Faculty of Information and Communications Technologies

An Evaluation of the Architecture Analysis and Design Language for Automobile Applications

Technical Report: C4-01 TR M52

Stefan Björnander, Mälardalen University, Sweden
Lars Grunske, Swinburne University of Technology, Australia
March 2, 2009

SWINBURNE UNIVERSITY
OF TECHNOLOGY

An Evaluation of the Architecture Analysis and Design Language for Automobile Applications

Stefan Björnander[1,2] and Lars Grunske[2]
[1]School of IDE, Mälardalen University, Box 883, 72123 Västerås, Sweden,
Phone: +46 21 101 689, stefan.bjornander@mdh.se
[2]Faculty of ICT, Swinburne University of Technology Hawthorn,
VIC 3122, Australia, Phone: +61 3 9214 5397, lgrunske@swin.edu.au

March 2, 2009

Abstract

Due to the growing complexity of software systems in modern automobiles, the software architecture design phase becomes more and more important. Useful tools and languages are specifically needed to manage this complexity. Recently, the Architecture Analysis and Design Language (AADL) has been proposed as a modeling language for critical systems and this language is becoming increasingly popular in the automobile industry. In order to test whether AADL is suitable for modeling software in automobile systems, we provide in this technical report a critical evaluation of the capabilities and features of this new language. Specifically, we evaluate AADL against a set of requirements that have been identified based on an exhaustive literature review and experiences gained from modeling an Adaptive Cruise Control (ACC) in AADL. The objective of this paper is to evaluate the suitability of AADL and identify gaps that would require future research and development.

Keywords: AADL.

Contents

1 Introduction **4**

2 Requirements on an Architecture Description Language for Automobile Systems **5**
2.1 Language Related Requirements 5
2.2 Method Related Requirements 7
2.3 Tool Related Requirements 9

3 Evaluation of the AADL Modeling Language **10**
3.1 Language Related Requirements 10
3.2 Method Related Requirements 10
3.3 Tool Related Requirements 14

4 Conclusions **14**

References **15**

A Code Listings and Examples **17**

1 Introduction

As software systems in modern automobiles become increasingly complex, it is important to appropriately structure the software to ease its development. In the eighties, when the automobiles originally become equipped with software, it was divide into modules and hence relatively easy to install even thought the modules were developed by different manufactures.

Today, however, a automobile can hold two thousand functions, fifty controllers, and ten million lines of source code [10]. Moreover, in many cases different systems need to communicate with each other through buses. Often the communication is delegated to only a few buses.

A useful tool to manage the complexity is the Architecture Design Languages (ADLs). They allow the developer to model the system at a high level. In this way, potential design errors will hopefully be detected before the implementation phase.

One of the most popular ADLs in the industry is the Architecture Analysis and Design Language (AADL) [9], which supports the modeling of system hardware and software. It is based on MetaH [21] and UML 2.0 [1, 17, 18]. AADL was originally developed for the avionic industry, but has later on been adapted by the automobile industry.

In order to evaluate AADL's suitability to model an automobile system, we have defined a set of requirements based on a literature review on software development methods for automobile systems. Furthermore, requirements are derived from experiences gained from modeling an Adaptive Cruise Controller (ACC) [4]. The ACC is a part of an automobile which purpose is to control the vehicle speed with regards to the surrounding environment. The requirements are grouped into three categories: language support, tool support and process support.

The objective of this paper is to evaluate AADL against the set of requirements. The rest of this technical report is structured as follows: Section 2 describes the requirements for an architecture description language for automobile systems and Section 3 evaluates AADL against these requirements. If AADL fulfills a requirement, evidence is provided that shows the fulfillment. Section 4 draws some conclusions and describes some future works.

2 Requirements on an Architecture Description Language for Automobile Systems

To analyze the suitability of AADL in the automobile domain, we have assembled a set of specific requirements for that domain. The basis for the requirements was a comprehensive literature review focussed on software engineering research for automobile systems and the experience we have gained from modeling example systems such as an adaptive cruise control system with AADL. The requirements will be grouped into the following three categories: Language support, Tool support and Process support. The main requirements in these categories will be labeled with **Req-L**x, **Req-T**x, and **Req-P**x, where **L**, **T**, and **P** indicate the categories and x is a running number. Furthermore, each of these main requirements may be associated with one or more sub-requirements. The sub-requirements will be abbreviated with the labels **Req-L**$x.y$, **Req-T**$x.y$, and **Req-P**$x.y$, where **L**, **T**, and **P** indicates the categories and x and y are running numbers. Some requirements may fit into multiple categories; in that case, multiple labels will also be associated with the requirement. Furthermore, to ensure traceability we will for each of the requirements indicate the source. This will make the process of analyzing AADL with these requirements more transparent.

2.1 Language Related Requirements

The language-related requirements for an architecture description language in the automobile domain are mostly associated with specific required notations and diagram types:

> **Req-L1.** An architecture description language for automobile systems should be able to specify the functionality required by the user in the Functional Design/Analysis Architecture (FAA/FDA) [2].
>
> **Req-L2.** An architecture description language for automobile systems should be able to specify the environment in which the system operates in the Operational Architecture (OA) [2].
>
> **Req-L3.** An architecture description language for automobile systems should be able to specify the high level structure of the system in the Logical Architecture (LA) [2], or Logical Component Architecture (LCA) [5]. The logical architecture is also specified in a Structure Diagram [2].
>
> **Req-L4.** An architecture description language for automobile systems should be able to specify the hardware platform that is used to execute the software in the Technical/Hardware Architecture (T/HA) [2], Hardware Architecture (HA) [5] or Platform Architecture [16].

Req-L5. An architecture description language for automobile systems should be able to specify the software structure in the Software Architecture (SA) [5, 16].

Req-L6. An architecture description language for automobile systems should be able to specify the deployments structure that maps software elements that are part of the software architecture to hardware elements that are part of the hardware architecture in the Deployment Architecture (DA) [5, 16].

Req-L7. An architecture description language for automobile systems should be able to specify the intra-component behavior (interactions between components), e.g in a Dataflow Diagram [2].

Req-L8. An architecture description language for automobile systems should be able to specify the component behavior, e.g with a State Chart (SC) [14] or a Mode and State Transition Diagram (MSTD) [2].

For the Functional Design/Analysis Architecture (FAA/FDA) (Req-L1) the following sub-requirements are further relevant:

Req-L1.1. An architecture description language for automobile systems should be able to specify requirements in a structured way, e.g. with a Functional Requirements Specification Architecture [5].

Req-L1.2. An architecture description language for automobile systems should be able to specify interaction and interdependencies between requirements [5].

For the Hardware Architecture (HA) (Req-L4) and Software Architecture (SA) (Req-L5) the following sub-requirements are further relevant:

Req-L4.1 and Req-L5.1. An architecture description language for automobile systems should be able to specify middleware and operating systems [16].

Req-L5.2. An architecture description language for automobile systems should be able to specify a Software Architecture that defines software elements as software tasks [5].

Req-L5.3. An architecture description language for automobile systems should be able to specify a Software Architecture that distinguishes class versus instance concepts [16].

Req-L5.4. An architecture description language for automobile systems should be able to specify a Software Architecture that is able to define type hierarchies [16].

For the specification of the intra-component behavior (Req-L7) and component behavior (Req-L8), the following sub-requirements are relevant:

> **Req-L7.1 and Req-L8.1.** An architecture description language for automobile systems should be able to specify not only state changes but also changes in different operational modes [2].
>
> **Req-L7.2 and Req-L8.2.** An architecture description language for automobile systems should be based on a sound communication scheme (well defined semantics) [2]. Examples of these communication schemes are time triggered architectures or event triggered architectures [2,16].

2.2 Method Related Requirements

The development of automobile systems requires specific methods to be used in the development process. Specifically mentioned are the following methods that are either required by law or certification authorities or desirable to save development costs: consistency checking between different diagram types [2], model-based development [10], tracing (e.g. requirements tracing) [10], modular construction and verification [5, 6], product line engineering [5] and variability management [16], cost estimation [10], change management [10], architecture evaluation [10], quality assurance and quality management [10], supplier management [5], and software acquisition management [10]. Consequently, for the method support of an architecture description language for automobile systems, we would define the following requirements:

> **Req-M1 and Req-T5.** An architecture description language for automobile systems should provide support for checking consistency between architectural views and diagram types [2].
>
> **Req-M2.** An architecture description language for automobile systems should provide support for model-based development [10].
>
> **Req-M3.** An architecture description language for automobile systems should provide support for tracing including requirements tracing and tracing to documents in later phases like implementation and test [10].
>
> **Req-M4.** An architecture description language for automobile systems should provide support for modular construction of the systems and modular verification [5, 6].
>
> **Req-M5.** An architecture description language for automobile systems should provide support for product line engineering [5] and variability managements [16] since each car is adapted to the specific needs of the customer.

Req-M6. An architecture description language for automobile systems should provide support for cost estimation [10].

Req-M7. An architecture description language for automobile systems should provide support for change management [10]. Reasons for changes are among others: requirements changes, changes in the legal environment or changes in the business environment.

Req-M8. An architecture description language for automobile systems should provide process support for evaluating the quality of the software architecture and the system to be build [10]. This process is also known as quality assurance and quality management [10].

Req-M9. An architecture description language for automobile systems should provide support for supplier management [5] and software acquisition management [10].

The quality assurance at the architectural level may include several different aspects [11]. Based on our experience and the literature review, the following sub-requirements are the most important ones of the automobile industry:

Req-M8.1. An architecture description language for automobile systems should provide a process support for evaluating the quality of the software architecture with respect to the system and function reliability [5, 10].

Req-M8.2. An architecture description language for automobile systems should provide a process support for evaluating the quality of the software architecture with respect to the system safety [5, 10] to avoid fatal car accidents.

Req-M8.3. An architecture description language for automobile systems should provide a process support for evaluating the quality of the software architecture with respect to real-time properties [5, 16].

Req-M8.4. An architecture description language for automobile systems should provide a process support for evaluating the quality of the software architecture with respect to reusability and portability [5] of the implemented software.

Req-M8.5. An architecture description language for automobile systems should provide a process support for identifying harmful feature interactions [5].

With respect to the supplier management and software acquisition management (Req-M9): the architecture description language should be able to

define clear requirements and clear component interfaces for the subcontractors and component/software suppliers [5]. As a result, the following two sub-requirements have to be considered:

> **Req-M9.1.** An architecture description language for automobile systems should provide process support within the supplier management and software acquisition management to create clear and precise requirements for components to be developed outside the companies development project [5].

> **Req-M9.2.** An architecture description language for automobile systems should provide process support within the supplier management and software acquisition management to create clear and precise interfaces for components to be developed outside the companies development project [5].

2.3 Tool Related Requirements

Due to the complexity of automobile software, tools are really important in the automobile industry. Over the years, several tools have become de-facto standards in the development of automobile software. These tools include Telelogic/IBM DOORS for requirements management and Matlab Simulink and Mathworks State Flow for designing the system. An architecture descriptions language for automobile systems should be able to interface with these legacy tools and consequently the following requirements are added for the tool support:

> **Req-T1.** A tool that is associated with an architecture description language for automobile systems should provide an upward interface to the requirement management tools (e.g. Telelogic/IBM DOORS) that are used to model software for automobile systems [10].

> **Req-T2.** A tool that is associated with an architecture description language for automobile systems should provide a downward interface to the design tools (Matlab Simulink and Mathworks State Flow) that are used to model software for automobile systems [2, 16].

The tool that is used for modeling and specifying software for an automobile system in the architecture description language should furthermore fulfill the following requirements:

> **Req-T3.** A tool that is associated with an architecture description language for automobile systems should provide an easy-to-use graphical user interface for the graphical representation of the models [16].

Req-T4. A tool that is associated with an architecture description language for automobile systems should be extendable and tolerable to provide support for additional feature that are required by the company that uses this tool [16].

Req-T5. A tool that is associated with an architecture description language for automobile systems should be able to automate certain task that are required be the associated methods [16]. The relevant tasks are defined in the requirements Req-M1-9.

3 Evaluation of the AADL Modeling Language

In this section, to the best of our knowledge, we evaluate AADL against the requirements of Section 2.

3.1 Language Related Requirements

The fulfillment of the language related requirements of Section 2.1 are indicated in Table 1.

Req-L3, Req-L4, and Req-L6. AADL supports components, ports, channels (called connections in AADL), bus systems, sensors, and actuators as first class modeling concepts. The AADL sample model in Listing 1 is part of an Adaptive Cruise Controller originally developed in [4]. It specifies hardware and software, defines connections between ports, and deploys a process onto an processor.

Req-L5. AADL supports declaration and implementation of component types as well as component hierarchies. See Listing 2 for an example that defines scheduling protocols of the process ConsoleProcessor and also defines the declaration and implementation of the ProductionCell system component. Moreover, it does also specify that ProductionCell inherits from the Base component. The components are parts of a Production Cell system originally defined in [3].

Req-L7 and Req-L8. AADL with its behavior annex, supports the definition of component behavior by defining an abstract state machine that communicates with the surrounding system through ports. See Listing 3 for an example.

3.2 Method Related Requirements

The fulfillment of the language related requirements of Section 2.2 are indicated in Table 2.

Table 1: The AADL Support for the Language Related Requirements.

Requirement	**Supported**
Req-L1	No, functional design, analysis architecture, and
Req-L1.1 - L1.2	operational architecture were not intended to be included
Req-L2	in AADL, see [9].
Req-L3	Yes, AADL supports the Logical Structure of the specification, such as:
Req-L3.1 - 3.3	Networks of components, Ports, Channels (Connections).
Req-L3.4 - 3.5	Bus Systems, Sensors and Actuators.
Req-L3.6	Interfaces between subsystems and devices.
Req-L4	Yes, AADL supports the specification of hardware and software structure of the specification as well as its
Req-L5	deployment architecture, such as:
Req-L5.1	Scheduling of Components.
Req-L5.2	Component Type and Implementation.
Req-L5.3	Component Hierarchies.
Req-L6	Yes, AADL supports deployment between hardware and software components.
Req-L7	Yes, AADL supports the specification of logical data flows,
Req-L7.1 - L7.2	execution in different modes, and event triggered
Req-L8	architecture. However, time triggered architecture is not
Req-L8.1 - L8.2	supported.

Table 2: The AADL Support for the Method Related Requirements.

Requirement	**Supported**
Req-M1	Yes, the Eclipse plug-in OSATE supports consistency checks between architectural views and diagrams.
Req-M2	No, model-based development (such as source code generation) is currently not fully supported by AADL.
Req-M3	No, requirement tracing is currently not supported by AADL.
Req-M4	Yes, modular construction is supported. However, modular verification is not supported.
Req-M5	Yes, product line engineering and variability management is supported by software modes.
Req-M6	No, cost estimation is currently not fully supported by AADL. Only basic examples are provided in models that are associated with the tools OSATE [7].
Req-M7	No, change management is currently not supported by AADL.
Req-M8 Req-M8.1 - L8.2	Yes, the quality of the software can be evaluated with the Error Annex.
Req-M8.3	Yes, the Eclipse plug-in OSATE provides the possibility to perform scheduling analysis.
Req-M8.4	No, evaluation of the quality with respect to reusability and portability is currently not supported by AADL.
Req-M8.5	No, process support for identifying harmful feature interaction is currently not supported by AADL.
Req-M9 Req-M9.1 - M9.2	Yes, the creation of component interfaces is supported by AADL. However, supplier management software acquisition is not supported.

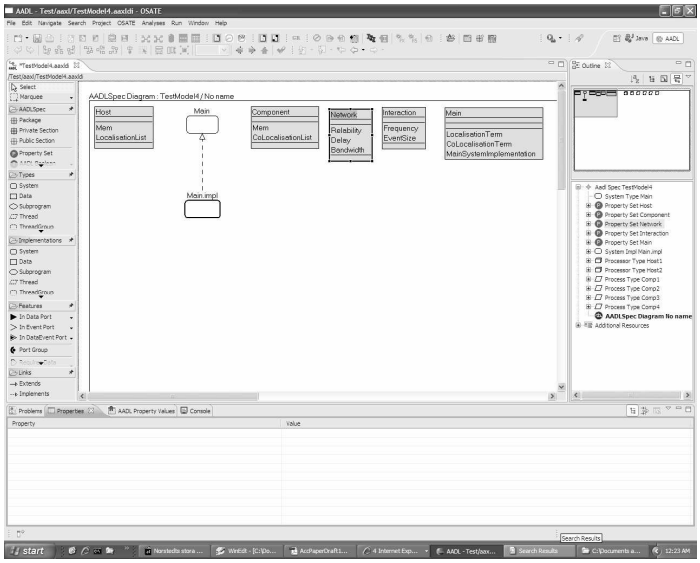

Figure 1: The OSATE Graphical Environment (Req-M1 and Req-T3).

Req-M1. In the Eclipse plug-in OSATE, it is possible to support consistency checks between architectural views and diagrams [7]. See Figure 1.

Req-M4. Modular construction is supported in AADL [9]. See Listing 2. However, modular verification is not supported.

Req-M5. Product line engineering and variability management is supported by AADL. See Listing 4 for an example that defines the subcomponents and the connections between the components, based on the component modes. Furthermore, [20] describes an approach to generate fault trees for product lines based on an AADL product line model.

Req-M8. The AADL Error Annex [8] is an extension to AADL with which the quality (probabilistic quality attributes [12]) of the software can be evaluated. Listing 5 shows an example of an abstract state machine modeling the console of an instrument panel. To analyze an AADL error model there are currently two approaches available. The first approach automatically translates an error model into a standard fault tree [15]. The second approach generates Generalized Stochastic

Table 3: The AADL Support for the Tool Related Requirements.

Requirement	Supported
Req-T1	No, currently there is no upward interface to model software requirement management tools.
Req-T2	No, currently there is no downward interface to design tools.
Req-T3	Yes, the Eclipse plug-in OSATE provides a graphical user interface for AADL.
Req-T4	Yes, the Eclipse plug-in OSATE provides a Java class library for developing plug-ins.
Req-T5	No, automatic code generation is currently not supported by AADL. However, automatic fault tree generation and evaluation model (GSPNs) generation is currently supported for the error annex, see [15] and [19].

Petri Nets (GSPNs) from error model specifications and uses existing GSPN tools for quantitative analysis [19].

Req-M9. The creation of component interfaces is supported by AADL. However, supplier management software acquisition is not supported.

3.3 Tool Related Requirements

The fulfillment of the language related requirements of Section 2.3 are indicated in Table 3.

Req-T3. The Eclipse plug-in OSATE (please see Figure 1) provide a rich set of functions, among them a graphical editor for graphical development.

Req-T4. Eclipse provides a class library to extend the environment with graphical textures. The Eclipse plug-in OSATE further extends the library with AADL specific classes. See Listing 6.

Req-T5. Automatic fault tree generation and GSPN generation is currently supported for the Error Annex, see [15] and [19].

4 Conclusions

The evaluation has provided evidence that AADL indeed is a language suitable for modeling software in automobile systems. It supports the architecture modeling at the right level of abstraction. The language is well

designed and contains all relevant aspects. Especially, the tool provides a good capability for graphical modeling and allows to evaluate the created models.

For future work, better support for modeling functional architectures and upward tracing to requirements management tools like Telelogic/IBM DOORS and downward tracing to low-level design tools like Matlab Simulink and Mathworks State Flow would be important. Furthermore, the methodological support could be improved.

References

[1] J. Arlow and I. Neustadt. *UML 2 and the Unified Process: Practical Object-Oriented Analysis and Design.* Addison-Wesley Professional, second edition edition, 2005.

[2] A. Bauer, M. Broy, J. Romberg, B. Schätz, P. Braun, U. Freund, N. Mata, R. Sandner, and D. Ziegenbein. AutoMoDe notations, methods, and tools for model-based development of automotive software. Technical Report 2005-01-1281, Institut für Informatik, Technische Universität München, 2005.

[3] S. Björnander, L. Grunsk, and K. Lundqvist. Timed Simulation of Extended AADL-Based Architecture Specifications with Timed Abstract State Machines. Submitted Draft, 2009.

[4] S. Björnander and L. Grunske. Modeling an Adaptive Cruise Controller in the Architecture Analysis and Design Language: A Case Study. Technical Report C4-01 TR M51, Faculty of Information and Communications Technologies, Swinburne University of Technology, Hawthorn, VIC 3122, Australia, 2008.

[5] M. Broy. Automotive software and systems engineering. In *MEMOCODE '05: Proceedings of the 2nd ACM/IEEE International Conference on Formal Methods and Models for Co-Design*, pages 143–149, Washington, DC, USA, 2005. IEEE Computer Society.

[6] M. Broy. Challenges in automotive software engineering. In *ICSE '06: Proceedings of the 28th international conference on Software engineering*, pages 33–42, New York, NY, USA, 2006. ACM.

[7] P. Feiler and A. Greenhouse. Plug-in Development for the Open Source AADL Tool Environment. Available from http://la.sei.cmu.edu/aadlinfosite/OSATEPlugInDevelopmentPresentationSerie.html#Topic19, 2005.

[8] P. Feiler and A. Rugina. Dependability modeling with the architecture analysis and design language (AADL). Technical Report CMU/SEI-2007-TN-043, Carnegie Mellon University, 2007.

[9] P. H. Feiler, D. P. Gluch, and J. J. Hudak. The Architecture Analysis and Design Language (AADL): An Introduction. Technical Report CMU/SEI-2006-TN-011, Society of Automotive Engineers, 2006.

[10] K. Grimm. Software technology in an automotive company: major challenges. In *Proceedings of the 25th International Conference on Software Engineering (ICSE-03)*, pages 498–505, Piscataway, NJ, May 3–10 2003. IEEE Computer Society.

[11] L. Grunske. Early quality prediction of component-based systems - A generic framework. *Journal of Systems and Software*, 80(5):678–686, 2007.

[12] L. Grunske. Specification patterns for probabilistic quality properties. In Robby, editor, *30th International Conference on Software Engineering (ICSE 2008), Leipzig, Germany, May 10-18, 2008*, pages 31–40. ACM, 2008.

[13] L. Grunske and J. Han. A Comparative Study into Architecture-Based Safety Evaluation Methodologies Using AADL's Error Annex and Failure Propagation Models. In *11th IEEE High Assurance Systems Engineering Symposium, HASE 2008*, pages 283–292. IEEE Computer Society, 2008.

[14] D. Harel. Statecharts: A Visual Formalism for Complex Systems. *Science of Computer Programming*, 1987.

[15] A. Joshi, S. Vestal, and P. Binns. Automatic Generation of Static Fault Trees from AADL Models. In *DSN Workshop on Architecting Dependable Systems*, Lecture Notes in Computer Science, page to appear. Springer, 2007.

[16] H. Lönn, T. Saxena, M. Nolin, and M. Törngren. FAR EAST: modeling an automotive software architecture using the EAST ADL. In *ICSE 2004 workshop on Software Engineering for Automotive Systems (SEAS)*. IEE, May 2004.

[17] R. Miles and K. Hamilton. *Learning UML 2.0*. O'Reilly Media, 2006.

[18] D. Pilone and N. Pitman. *UML 2.0 in a Nutshell*. O'Reilly Media, second edition edition, 2005.

[19] A.-E. Rugina, K. Kanoun, and M. Kaâniche. A System Dependability Modeling Framework Using AADL and GSPNs. In *Architecting*

Dependable Systems IV, volume 4615 of *LNCS*, pages 14–38. Springer, 2007.

[20] H. Sun, M. Hauptman, and R. R. Lutz. Integrating Product-Line Fault Tree Analysis into AADL Models. In *Tenth IEEE Int. Symp. on High Assurance Systems Engineering (HASE 2007)*, pages 15–22. IEEE Computer Society, 2007.

[21] S. Vestal. Formal verification of the metaH executive using linear hybrid automata. In *Proceedings of the Sixth IEEE Real-Time Technology and Applications Symposium (RTAS '00)*, pages 134–144, Washington - Brussels - Tokyo, June 2000. IEEE.

A Code Listings and Examples

The code listing are extract of three case studies: an steam boiler system [13], a adaptive cruise controller [4] and a industrial production cell [3]. To get an overview of these case studies we refer to the original literature [3, 4, 13].

Listing 1 Hardware Communication and Deployment (Req-L3, Req-L4, and Req-L6).

```
-- Req-L4. Specification of the Hardware.
processor ControllerProcessor
  features
    -- Req L3.4. The memory is connected to the bus.
    memoryBus: requires bus access MemoryBus;
end ControllerProcessor;

-- Req L3.5. The device establish a way to
-- communicate with the surrounding environment.
device PanelDevice
  features
    -- Req-L3.2. These ports are used to communication
    -- with other components.
    outToggle: out event port;
end PanelDevice;

system implementation ConsoleSystem.impl
  subcomponents
    -- Req-M4. Modular Construction.
    -- Req-L3.1. These subcomponents are forms
    -- a network of communicating components.
    consoleProcess: process ConsoleProcess.impl;
    consoleProcessor: processor ConsoleProcessor.impl;
    panelDevice: device PanelDevice;
  connections
    -- Req-L3.3 and Req-L3.6. These connections (also known as
    -- channels) establish communications between components and
    -- between devices and components.
    event port panelDevice.outToggle -> outToggle;
    data port consoleProcess.outSpeed -> outData;
    event port panelDevice.outPlus -> consoleProcess.inPlus;
    event port panelDevice.outMinus -> consoleProcess.inMinus;
  properties
    -- Req-L.6. The process (software) is
    -- connected to the processor (hardware).
    Actual_Processor_Binding => reference consoleprocessor
                                applies to consoleProcess;
end ConsoleSystem.impl;
```

Listing 2 Software Structure (Req-L5 and Req-M4).

```
-- Req-L5. Specification of the Software.
-- Req-L5.1. Specification of the scheduling
-- protocols.
processor implementation
ConsoleProcessor.impl
  properties
    Scheduling_Protocol => (RMS, EDF, Sporadicserver, SlackServer, ARINC653);
end ConsoleProcessor.impl;

-- Req-L5.2 and Req-M4. Component type.
system Base
  features
    InBlockReady: out event port;
    OutBlockLoaded: out event port;
end Base;

-- Req-L5.3. Extension render it possible
-- to establish component hierarchies.
system ProductionCell extends Base
end ProductionCell;

-- Req-L5.2 and Req-M4. Implementation of component type.
system implementation
ProductionCell.impl
  subcomponents
    Loader :system RobotArm;
    FeedBelt :system ConveyorBelt;
end ProductionCell.impl;
```

Listing 3 Behavior Annex (Req-L7 and Req-L8).

```
-- Req-L8. The annex defines the component
-- behavior as an abstract state machine.
-- Req-L7. The abstract state machine
-- communicates with the surrounding system by
-- sending and receiving data thought ports.
system ConveyorBelt extends Base
  annex ConveyorBelt
  {**
    states
      BeltEmpty :initial state;
      BlockAtBeginning, BlockAtEnd :state;
    transitions
      BeltEmpty -[InBlockLoaded?]-> BlockAtBeginning;
      BlockAtBeginning -[OutBlockReady?]-> BlockAtEnd
                        {InBlockPicked!; OutBlockLoaded!;}
      BlockAtEnd -[OutBlockPicked?]-> BeltEmpty {InBlockReady!;}
  **};
end ConveyorBelt;
```

Listing 4 Software Modes (Req-M5 and Req-M9).

```
-- Req-M5 and Req-M9. Product Line Engineering
-- and Variability Management are supported by
-- Software Modes.
process ControllerProcess.impl
  subcomponents
    radarSampleThread: thread RadarSampleThread in modes
                       (RADAR_UP_CONSOLE_UP, RADAR_UP_CONSOLE_DOWN);
  connections
    data port inRadarData -> radarSampleThread.inData in modes
                             (RADAR_UP_CONSOLE_UP, RADAR_UP_CONSOLE_DOWN);
  modes
    RADAR_UP_CONSOLE_UP: initial mode ;
    RADAR_UP_CONSOLE_DOWN: mode ;
    RADAR_DOWN_CONSOLE_UP: mode ;
    RADAR_DOWN_CONSOLE_DOWN: mode ;
end ControllerProcess.impl;
```

Listing 5 Error Annex (Req-M8).

```
process ConsoleProcess
  features
    inPlus: in event port;
    inMinus: in event port;
    outSpeed: out data port AccTypes::Stream;
  -- Req-M8. The AADL Error Annex is an extension to AADL with
  -- which the quality of the software can be enhanced.
  annex error_specification
  {**
    error model ConsoleError
      features
        Ok: initial error state;
        PanelDown, DisplayDown, PanelDisplayDown: error state;
        PanelFailure, DisplayFailure : error event;
    end ConsoleError;

    error model implementation ConsoleError.impl
      transitions
        Ok-[PanelFailure]->PanelDown;
        DisplayDown-[PanelFailure]->PanelDisplayDown;
        Ok-[DisplayFailure]->DisplayDown;
        PanelDown-[DisplayFailure]->PanelDisplayDown;
      properties
        Occurance => fixed 0.01 applies to PanelFailure;
        Occurance => fixed 0.1 applies to DisplayFailure;
    end ConsoleError.impl;
  **};
end ConsoleProcess;
```

Listing 6 OSATE Plug-In (Req-T4).

```
import java.util.*; import
edu.cmu.sei.osate.ui.actions.*;

public final class Plugin extends AaxlReadOnlyActionAsJob
{
  public void doAaxlAction(IProgressMonitor monitor, AObject aRoot)
  {
    CMap componentMap = new OrderedMap();
    Iterator iterator = aRoot.getChildren().iterator();

    while (iterator.hasNext())
    {
      AObject aObject = (AObject) iterator.next();

      if (aObject instanceof ComponentType)
      {
        ComponentType componentType = (ComponentType) aObject;
        String parentName = componentType.getExtendedQualifiedName();

        CSet subcomponentSet = new OrderedSet();
        CSet connectionSet = new OrderedSet();
        CMap annexMap = new OrderedMap();

        traverse(componentType.getChildren().iterator(),
                 subcomponentSet, connectionSet, null, null, annexMap);
        Component component = new Component(subcomponentSet,
                                connectionSet, inPortSet, null, null);
        componentMap.put(componentType.getName(), component);
      }
    }
  }
}
```