component composed of software and hardware or even system components.

used to exchange data and events. AADL categorizes *ports* as *data ports*, *event ports* and *event data ports*. Moreover, multiple ports can be grouped together in a *port group*. *Component access* enables components to access shared data and bus. For access, components are required to explicitly use *provides access* and *requires access* declarations. *Subprogram calls* are used for synchronous calls to subprogram components, and *parameters* are used to represent data values passing in and out of a subprogram. Connectors are used to connect interaction points of components. AADL provides *data*, *event*, *eventdata*, *dataaccess*, *bussaccess* and *portgroup* connectors.

b. Analysis and Tool Support

In its conformity to the ADL definition, AADL provides support for various kinds of analyses along with conventional modelling. A few of the supported analysis are:

- Flow Latency Analysis
- Resource Consumption Analysis
- Real-Time Schedulability Analysis
- Security Analysis
- Safety Analysis

Various tools are available to perform these analyses; some of them are OSATE, ADeS, Cheddar and ANDES.

OSATE (Open Source AADL Tool Environment) [9] developed by SEI is a set of Eclipse plug-ins for front-end processing and various analyses of AADL models. ADeS (Architecture Description Simulation) [10] by Axlog simulates various aspects of the system behaviour. Such as scheduling of processes and threads by processors.

Cheddar [11] developed by LISyC Team is a real-time scheduling tool, which provides support for quick prototyping of real-time schedulers and schedulability analysis. Like ADeS, Cheddar also supports simulation of scheduling properties of a system.

ANDES (ANalysis-based DEsign tool for wireless Sensor networks) [12] was developed for modelling and analysis of wireless sensor network systems. It provides support for real-time communication schedulability, target tracking and real-time capacity analyses.

3. End-to-End Flows with AADL

Flows in AADL describe the different sequences of an information flow through a set of contributing components. The description of this flow is subsequently used in certain analyses such as a flow

latency analysis. In AADL flows are defined through flow specification and flow implementation. A flow specification represents the externally visible flow of information in a component; it is specified within the component type declarations using flow sources, flow paths and flow sinks [2]. A flow source represents the originator of the flow, the flow sink represents the end consumer of the flow information, and the flow path embodies the link between incoming and outgoing ports involved in the flow.

A flow implementation on the other hand represents the actual realization of a flow within a component; it is specified within the component implementation declarations.

```

system motion_client
  flows
    start_motion_flow: flow source C_start_motion;
    move_status_flow: flow sink C_motion_status;
end motion_client;

system motion_server
  flows
    start_motion_flow: flow sink S_start_motion;
    axis_move_flow: flow source S_move_axis;
    move_status_flow: flow path S_move_status ->
      S_motion_status;
end motion_server;

device motion_controller
  flows
    axis_move_flow: flow sink Ctr_move_axis;
    move_status_flow: flow source Ctr_move_status;
end motion_controller;

```

Figure 2 AADL Flow Specification

An example of both a flow specification and a flow implementation is given in Figures 2 and 3, which are taken from our industrial case: an electron microscope. The excerpts are taken from the electron microscope's motion-subsystem that is responsible for moving the specimen holder. The system consists of three major components working in a client-server environment: the motion client, the motion server and the motion controller. The motion server receives its stage movement commands from a client application, processes it and moves the motion controller to the desired position. The externally visible flow of the move command is shown in Figure 4, which corresponds with the textual AADL representation in Figure 2.

```

system implementation motion_server.server_app

--following flow receives information at a sink port of the server
--and passes it to a flow specification of the MdlMotion
--subcomponent of the server through a connector.
start_motion_flow: flow sink S_start_motion
->ServertoMdlConnector->
    MdlMotion.start_motion_flow;

--following flow connects a flow specification of the MdlMotion
--subcomponent to a flow specification of the HalMotion
--subcomponent through a connector and continues through
--another connector to an out port of the Server.
axis_move_flow: flow source MdlMotion.axis_move_flow
->MdltoHalConnector->
    HalMotion.axis_move_flow
->HaltoServerConnector->
    S_move_axis;

--following flow passes information received from an incoming
--port of the server to the HalMotion subcomponent through a
--connector, subsequently it connects a flow specification of
--the HalMotion to a flow specification of the MdlMotion by
--using another connector. Finally, it connects a MdlMotion flow
--specification to an out port of the server.
move_status_flow: flow path S_move_status
->ServertoHalConnector->
    HalMotion.move_status_flow
->HaltoMdlConnector->
    MdlMotion.move_status_flow
->MdltoServerConnector->
    S_motion_status;

end motion_server.server_app;

```

Figure 3 AADL Flow Implementation

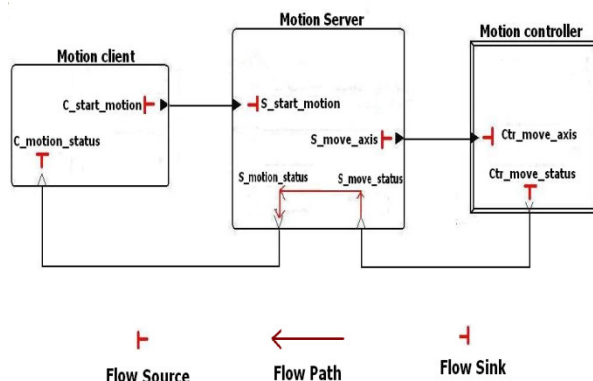


Figure 4 Motion-Subsystem Move Command Flow

EtE flow latency analysis requires the specifications of EtE flows. An EtE flow represents a logical flow of information from a source to destination passing through various system components. It is defined in the component implementation (typically in the top level component

in the system hierarchy) and refers to the specifications of other flows in the system.

The EtE flow specification consists of the flow specification of the contributing components connected through the AADL connector, Figure 5 contains the standard syntax for EtE flow specification.

```

end_to_end_flow_spec ::=
    defining_end_to_flow_identifier: end to end flow
    start_subcomponent_flow_identifier
    { -> connection_identifier ->
        flow_path_subcomponent_flow_identifier
    }*-> connection_identifier
    -> end_subcomponent_flow_identifier
    [ {(property_association)+} ]
    [ in_modes_and_transitions ];

```

Figure 5 Standard AADL EtE Flow Syntax

According to the standard specification, an EtE flow starts with a flow specification of the starting subcomponent, connects it to a flow specification of the subsequent subcomponent and so on, and finally connects to a flow specification of the last component in a flow. A notable point in the EtE flow modelling is that each contributing flow specification is connected to its adjacent flow specifications. The starting and ending flows are connected to one flow each, all intermediate flows are connected to both the predecessor and successor flows. This concept restricts EtE flow analysis only to those flow specifications that are linked. AADL does not provide support for more abstract flows whose internal flows are not linked. For clarity, we distinguish in this paper between discrete EtE flows (in which all sub-flows are connected) and composite EtE flows (that consists of unlinked sub-flows).

In case of our motion-subsystem example, `Move_Stage` is a composite EtE flow: the flow starts when a client application sends a move request to the server and ends when client gets acknowledgement that stage has been moved to a desired position. Internally, `Move_Stage` consists of three consecutive discrete (disconnected) EtE flows `Start_motion_flow`, `Axis_move_flow` and `Move_status_flow`.

Start_motion_flow:

The flow starts with a move request from the client application and ends with the server. Internally, the server simply places the command in a queue, therefore this flow ends here. A part of the AADL textual representation given in Figure 2 specifies this behaviour of the flow.

Axis_move_flow:

As soon as a move command is available in the queue, a component of the motion server processes it and generates a new instruction for the motion controller. The server component involved in this task is different from the component involved in previous EtE flow. More importantly, both are not connected with each other for this particular task. Since the flow specifications of both components are not linked with each other, according to the AADL specification they can not be a part of a single EtE flow. Hence, this results in another EtE flow starting from the motion server and ending with the motion controller. Specification of the flow can be seen in Figure 2. The motion controller upon receipt of the instruction moves the stage to the desired position.

Move_status_flow:

As soon as the stage is moved to its position, the motion controller sends a motion completion acknowledgement back to the server which subsequently dispatches it to the client application. Internally, two different subcomponents of the motion controller are responsible for stage movement and acknowledgement generation. As such, the flow specifications of both components are not connected with each other. Therefore, sending acknowledgement back to the server is the start of a separate EtE flow with the motion controller as its starting point and the client as its ending point. Figure 2 contains the specification of the flow.

AADL's incapability to model composite EtE flows exists at any level of abstraction in AADL models, although chances of having such flow specifications increase with increase in abstraction. Therefore, need for modelling and analysis of such EtE flows is significant at higher abstraction (system architecture) levels. Providing modelling support for composite EtE flows will also enhance flow latency analysis. The capabilities will enable system architects to analyse system flows at higher abstraction.

4. Modelling Composite EtE Flows with AADL

The incapability, as we described earlier exist because AADL does not provide any support for linking disconnected flow specifications. Bridging such flow specifications can enable modelling and analysis of composite EtE flows.

Initially designed as a language for modelling avionic systems, AADL includes core modelling concepts and certain analyses essential for real-time

systems in the aerospace domain. However, during its design, it was foreseen that use of AADL in other domains could require additional modelling concepts and analyses. To meet potential needs AADL was designed as an extensible ADL.

In order to address the shortcomings, we can use one of the extension mechanisms available in AADL. It is possible to extend the AADL concepts either by introducing new properties to the modelling elements, by addition of new modelling notations, or by developing a sublanguage as annex to the AADL standard [6]. The latter technique is mainly used for large-scale extensions and was considered out of scope for our own purpose. Therefore, we have focused ourselves to the first two techniques in order to extend the AADL concepts for modelling composite end-to-end flows.

a. Property Based Solution

An AADL property provides descriptive information about components, subcomponents, features, connections, modes, subprogram calls and flows [3]. A property consists of an associated value and type; the AADL standard consists of a set of predefined properties. However, new properties can be introduced in order to add additional information about the above mentioned architectural elements. The standard properties for EtE flows are:

- Expected_Latency: Time
- Actual_Latency: Time
- Expected_Throughput: Data_Volume
- Actual_Throughput: Data_Volume

We introduce a new property called `Link_Flow` to the existing properties. The new property holds a string value representing the identifier of the EtE flow that is to be linked. New properties are defined in the AADL property sets. We declare our new property in the `FLOWCONN` property set, the declaration of our new property is:

```
property set FLOWCONN is  
  
Link_Flow: aadlstring applies to (flow);  
  
end FLOWCONN;
```

Afterwards, the `Link_Flow` property can be used in the EtE flow declaration, in which it is assigned the identifier of the EtE flow to be linked.

b. Customizing AADL Connector

An AADL EtE flow starts with a source component and ends with a sink component. In a top-level component where EtE flows are defined, source and sink are represented by AADL ports; called source and sink port respectively. Two distinct EtE flows can be transformed into one by connecting the sink port of the first flow to the source port of the second flow. This concept can be used to connect multiple distinct EtE flows by connecting the sink port of a flow in the sequence with the source port of the next.

AADL does not provide support for linking a sink port to a source port in above mentioned scenario; however we can realize such linkage by extending the AADL connection construct.

While linking discrete EtE flows by using any of the proposed solutions we assume in this paper that the delay between adjacent EtE flows is zero. Although this is the case in the motion subsystem (information is passed on through shared memory), many scenarios can be thought of in which this is not the case (such as the presence of a queue). Although not addressed in depth in this paper, our solution remains applicable for those cases as well by including the delay in the model as a connector-like construct and applying a statistical model to them. The resulting latency of the EtE flow will also be a statistical distribution. Further examination of this issue is a future work.

In this paper, we will focus specifically on extension through addition of new properties. The introduction of properties allows us to attach additional information to the different elements of the AADL model. Subsequently this information is manipulated by the accompanying tools that carry out the EtE flow latency analysis. The main reason for our choice of the property based solution instead of the connector based approach is that it does not require any change in the core AADL concepts and is therefore easier to implement. However, an implementation of the connector based solution is planned as well, as it seems a more elegant way of addressing the issue.

5. Case Study

In the previous section we defined a new property `Link_Flow` to connect discrete EtE flows. Now, we apply the proposed solution on the motion-subsystem, by using the property to connect internal discrete EtE flows of the `Move_Stage` flow. The use of our property within the standard AADL syntax is shown in Figure 6.

```
end_to_end_flow_spec ::=
  defining_end_to_flow_identifier: end to end flow
  start_subcomponent_flow_identifier
  { -> connection_identifier ->
    flow_path_subcomponent_flow_identifier
  }* -> connection_identifier
  -> end_subcomponent_flow_identifier
  [{"FLOWCONN::Link_Flow=>"identifier_of_subsequent_EtE_flow" | (property_association)+}]
  [in_modes_and_transitions];
```

Figure 6 EtE Flow Syntax with Link_Flow Property

As stated earlier the `Move_Stage` EtE flow is composed of `Start_motion_flow`, `Axis_move_flow` and `Move_status_flow` in sequence. By using the `Link_Flow` property we will link `Start_motion_flow` to `Axis_move_flow`, and `Axis_move_flow` to `Move_status_flow`. AADL textual description of the linkage is given in subfigures 7(a) and 7(b).

```
Start_motion_flow: end to end flow
  MotionClient.move_request_flow ->
  ClienttoServerMotionConnection ->
  MotionServer.move_request_flow
  { FLOWCONN::Link_Flow => "Axis_move_flow"; };
  (a)

Axis_move_flow: end to end flow
  MotionServer.axis_request_flow ->
  ServertoControllerAxisConnection ->
  MotionController.axis_request_flow
  { FLOWCONN::Link_Flow => "Move_status_flow"; };
  (b)

Move_status_flow: end to end flow
  MotionController.status_flow ->
  ControllertoServerStatusConnection ->
  MotionServer.status_flow ->
  ServertoClientStatusConnection ->
  MotionClient.status_flow;
  (c)
```

Figure 7 Link_Flow Property for Move_Stage Composite EtE flow

Since `Move_status_flow` is the last EtE flow in the composition therefore it will not be linked further, as shown in subfigure 7(c).

Using our technique, we can link any number of EtE flows in a composition. Afterwards, the value of the defined property can be used during analysis to calculate the total flow latency of the `Move_stage` EtE flow. The calculation starts with the latency of the first flow, subsequently adding the latency of the EtE flow given in the `Link_Flow` property value. The addition

continues until an EtE flow without `Link_Flow` property (or with empty value) is found.

Our work is built on top of the AADL design and analysis-tool OSATE. Since OSATE itself is built on top of the Eclipse platform, it is very well suited as a basis for the development of tools that operate on AADL models. OSATE's extensible plug-in architecture provides a wide range of methods and services generated from the AADL meta model that can be used by plug-ins to manipulate AADL models.

We have developed an OSATE-plugin that analyzes AADL models for composite EtE flows. By using the proposed property `Link_Flow` the plug-in differentiates composite flows from distinct flows, counts them, identifies their compositions (list of the contributing discrete EtE flows) and calculates their latencies (total time consumed by the composite EtE flow). The results of the analysis of our tool on the motion-subsystem are shown in Figure 8.

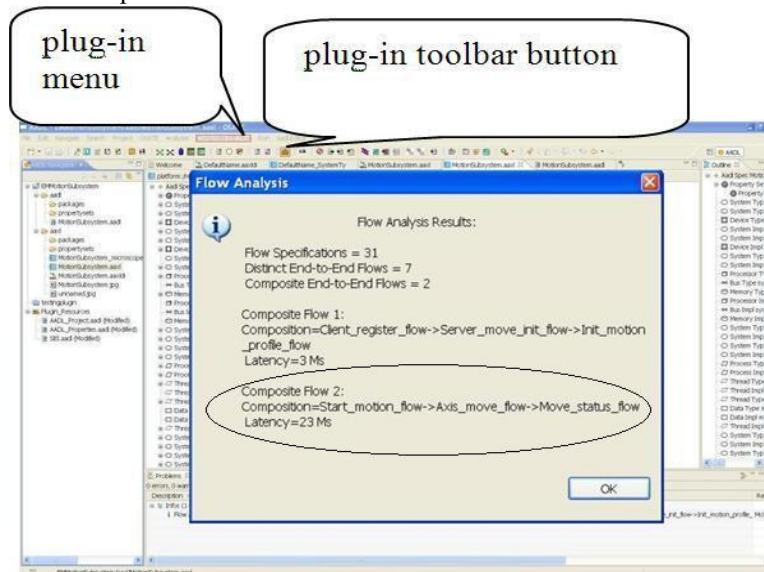


Figure 8 Composite EtE Flow Analysis

The analysis shows that the motions-subsystem contains 31 flow specifications, 7 discrete and 2 composite EtE flows. The `Move_stage` composite EtE flow is among the two identified by our tool. The encircled part of the Figure contains its composition and latency. Statistics shown on the report represent complete flow specification of the motion-subsystem; description of all flow specifications and composite EtE flows is not relevant here.

The validity of the tool results was tested against results of manual analysis. Our experiment not only clearly showed that our tool automates (and hence speeds up) the analysis process, but also that the approach presented in this paper is valid.

6. Related Work

Many aspects of EtE flow modelling have been explored in various studies. We will discuss them here in relation to our work.

Lee, Mallet and Simone [7] performed a study to understand how an AADL flow latency model can be represented with the MARTE profile. In this regard, authors discussed mapping of the AADL elements to MARTE elements, and compared models developed

with both modelling formalisms. The focus of their work is on modelling and analysis of discrete EtE flows with both AADL and MARTE. Whereas, we distinguished discrete EtE flows from composite EtE flows, and modelled composite flows by linking discrete EtE flows.

In another study, Oleg and Alexander [8] analyzed EtE timing properties of an AADL model with the help of the Real-Time Calculus (RTC) technique, because of its suitability for analyzing hard real-time timing properties. They proposed an algorithm for transformation of the AADL models to RTC based analytical model. They applied the proposed approach on a wireless sensor network architecture system to analyze the EtE delays. Our work differs from theirs in that we enhanced the AADL modelling and analysis capabilities for timing (flow) analysis, whereas they transformed the AADL models to RTC models to better analyse the timing properties.

7. Conclusions and Future Work

In this paper we highlighted the issue of modelling composite EtE flows with the AADL architecture description language. We discussed different possible solutions, and discussed in detail the property based solution, which uses a new AADL property to link distinct (disconnected) EtE flows.

We applied the proposed solution to an industrial case of motion-subsystem, a subsystem of the electron microscope software. The case study included modelling a composite EtE flow of the motion-subsystem, which is composed of three discrete EtE flows. With the help of a new AADL property we successfully linked the discrete EtE flows to model them as a single abstract EtE flow. Our proposed solution will enable system architects to model and analyse highly abstract system flows. In addition, we developed a tool to automate the composite EtE flow analysis. Our tool will aid system architects during analysis activities.

Moreover, in our work we distinguished system EtE flows into two categories i.e. discrete and composite. The two categories basically define two levels in the EtE flows; composite being one level higher than its composing (discrete) flows. This perspective of our work will help in developing in depth understanding about the system flows. We believe that our technique and tool support will help in developing a comprehensive understanding of the system flows and their latencies. This subsequently will lead to a better analysis of the system qualities at the early stage in the development life cycle. Results of the case study confirmed that our technique is quite efficient to support system architects during the system analysis.

Our future work remains to broaden our work by modelling composite EtE flows by extending capabilities of the existing AADL connector element as well. This will also provide us an opportunity to compare it with the technique we used in this paper.

Currently, our technique supports modelling of a single link for each discrete EtE flow. There are certain cases where a single EtE flow is required to be linked with multiple EtE flows. We will extend our solution to support modelling and analysis of such EtE flows.

8. References

[1] Feiler, P.H.; Lewis, B.A.; Vestal, S., "The SAE Architecture Analysis & Design Language (AADL) A Standard for Engineering Performance Critical Systems," *Computer-Aided Control Systems Design, 2006 IEEE International Symposium on*, pp.1206-1211, 4-6 Oct. 2006

[2] Feiler, P.H.; J. Hansson, "Flow Latency Analysis with the Architecture Analysis and Design Language (AADL)," Technical Note CMU/SEI-2007-TN-010, Software Engineering Institute, 2007

[3] Feiler, P.H., Gluch, D.P., and Hudak, J.J., "The architecture analysis & design language (AADL): An introduction," Technical Report CMU/SEI-2006-TN-011, Software Engineering Institute, 2006

[4] Medvidovic, N.; Taylor, R.N., "A classification and comparison framework for software architecture description languages," *Software Engineering, IEEE Transactions on*, vol.26, no.1, pp.70-93, Jan 2000

[5] Feiler P.H., Lewis B., Vestal S., "The SAE Avionics Architecture Description Language (AADL) Standard: A Basis for Model-Based Architecture-Driven Embedded Systems Engineering," *In Proceedings of the RTAS 2003 Workshop on Model-Driven Embedded Systems*, May, 2003

[6] Frana, R.B.; Bodeveix, J.-P.; Filali, M.; Rolland, J.-F., "The AADL behaviour annex -- experiments and roadmap," *Engineering Complex Computer Systems, 2007. 12th IEEE International Conference on*, pp.377-382, 11-14 July 2007

[7] Su-Young Lee; Mallet, F.; de Simone, R., "Dealing with AADL End-to-End Flow Latency with UML MARTE," *Engineering of Complex Computer Systems, 2008. ICECCS 2008. 13th IEEE International Conference on*, pp.228-233, March 31 2008-April 3 2008

[8] Oleg Sokolsky; Alexander Chernoguzov, "Analysis of AADL Models Using Real-Time Calculus with Applications to Wireless Architectures," Technical Report MS-CIS-08-25, University of Pennsylvania Department of Computer and Information Science, 23 July 2008

[9] SEI. The SEI Open Source AADL Tool Environment (OSATE), <http://www.aadl.info/>

[10] AXLOG. ADeS: A Simulator for AADL http://www.axlog.fr/aadl/ades_en.html

[11] Singhoff, F. and Plantec, A., "AADL modeling and analysis of hierarchical schedulers," *In Proceedings of the 2007 ACM international Conference on Sigada Annual international Conference*, pp. 41-50, 4- 8 Nov. 2007

[12] Prasad, V.; Ting Yan; Jayachandran, P.; Zengzhong Li; Son, S.H.; Stankovic, J.A.; Hansson, J.; Abdelzaher, T., "ANDES: An ANalysis-Based DEsign Tool for Wireless Sensor Networks," *Real-Time Systems Symposium, 2007. RTSS 2007. 28th IEEE International*, pp.203-213, 3-6 Dec. 2007

[13] Balsamo, S., Bernardo, M. and Simeoni, M., "Performance evaluation at the software architecture level," *In Formal Methods for Software Architectures*, volume 2804, pp. 207--258, 2003