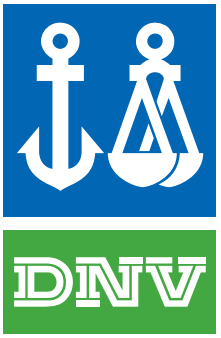
ModelME! Technical Report 2011-03

An AADL-Based SysML Profile for Architecture Level Systems Engineering: Approach, Metamodels, and Experiments

Authored by:
Razieh Behjati, Tao Yue, Shiva Nejati, Lionel Briand and Bran Selic

Date: February 12, 2011



[simula . research laboratory]
- by thinking constantly about it

Executive Summary

Recent years have seen a proliferation of languages for describing embedded control systems. Some of these languages have emerged from domain-specific frameworks, and some are adaptions or extensions of more general-purpose languages. In this paper, we focus on two widely-used standard languages: the Architecture Analysis and Design Language (AADL) and the Systems Modeling Language (SysML). AADL was born as an avionics-focused domain-specific language and later on has been revised to represent and support a more general category of embedded real-time systems. SysML is an extension of the Unified Modeling Language (UML) intended to support modeling system engineering applications. We propose the ExSAM profile that extends SysML by adding AADL concepts to it with the goal of exploiting the key advantages of both languages in a seamless way. We describe this profile through several examples and compare it with existing alternatives. We have implemented ExSAM using IBM Rational Rhapsody and evaluated its completeness and usefulness through two case studies.

Contents

List of Figures

List of Tables

1 Introduction

Many control applications are systems-of-systems, integrating various mechanical, electronic, and information technology systems. The design of these systems, classified as Integrated Control Systems (ICSs), depends more and more on effective solutions that can address heterogeneity and interplay of physical and software elements. In particular, design languages used for specifying ICSs should incorporate, in a consistent manner, essential concepts from multiple disciplines, such as mechanical, electronic, and software engineering.

Model-Driven Engineering (MDE) approaches to system development have been adopted in diverse domains, in particular, ICSs. This is because the use of models has shown to be promising in addressing the above issues, as well as, in handling the increasing complexity of ICSs, reducing their cost of construction, and supporting efficient maintenance and evolution [15, 13].

A number of modeling languages have been proposed to help engineers from different disciplines to communicate and compare their perspectives, to reason about properties of heterogeneous ICSs, and to develop optimized system-level solutions by assessing multidisciplinary design trade-offs. Of particular interest in our work are *standardized* languages, which are generally preferred by industry because they can reduce training costs, and reduce the risk of vendor lock in. In recent years, a number of modelling languages targeting ICSs have been standardized, but, to the best of our knowledge, none of them provide the full range required to deal effectively with the kinds of ICSs that we have encountered. Some of them, such as Systems Modeling Language (SysML) [6], focus on the "big picture" architectural views, whereas others, such as Architecture Analysis and Design Language (AADL) [12] address the more detailed platform-oriented and physical aspects of such systems.

Consequently, we investigated the possibility of combining these two languages, SysML and AADL, since they are both widely used in industry with adequate tool support. However, since it is generally preferable to work with a single internally consistent formalism, we chose to merge the two into a single unified language, which, moreover, takes advantage of existing commercial tools.

Figure 1 illustrates the relationship between the capabilities of the two languages, showing that they are mutually complementary. SysML, an extension of Unified Modeling Language (UML), version 2, is a standardized language for systems engineering. In addition to retaining much of UML, it also provides specialized support for requirements engineering, traceability, and precise modeling of diverse physical phenomena. AADL, on the other hand, is oriented towards the modeling of real-time embedded systems and includes a comprehensive catalogue of hardware and software elements common in such systems and their characteristics, allowing relatively precise and dependable analyses of different system properties such as performance, timing, or power consumption.

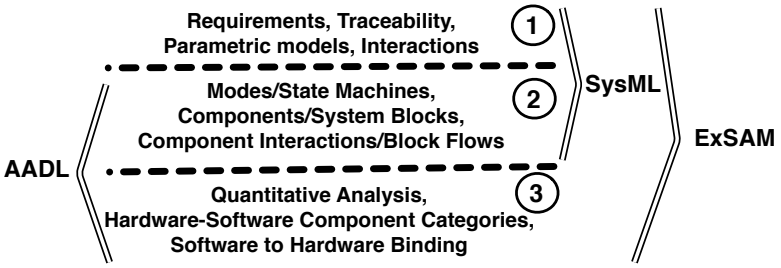


Figure 1: The relationship between SysML, AADL, and the combined profile ExSAM.

Although both SysML and AADL have extension mechanisms, in unifying the two languages, we chose to extend SysML. This is because its extension mechanism (profiles) comes from UML, which is much more widespread and better supported by tools, and which is also more powerful compared to that of AADL (annexes). We propose a SysML profile, Extended SysML for Architecture Analysis Modeling (ExSAM), that combines all the modeling capabilities of AADL and SysML (Figure 1). Specifically, ExSAM extends SysML to cover all AADL concepts, by expressing these concepts using SysML constructs.

Since SysML and AADL both target modeling embedded software systems, there are conceptual overlaps between them. Figure 1, level 2, represents an overview of the overlapping constructs between these two languages. These constructs seemingly specify the same concepts, however they tend to have different meanings, usages, or design rationales. To map the overlapping constructs, one needs to resolve inconsistencies or augment either side of the mappings with constraints to account for semantic differences. For example, realization relation is used both in AADL and SysML. In AADL, realization allows the reuse of attributes of physical components, whereas in SysML, the physical attributes cannot be manipulated through realization relations because such attributes appear at the instance-level while realization is defined at the block-level (UML class-level). Hence, to model AADL realization, SysML realization must be extended and constrained (Section 3.1). In ExSAM, we have made several such alignments to fully embed AADL constructs into SysML.

To evaluate the completeness and usefulness of ExSAM for ICSs, we have applied it to two case studies: One is a benchmark case study and the other is a large-scale, industrial case study. The first case study showed that ExSAM was adequate to capture all the used AADL concepts. The second case study showed that ExSAM was sufficient to satisfy the modeling needs of our industrial partner, while AADL and SysML alone were not.

The reminder of the paper is organized as follows. We provide a brief introduction to SysML and AADL in Section 2. In Section 3, we present ExSAM and illustrate its use. We describe the tool support and an evaluation of ExSAM through two case studies in Section 4. We compare ExSAM with other alternatives in Section 5. Finally, we conclude the paper in Section 6.

2 Background

In this section, we provide a brief introduction to SysML (Section 2.1) and to AADL (Section 2.2).

2.1 SysML

SysML is a modeling language with a graphical syntax developed and standardized by the Object Management Group (OMG). It was designed to, among other things, capture the interactions of software with physical entities, and is widely used for systems engineering [14]. Compared to UML 2, SysML adds support for systems engineering (e.g. through requirements engineering, and quantitative analysis of physical aspects of the system), while removing many of UML's software-centric constructs. In ExSAM, we have particularly benefited from the following SysML-specific constructs:

SysML blocks. Blocks are modular units of system descriptions in SysML and are generalizations of the UML class concept. The notion of block in SysML enables better expression of Systems engineering semantics compared to UML, and particularly, reduces the UML bias towards software. Blocks and their relationships are visualized in SyML block definition diagrams (bdds). The definition of a block in SysML can be further detailed by specifying its parts; ports, specifying its interaction points; and connectors, specifying the connections among its parts and ports. This information is visualized using SysML internal block diagrams (ibds).

SysML flows ports and SysML item flows. The SysML flow port concept extends the UML port concept and is intended to describe an interaction point for a block through which the block interacts with its environment [16]. The rationale for having flow ports is that some interactions of a block may not involve message passing or service calls, but rather phenomena such as continuous or discrete energy flows. In particular, a block can have interaction points over which it supplies or is supplied with electric power, fuel, air, or any other kind of streaming input or output. SysML FlowPorts are typed by FlowSpecifications, which specify the types of flows that can pass through them. The SysML ItemFlow concept extends the UML InformationFlow concept, which has the ability to be explicitly associated with Connectors via the realizingConnector dependency [7]. This capability allows us to describe the detailed implementation of an item flow through connectors and flows realizing it.

2.2 AADL

AADL, is a modeling language originally designed for and used in avionics. It was standardized by the Society of Automotive Engineers (SAE) and has been used to describe software execution platforms (e.g., processors, memory, buses, devices) as well as the physical environments of embedded software systems (e.g., electronic and mechanical parts interacting with ICSs). In addition, AADL supports the early prediction and analysis of critical system qualities – such as performance, schedulability, and reliability [12].

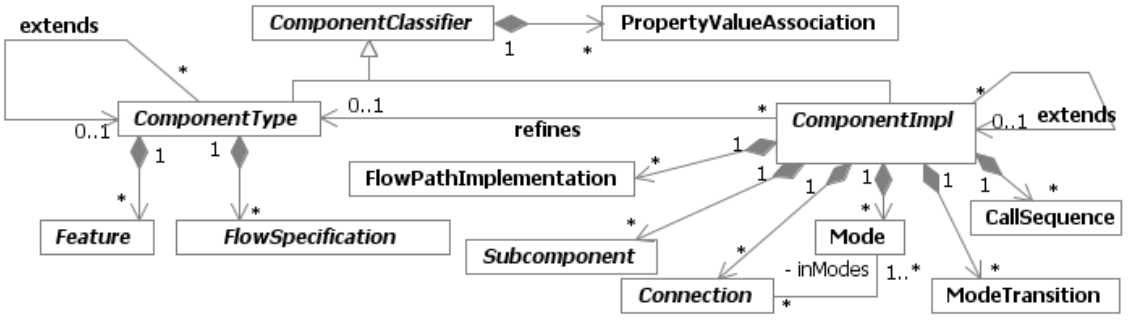


Figure 2: The core AADL concepts.

In this section, we provide an overview of AADL using a domain model representing the main AADL concepts and their relationships. We have developed this domain model based on the AADL reference manual [12]. A fragment of this domain model capturing the main AADL concepts is shown in Figure 2. Below, we discuss the concepts shown in Figure 2 and illustrate them using the AADL example in Figure 3.

```
1  system redundant_pattern
2  features
3    indata: in data port;
4    outdata: out data port;
5    reinitialize: in event port;
6  flows
7    primary_flow: flow path indata -> outdata;
8  properties
9    Period => 20ms;
10 end redundant_pattern;

11 system implementation redundant_pattern.primary_backup
12 subcomponents
13   primary_system: system principal_functions;
14   backup_system: system principal_functions;
15   observer: process observer_pattern;
16 connections
17   in_nom:data port indata -> primary_system.indata in modes nominal;
18   in_fail: data port indata -> backup_system.indata in modes backup, reinit;
19   out_nom:data port primary_system.outdata -> outdata in modes nominal;
20   out_bk: data port backup_system.outdata -> outdata in modes backup, reinit;
21   indata_p: data port primary_system.outdata -≫ observer.indata_P;
22   indata_b: data port backup_system.outdata -≫ observer.indata_B;
23 flows
24   primary_flow: flow path indata -> in_nom -> primary_system.flow1 -> out_nom -> outdata;
25 modes
26   nominal: initial mode;
27   backup: mode;
28   reinit: mode;
29   nominal -[observer.primary_fail]-> backup;
30   backup -[reinitialize]-> reinit;
31   reinit -[observer.primary_ok] -> nominal;
32 end redundant_pattern.primary_backup;
```

Figure 3: An example AADL model.

AADL provides abstractions for describing a system in terms of its components, their interfaces, and the connectors between the interfaces. AADL provides two mechanism for declaring components: using the *component type* construct, which specifies a component by describing only its interface, or using the *component implementation* construct, which specifies a component by describing its internal structure. ComponentType and ComponentImpl in Figure 2, respectively, repre-

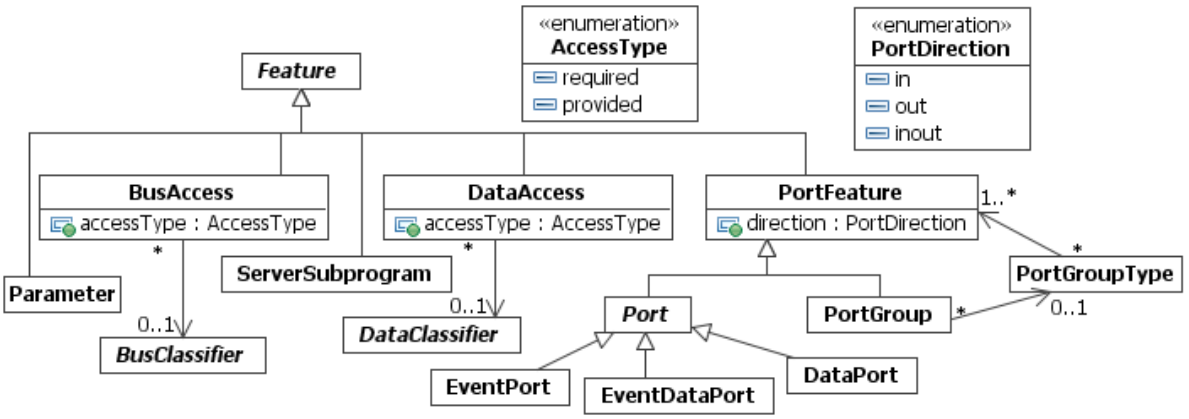


Figure 4: Features of a component type.

sent these concepts. In Figure 3, lines 1-10 specify a component type named `redundant_pattern`, and lines 11-32 specify a component implementation named `redundant_pattern.primary_backup`. In AADL, components are declared as a type or implementation within a particular *component category* [12]. These core concepts are elaborated further below.

Component categories. AADL provides ten component categories to define the runtime nature of software, hardware, and composite components. Software component categories are *data, subprogram, thread, thread group* and *process*. Hardware component categories are *memory, processor, bus* and *device*. A special component category named *system* indicates either a system consisting of several software components only, or a system consisting of both hardware and software components. In Figure 3, the component type `redundant_pattern` (Line 1) and the component implementation `redundant_pattern.primary_backup` (Line 11) both belong to the component category `system`. Fragments of the domain model for different component categories and their relationships are available in Appendix A.

Component types. A component type specifies the externally visible characteristics of a component in terms of features, flow specifications, and property value associations (Figure 2).

Features. Different types of features are used to specify the interfaces of a component. Ports and port groups (collections of ports or other port groups) are hardware features that represent the directional exchange of data, events, or both. Data/bus accesses are specific features that make data and bus subcomponents accessible throughout the encompassing component. For data subcomponents, the capability of declaring data accesses supports modeling of shared access to a common data. For bus subcomponents, this capability models the connectivity of execution platform components. Subprograms, server subprograms, and their parameters are used to specify the software features of a component. Figure 4 shows different types of features and their relationships to other concepts. In Figure 3, lines 3-5 specify the `features` of the component type `redundant_pattern`.

Flow specifications. In a component type, flows are directional and designate a source or a sink, which are features of a component, or a flow path, which represents a flow through a

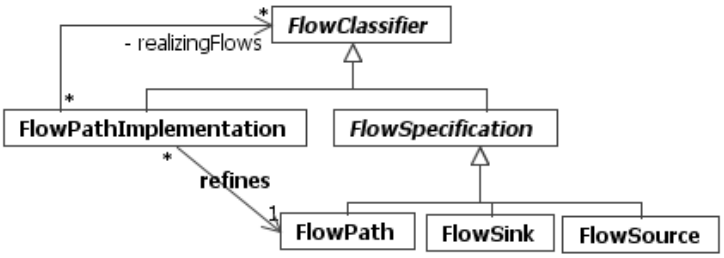


Figure 5: Flow specification and flow path implementation.

component from one feature to another. Figure 5 shows different types of flow specifications. Line 7 in Figure 3 declares a flow path connecting `indata` to `outdata`, which are declared as features of the component type.

Property value associations. Property value associations are used to assign a value or a list of values to properties. Note that these properties are defined as part of the component categories. Line 9 in Figure 3 shows a property value association.

Component implementations. As shown in Figure 2, component implementations refine component types by specifying subcomponents, interactions (connections and call sequences), flow path implementations, property value associations and modes. In Figure 3, `redundant_pattern.primary_backup` implements the component type `redundant_pattern`.

Subcomponents and interactions. Subcomponents in a component implementation can be other component types or component implementations. Connections and call sequences are used to describe the interactions among subcomponents. In Figure 3, subcomponents of the component implementation `redundant_pattern.primary_backup` are listed in lines 13-15. Connections among these subcomponents are declared in lines 17-22.

Flow path implementations. A flow path implementation describes a sequence of paths through and connections among subcomponents within a component implementation. This path is a realization of the corresponding flow path declared in the component type declaration [12]. This relationship is depicted in Figure 5 using the refines associations relation between FlowPathImplementation and FlowPathSpecification. In Figure 3, the flow path implementation declared in line 24 refines the flow path specification declared in line 7.

Modes. An AADL component implementation declaration may contain the declaration of modes and mode transitions. Modes represent alternative operational states of a system or component [12]. Transition from one mode to another is triggered by events. In Figure 3, lines 26-28 declare three modes of `redundant_pattern.primary_backup`. In addition, lines 29-31 show the mode transitions among these three modes.

In AADL, a mode is an explicit configuration of its contained elements, e.g., subcomponents and connections. The in modes clause in the declaration of a component implementation is used to specify the active elements in each mode. The declarations of the connections in lines 17-20, in Figure 3, also contain the specification of the modes these connections are active in.

3 Profile Description

In this section we describe how we extend SysML using a profile, ExSAM, with the purpose of combining architecture design and analysis concepts of AADL with the system modeling concepts of SysML. This profile is resulted from our practice of mapping AADL to SysML and based on our observations of SysML limitations in addressing important AADL concepts discussed in Section 2.2. In this section, we briefly describe the most important features of ExSAM. Here, we first describe the mapping for AADL components in Section 3.1. In Section 3.2 we describe the mapping for AADL component extension and generalizations. Section 3.3 is dedicated to the mapping of AADL modes. In Section 3.4 we present the mapping for AADL bindings. Finally, in Section 3.5 we explain how we can use ExSAM to benefit from AADL analysis capabilities for SysML models.

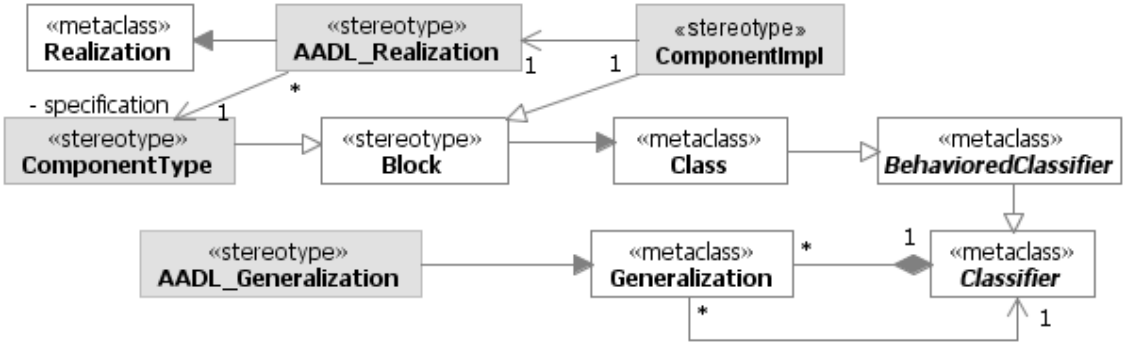


Figure 6: Metamodel for mapping AADL concepts: component type and component implementation.

3.1 Mapping component types and component implementations

Recall from Section 2.2 that in AADL, component types and component implementations describe a component, respectively, through its externally visible interface and its internal structure. In ExSAM, we use SysML blocks to model both AADL component types and AADL component implementations. To distinguish between component types and component implementations, we use two newly defined stereotypes, «ComponentType» and «ComponentImpl», both extending SysML block as shown in Figure 6[1]. SysML blocks extend UML classes, and are chosen to model components because they can describe both structural and behavioral features of a system or component. In addition, using blocks for modeling components allows us to easily use other SysML constructs (e.g. parts and ports) to model AADL constructs (e.g. subcomponents and ports) associated with a component in a consistent and straightforward manner.

As mentioned in Section 2.2, AADL provides ten different component categories. In ExSAM, we dedicate to each AADL component category a stereotype with a set of attributes representing the properties of the corresponding component category. AADL property value associations are then

[1] In Figures 6, 7, and 10, the stereotypes in gray are introduced by us, and the rest are from SysML. Also, in these two figures, a solid line ending in a filled triangle shows UML extension and a solid line ending in a hollow triangle shows UML generalization.

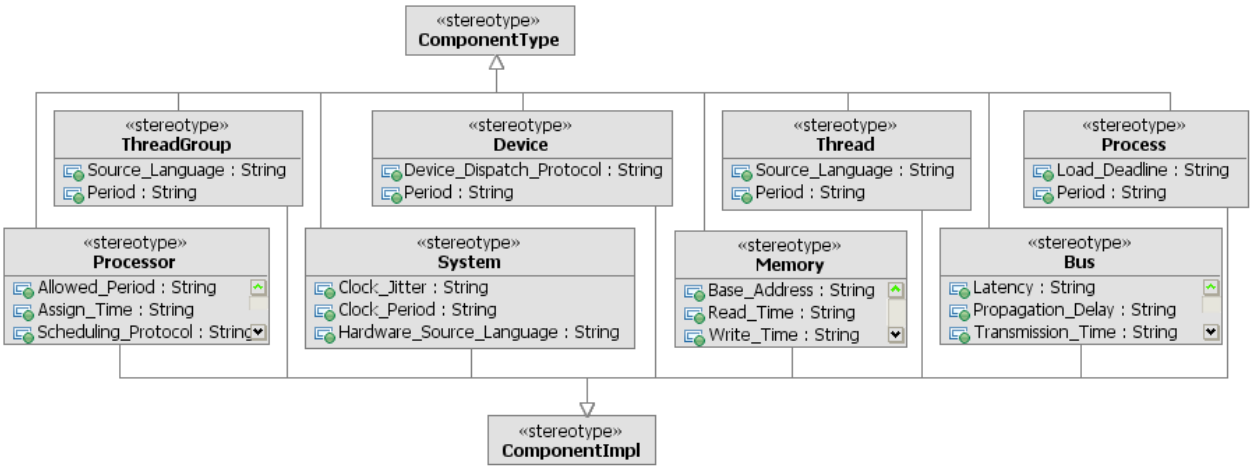


Figure 7: Category identifier stereotypes and their attributes.

automatically mapped to the values assigned to the attributes of stereotypes applied to blocks. In the rest of this section, we refer to this set of stereotypes as *category identifier stereotypes*. In ExSAM, all category identifier stereotypes generalize «ComponentType» and «ComponentImpl».

Figure 7 shows the category identifier stereotypes and some of their attributes. We apply two stereotypes to each block: one stereotype specifies whether it is a component type («ComponentType») or a component implementation («ComponentImpl»), and the other specifies its component category. All the category identifier stereotypes, also, generalize UML Property and UML InstanceSpecification; this guarantees that we can use these stereotypes on parts appearing in internal block diagrams and instances, not only blocks in block definition diagrams.

As shown in Figure 2, a component implementation in AADL can realize and refine a component type specification by adding implementation details to it, namely, by specifying its subcomponents, connections, modes, properties, and flow paths. A realization relationship in AADL transfers all property value associations from the component type to the component implementation, and makes all features and flow specifications of the component type accessible from the component implementation. In ExSAM, we capture this refinement using a UML/SysML realization relationship between a block stereotyped by «ComponentImpl» and a block stereotyped by «ComponentType». However, the semantics of UML/SysML realization is different from that of AADL realization. For example, it does not support the transfer of property value associations from the block stereotyped by «ComponentType» to the block stereotyped by «ComponentImpl». In addition, for the realization to be meaningful in this context, the involved blocks should be stereotyped by the same category identifier stereotype. In ExSAM, we have specialized and constrained UML/SysML realization using «AADL_Realization» (Figure 6) to capture these detailed semantics. Each realization relationship in an ExSAM model, should, therefore, be stereotyped by «AADL_Realization» to represent an AADL realization.
Figure 8 shows an excerpt of the ExSAM model created for the AADL model in Figure 3. In this model, a block named `redundant_pattern`, stereotyped by «ComponentType» and «System», represents the AADL component type `redundant_pattern`. The value of the attribute `period`, defined

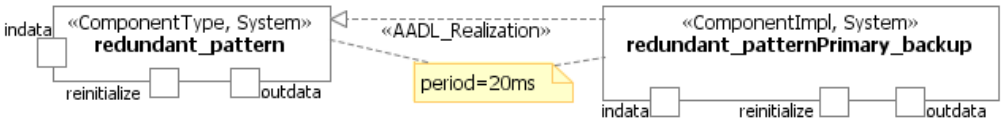


Figure 8: An fragment of the ExSAM model for the AADL model in Figure 3.

in «System», is set to `20ms` in this block. The other block, `redundant_patternPrimary_backup`, realizes `redundant_pattern`, and is stereotyped by «ComponentImpl» and «System». Note that the realization relation between the two blocks is also stereotyped by «AADL_Realization», which, as shown in Figure 8, ensures that in `redundant_patternPrimary_backup` the value of `period` is `20ms`.

As mentioned in Section 2.2 and shown in Figure 2, a component type declaration in AADL defines the interface of a component in terms of features and flow specifications. In ExSAM, we use attributes and operations of a SysML block to model AADL software features (e.g. subprograms and parameters), and ports (i.e. SysML FlowPorts and UML StandardPorts) to model AADL hardware features (e.g. ports). Specifically, for modeling an AADL port group in ExSAM, we use a port typed by a SysML FlowSpecification. In Figure 8, the ports (`indata`, `outdata`, and `reinitialize`) on the border of the two blocks `redundant_pattern` and `redundant_patternPrimary_backup` represent the features of the corresponding component type and component implementation.

In AADL, flow specifications are used to specify flow sources, flow sinks, and flow paths that connect flow sources to flow sinks. Since, flow sources and flow sinks are hardware features of a component, according to the mapping specified earlier in this section, they are mapped to ports in ExSAM. To model a flow path connecting a flow source to a flow sink, we use a SysML FlowItem connecting the two ports representing the flow source and the flow sink. For example, in the ExSAM model created for the AADL model in Figure 3, there is a FlowItem named `primary_flow` in the block named `redundant_pattern` connecting the ports representing `indata` and `outdata` (Figure 8).

Recall from Section 2.2 that in AADL, a component implementation specifies the internal structure of a component through its subcomponents and the connections among them. Subcomponents of an AADL component implementation are naturally mapped to parts of a SysML block in ExSAM. The connections among AADL subcomponents are then captured through SysML ports (owned by the parts or by the encompassing block) and connectors connecting them. SysML ibds are used to visualize this information. Figure 9 shows the ibd depicting the internal structure of the block `redundant_patternPrimary_backup` in the `nominal` mode. The part named `primary_system` represents a subcomponent of the component implementation `redundant_patternPrimary_backup` in Figure 3. As shown in this figure, several connectors are used to connect the ports of this part to the ports of the encompassing block.

As mentioned in Section 2.2, an AADL component implementation can declare flow path implementations (e.g. line 24 in Figure 3) to refine and implement the flow path specifications (e.g. line 7 in Figure 3) declared in the component type that it implements. A flow path implementation involves connections that specify a flow path starting from a flow source, passing through a number of subcomponents and their specified flows and finally reaching the flow sink. In ExSAM, an AADL flow path implementation is mapped to a set of connectors that are associated to the

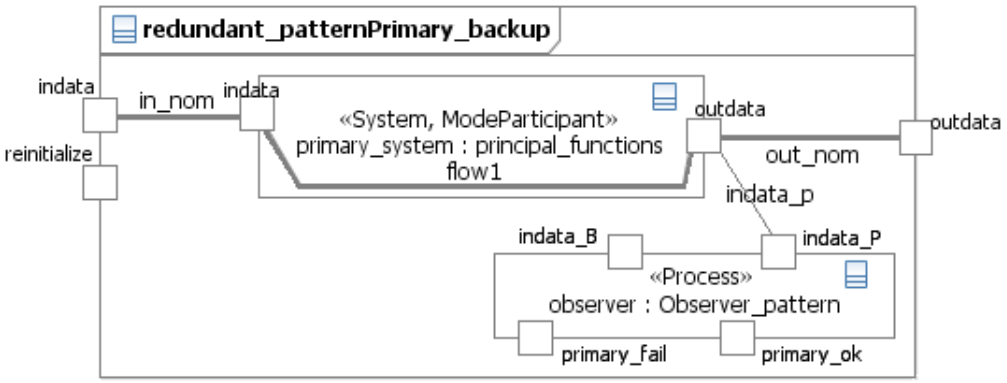


Figure 9: An ibd in the ExSAM model created for the AADL model in Figure 3. This ibd shows the internal structure of redundant_patternPrimary_backup in the nominal mode.

FlowItem representing the AADL flow path specification, which is implemented by the flow path implementation. In addition, for each flow path implementation we create an ibd in the ExSAM model to visualize these connectors and the way they connect parts and ports, as shown in Figure 9. The highlighted connectors in Figure 9, `in_nom`, `out_nom`, and `flow1`, are used to realize the `primary_flow` FlowItem depicted in Figure 8.
We use FlowItems to model flow path specifications because SysML FlowItem extends UML InformationFlow, which according to UML metamodel [7], can be realized using a set of connectors named realizingConnectors. In our mapping, the realizingConnectors of SysML FlowItems are used to model AADL flow path implementations.

A component implementation in AADL can also declare call sequences. In ExSAM, we use SysML activities or interactions to model call sequences.

3.2 Extension and generalization

As shown in Figure 2, in AADL, a component type can extend another component type. A component type inherits all the features, flow specifications and property value associations from its base component. Similarly, a component implementation can extend another component implementation. A component implementation inherits subcomponents, flow path implementations, call sequences, modes and property value associations from its base component implementation.

In an ExSAM model, we can use UML/SysML generalization to model the extension relationships between AADL components. However, the semantics of UML/SysML Generalization is different from that of AADL extension. For example, using UML/SysML Generalization we cannot capture the inheritance of property value associations. To address this semantic difference, in ExSAM, we define a new stereotype named «AADL_Generalization» that specializes UML/SysML Generalization and implements the semantics of AADL extension using an OCL constraint. Such an OCL constraint specifies that: 1) a block can extend another if they both have the same category identifier stereotype, 2) if the blocks are component types, the sub block must inherit from its supper block all the attributes, ports, item flows, and values assigned to the attributes of the category identifier stereotype applied to its super block, 3) if the blocks are component implementations, the

sub block must inherit from its super block all parts, item flows and their realizing connectors, interactions, modes, and values assigned to the attributes of the category identifier stereotype applied to its super block.

3.3 Modes

As mentioned in Section 2.2, a component implementation can operate in several modes. Each mode represents one configuration of the component. The component can transit from one mode to another in response to the occurrence of an event. In ExSAM, we use states to model modes and state machine transitions to capture mode transitions. Such a state machine is associated with the block - stereotyped by «ComponentImpl» - representing the component implementation with modal behavior. The association between «ComponentImpl» and StateMachine in Figure 10 shows this.

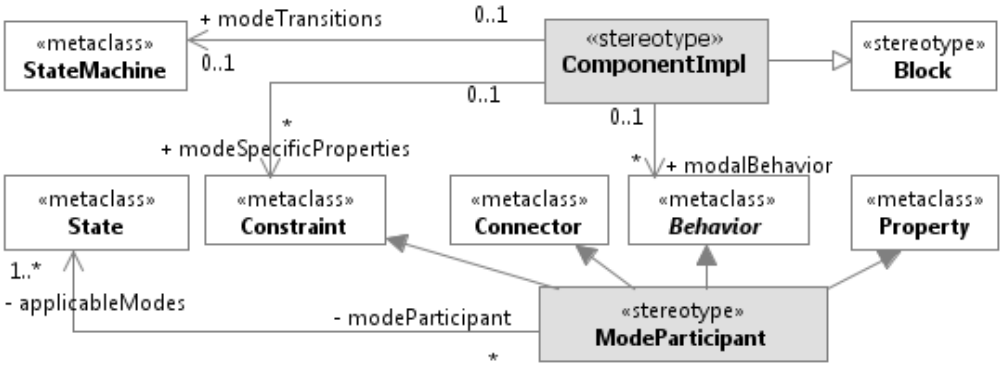


Figure 10: Metamodel for mapping AADL concepts: modes.

In addition, in AADL, the in modes clause is used to specify the active subcomponents and connections in each mode. Notice that a state machine can only model modes and the transitions among them, but have no way to link modes (states in state machine) to their own structure, behavior, and/or constraints. Therefore, to model the AADL in modes concept in ExSAM, we have introduced a stereotype named «ModeParticipant». As shown in Figure 10, the «ModeParticipant» stereotype has a relation to the UML State metaclass. This relation shows the set of modes that an element stereotyped by «ModeParticipant» can be active in. Notice that a «ModeParticipant» can be active in one or more modes. The state machine in Figure 11 shows the modes and mode transitions of the block `redundant_patternPrimary_backup` shown in Figure 8.

There are three different types of modes in AADL: (1) mode-specific structure of subcomponents and connections, which describes alternative configurations of active components and connections; (2) modal configurations of call sequences, which describe alternative behavioral interactions of subcomponents; and (3) mode-specific properties, which define alternative characteristics and behaviors of the components. To support these modeling capabilities in ExSAM, we use the «ModeParticipant» stereotype, which, as shown in Figure 10, extends the UML metaclasses Property,

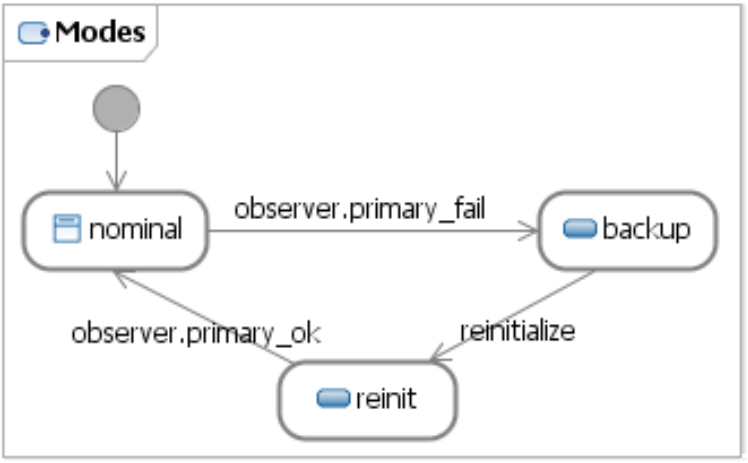


Figure 11: State machine for the mode transitions of redundant_patternPrimary_backup.

Connector, Behavior, and Constraint. Being able to model properties, including parts and ports, as well as connectors as mode participants allows us to precisely model the mode specific structure in each state. Similarly stereotyping behaviors, including interactions and activities as mode participants allows us to precisely specify AADL behaviors, e.g. call sequences, and associate them to the appropriate state.

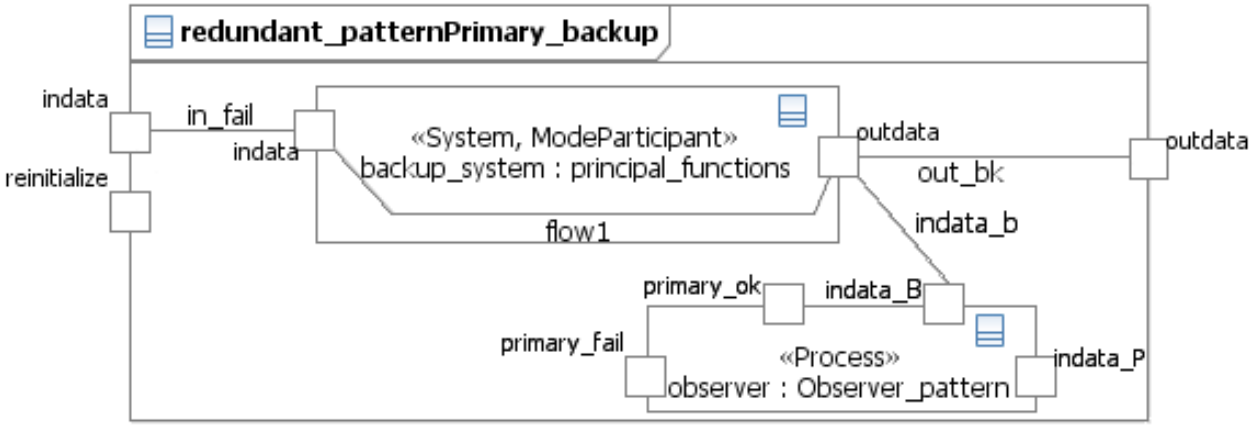


Figure 12: An ibd visualizing the part and connectors that are active in the backup mode.

Figure 9 shows an excerpt of the ExSAM model depicting an ibd for the internal structure of the component `redundant_pattern.primary_backup` (Figure 3) in the `nominal` mode. In this ibd the part named `primary_system` represents the subcomponent `primary_system` in the AADL model. Note that the other subcomponents are not shown in this ibd, since the purpose of this ibd is to visualize only the elements that are involved in the mode `nominal`. The part `primary_system` is stereotyped by «System» to show its component category. Figure 12 shows another ibd, which is used to visualize the active part and connectors (e.g. `backup_system`, `in_fail`, `out_bk`) in the `backup` mode. Table 1, lists model elements of Figures 9 and 12 that are stereotyped by «ModeParticipant». For each

Table 1: Mode participants and their active modes.

	indata	in_fail	backup_system	out_bk	outdata	in_nom	primary_system	out_nom
nominal	✓		✓		✓	✓	✓	✓
backup	✓	✓	✓	✓	✓		✓	
reinit	✓	✓	✓	✓	✓		✓	

element in the table, the modes in which that element is active are indicated using ✓. Note that this information is not shown in Figures 9 and 12 to minimize cluttering the figures.

Note that in the development of ExSAM, we were restricted by the design choices of SysML, including the fact that only a subset of UML constructs are imported into SysML. The most interesting UML concepts that are, in our opinion, missing in SysML are Collaboration and CollaborationUse, which can be used to capture AADL modes in UML. The solution we proposed in this section, however, uses the «ModeParticipant» stereotype and applies it to several metaclasses, mentioned above, to precisely convey the semantics of AADL modes in ExSAM models.

3.4 Mapping for bindings

An important aspect of AADL is the ability to model the deployment of software components to hardware components. In AADL it is done through value associations to specific properties of certain, deployable component categories. For example, in AADL to model that process `FlightDirector` is deployed to the processor named `Xeon_solo`, in the component `FlightDirector` we set the value of the property `bound_processor` to `Xeon_solo`.

In order to model this information in ExSAM, we use SysML allocation relationships. For example, in the example mentioned above, we add a dependency stereotyped by SysML «allocate» to the model, connecting the block representing `FlightDirector` to the block representing `Xeon_solo`. Note that this mapping is an exception to the general rule on mapping property value associations (Section 3.1), which is done through values assigned to the attributes of stereotypes. This is because allocation is an important piece of information in ICSs, and it is crucial to explicitly capture and visualize it using dependencies stereotyped by «allocate».

3.5 Support for AADL analysis

As mentioned in Section 2.2, AADL supports quantitative analysis of non-functional properties as well as early prediction of critical systems qualities. In order to take advantage of the analysis capabilities of AADL, we can use ExSAM to extract the fragment of a SysML or an ExSAM model that conforms to AADL. We can then transform this fragment to AADL using the mappings specified in ExSAM, and apply AADL analysis tools to the result of transformation. Note that the only modeling elements of SysML that are translatable to AADL are those in level 2 of Figure 1 because these are the only concepts that have corresponding elements in AADL.

4 Application and Evaluation of the Profile

The ExSAM profile presented in Section 3 is implemented and its applicability and usefulness are evaluated through two case studies. The first case study, presented in Section 4.1, is taken from the Carnegie Mellon Software Engineering Institute (SEI) report [11] on applying AADL to analyze an avionics system design. The purpose of this case study is to evaluate the ability of ExSAM in capturing all the AADL concepts. The next case study, presented in Section 4.2, is a real industrial case study, in which we applied ExSAM to model one of the Subsea Production Systems of FMC Technologies [3]. The goal of this case study is to evaluate the ability of ExSAM in addressing the modeling requirements of large, distributed, integrated control systems (ICSs).

We have implemented ExSAM as a set of stereotypes and constraints using IBM Rational Rhapsody Architect for Systems Engineers, version 7.5.1 [4], which supports SysML.

4.1 The Avionics case study

In the first case study, we used ExSAM to model an avionics system. The AADL model for this system is presented in [11] and is accessible from [1]. An avionics system typically consists of a collection of hardware and software components that controls the flight, navigation, and radio communication [11].

The SysML features used in this case study are bdds and ibds to show the blocks, their relationships, and their internal structures. In addition, the model consists of a total of six SysML FlowSpecifications for modeling AADL port groups and 49 blocks for capturing the components. The blocks are all stereotyped by either «ComponentType» or «ComponentImpl». In addition, each block is stereotyped by one of the category identifier stereotypes specified in Section 3.1. There are 10 realization relationships that are stereotyped by the newly proposed «AADL_Realization», and four generalization relationships that are stereotyped by «AADL_Generalization». The ExSAM model and diagrams created for this case study are presented in Appendix B of this technical report.

From the above description, we can conclude that the developed ExSAM model captured all the features and details in the AADL model for the avionics system. As expected, many aspects of the AADL model required that we use the ExSAM features as SysML turned out to be insufficient.

4.2 The FMC case study

As a typical integrated control system, one of the subsea control systems of FMC Technologies (henceforth referred to simply as FMC) is selected as the industrial case study in this work. FMC is a leading global provider of technology solutions for the energy industry. One of the key technologies of FMC subsea systems is the Subsea Production System (SPS), which is used for managing and improving oil production fields. The main component of the system is the Subsea Control Module (SCM), which contains electronics, instrumentation, and hydraulics for safe and efficient

operation of subsea valves, chokes, etc. FMC subsea systems are large-scale, integrated and distributed systems of systems connected through high speed electric and fiber-optic network communication links, highly configurable with various field layouts ranging from single satellite wells to large multiple well developments. In this case study, we focused on FMC SPS, which according to the characteristics listed above, is a typical, complex ICS, which is representative in terms of modeling requirements. Based on the characteristics of SPS and results of the detailed domain analysis we conducted, the main requirements for modeling the architecture of such complex ICSs are:

- Req-1) For a particular installation, we need to model how the SPS is configured into a deployable product by capturing how its software and hardware components are connected and what roles they play,
- Req-2) For a particular installation, we must specify the software deployment across distributed hardware computing resources, which in our case consists of many instances of the SCM,
- Req-3) We need to model the behavior of the SPS in several possible modes of operation. For example, the SPS can be operated by different topside control modules, requiring operation in different control modes. It can also operate in the maintenance mode or the normal operation mode. In each mode, we need to identify which hardware and software components are actively operating, and identify constraints and behavior related to this mode.
- Req-4) We need to specify requirements and link them to the SPS architecture and design. This is very important to support safety inspection and certification in the maritime and energy sectors.
- Req-5) The hardware characteristics on which the software will be deployed need to be specified to facilitate the actual configuration of the SPS and to enable performance and resource consumption analysis.

In addition, there are practical considerations that should be accounted for. FMC wishes to use well-supported commercial tools and, because of their use of schematics for mechanical and hydraulic parts, they also favor the use of graphical notations for modeling SPS.

We applied ExSAM to model the architecture of the FMC SPS. In the model, we have a total of 13 bdds, 15 ibds, two state machine diagrams for describing modes and mode transitions, three sequence diagrams and two activity diagrams for describing behaviors. We used, 104 blocks stereotyped by category identifier stereotypes of ExSAM to model different components of the system. In this model, 43 realization relationships are stereotyped by «AADL_Realization», and 18 generalization relationships are stereotyped by «AADL_Generalization». Deployment of software to processors, and processors to underlying hardware for one part of the system is modeled using seven SysML allocation links, and is visualized in an ibd.

The description of the ExSAM model for the FMC case study suggests that SysML provides the basis for achieving the above requirements with its block concept, various diagram types (e.g., ibd,

bdd, state machine diagram), and allocation modeling capabilities. SysML blocks, bdds and ibds can together be used to describe an FMC product through its physical and logical elements, their relationships, and internal structures, addressing Req-1. As pointed in Req-2, an important modeling requirement for the FMC case study is to capture software deployment to distributed hardware computing resources. In the ExSAM model, this is done through SysML allocations, which are equivalent to AADL bindings. Req-4 can be fulfilled using SysML requirements modeling capability, including the specification of requirements, their relationships (e.g., decomposition), and the traceability links between requirements or between them and other model elements.

In addition to the above, the FMC case study illustrated that SysML alone does not have the necessary mechanisms to satisfy Req-3 and Req-5, while ExSAM does. Using the «ModeParticipant» stereotype and its association to UML State metaclass (Figure 10), ExSAM can explicitly identify the components that are currently active in a mode and also link each mode to its own behavior, structure, and/or constraints (Section 3.3). Regarding Req-5, using newly introduced stereotypes and their attributes in ExSAM (e.g., «Memory», «Device») allows us to capture the hardware characteristics of the SPS such that system configuration can be facilitated and performance and resource consumption analysis can then be supported.

As mentioned earlier in this section, one practical consideration at FMC is the need for well-supported commercial modeling tools. To the best of our knowledge, however, AADL lacks professional tool support and a well-defined, complete graphical notation. AADL tool support is restricted to only one commercial tool and a few open-source ones. In contrast, SysML is supported by an increasing number of commercial (e.g. [2, 4, 5]) and open source tools justifying our choice for using SysML as the basis for ExSAM. In addition, AADL has no mechanisms to model requirements and traceability. However, AADL has some important features such as modes, and detailed component categories that are missing from SysML, thus justifying their reuse in ExSAM.

In sum, ExSAM brings missing features from AADL into SysML so that we can benefit from both AADL and SysML strengths. We applied ExSAM to model the architecture of the FMC SPS, which is a typical and complex ICS, and is representative in terms of architecture modeling requirements in ICSs domain. According to the two domain experts who reviewed the resulting architecture models, ExSAM is able to fulfill the five requirements mentioned above.

5 Related Work

MARTE is a UML profile for modeling real-time and embedded systems [8]. The two approaches presented in [8] and [10], that have used MARTE to create AADL-like models, suggest that MARTE is an interesting alternative to SysML. MARTE can indeed be extended with AADL-like constructs, as we did with SysML. However, in this work we chose to focus on SysML because of its wide acceptance in a wide spectrum of industrial sectors, as well as its support for systems engineering through features such as traceable requirements and parametric diagrams.

Combining SySML and MARTE is another alternative to bring together SysML's systems engineering constructs and MARTE's ability in specifying non-functional aspects, thus enabling quantitative analysis. The combination of SySML and MARTE is investigated and discussed in [9] in the context of four usage scenarios.

6 Conclusion and future work

The increasing complexity of integrated control systems demands more effective design languages that can address, in a consistent manner, the heterogeneity resulting from the multidisciplinary nature of such systems.

In this paper, we describe how we combined two highly complementary standard modeling languages, SysML and AADL, to provide a common modeling language (in the form of the ExSAM profile) for specifying embedded systems at different abstraction levels, and from different stakeholder perspectives.

In Section 3, we specified ExSAM, which extends SysML with AADL-like concepts. The applicability and usefulness of ExSAM were investigated through two case studies. One benchmark case study showed that ExSAM can fully cover all AADL aspects and one large-scale industrial case study, performed in collaboration with an industrial partner developing integrated control systems, showed that ExSAM could successfully address all their modeling requirements, whereas neither SysML nor AADL could do so in isolation.

Future work will include the development of tool support for extracting relevant information from models expressed with ExSAM into analyzable AADL models.

Acknowledgments

This work was supported by a grant from Det Norske Veritas (DNV) and Simula Research Laboratory, Norway, in the context of the ModelME! project. We are grateful to FMC Technologies Inc. for their support and help on performing the industrial case study.

References

[1] Aadl model for the avionics case study. `http://aadl.sei.cmu.edu/aadl/downloads/Models/IntegratedModel10292007.zip`.

[2] Enterprise Architect Tool. `http://www.sparxsystems.com/`.

[3] FMC Technologies, Inc. http://www.fmctechnologies.com/.

[4] IBM Rational Rhapsody Architect for Systems Engineers. `http://www-01.ibm.com/software/rational/products/rhapsody/sysarchitect/`.

[5] MagicDraw SysML Plugin. `http://www.magicdraw.com/sysml`.

[6] OMG systems modeling language. http://www.omgsysml.org/.

[7] UML 2.0 Superstructure Specification, August 2005.

[8] A UML profile for MARTE: Modeling and analysis of real-time embedded systems, May 2009.

[9] H. Espinoza, D. Cancila, B. Selic, and S. Gérard. Challenges in combining SysML and MARTE for model-based design of embedded systems. ECMDA-FA '09, pages 98–113, Berlin, Heidelberg, 2009. Springer-Verlag.

[10] M. Faugere, T. Bourbeau, R. de Simone, and S. Gérard. MARTE: Also an UML profile for modeling AADL applications. *Engineering of Complex Computer Systems, IEEE International Conference on*, 0:359–364, 2007.

[11] P. H. Feiler, D. Gluch, J. J. Hudak, and B. A. Lewis. Embedded system architecture analysis using SAE AADL. Technical report, Carnegie Mellon Software Engineering Institute, 2004.

[12] P. H. Feiler, D. P. Gluch, and J. J. Hudak. The Architecture Analysis & Design Language (AADL): An Introduction. Technical report, Software Engineering Institute, 2006.

[13] N. J. Nunes, B. Selic, A. R. da Silva, and J. A. T. Álvarez, editors. *UML Modeling Languages and Applications, UML 2004 Satellite Activities, Lisbon, Portugal, October 11-15, 2004, Revised Selected Papers*, volume 3297 of *Lecture Notes in Computer Science*. Springer, 2005.

[14] W. Schafer and H. Wehrheim. The challenges of building advanced mechatronic systems. In *2007 Future of Software Engineering*, FOSE '07, pages 72–84, Washington, DC, USA, 2007. IEEE Computer Society.

[15] T. Weigert and F. Weil. Practical experience in using model-driven engineering to develop trustworthy computing systems. In *Proceedings of the IEEE International Conference on Sensor Networks, Ubiquitous, and Trustworthy Computing*, volume 1 of *5–7*, pages 208–217, 2006.

[16] T. Weilkiens. *Systems Engineering with SysML/UML: Modeling, Analysis, Design*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2008.

A AADL domain model

A.1 Component types and component implementations

In this appendix we present the domain model for AADL. This domain model shows AADL concepts and constructs and their relationships. Figure 13 shows the core AADL concepts. These include ComponentClassifier, ComponentType, ComponentImpl, which are abstract concepts. As shown in this figure each component type (denoted by ComponentType) is composed of a set of Features, FlowSpecifications, and PropertyValueAssociations; and each component implementation (denoted by ComponentImpl) is composed of a set of Subcomponents, FlowPathSpecifications, Connections, CallSequences, Modes, ModeTransitions, and PropertyValueAssociations.

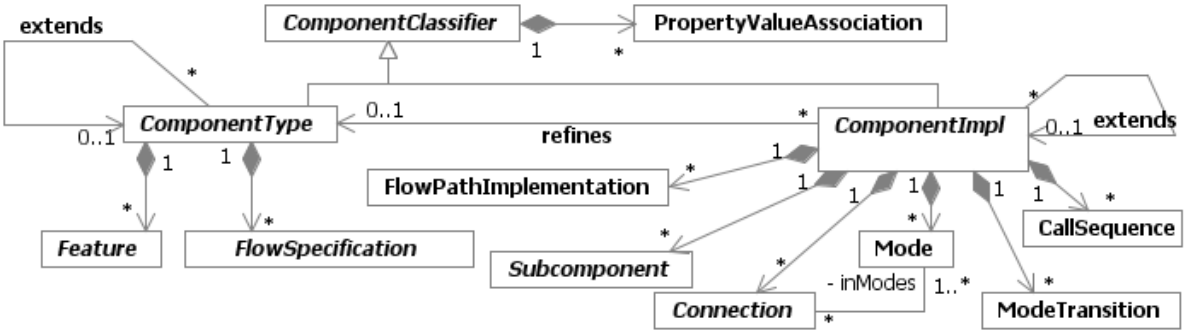


Figure 13: The core AADL concepts.

In AADL, components are declared as a type or implementation within a particular *component category* [12]. AADL provides ten component categories to define the runtime nature of software, hardware, and composite components. Software component categories are *data, subprogram, thread, thread group* and *process*. Hardware component categories are *memory, processor, bus* and *device*. A special component category named *system* indicates either a system consisting of several software components only, or a system consisting of both hardware and software components. In Section A.2 we briefly explain each component category.

A.2 Component Categories

Figures 14 and 15 show the fragments of the domain model for, respectively, software and hardware component categories.

Figures 16 and 17 respectively show the relationships among software and hardware component categories.

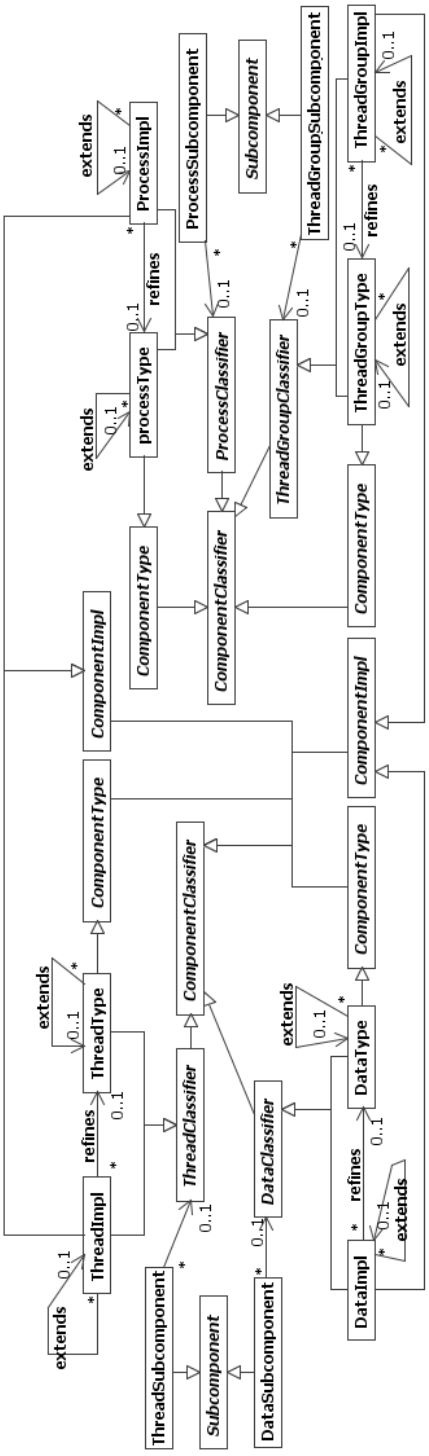


Figure 14: Software component categories.

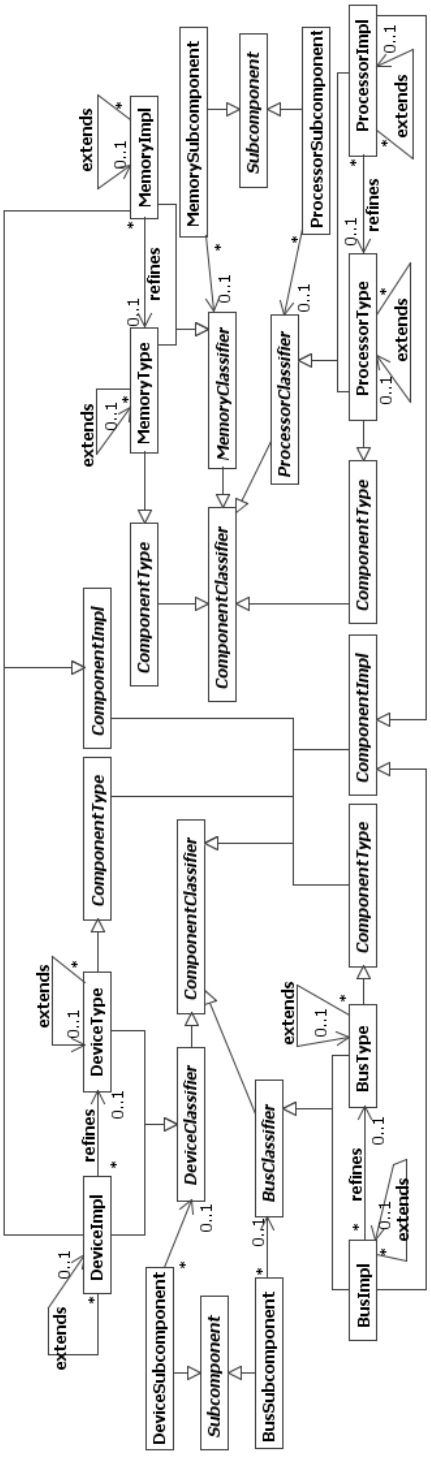


Figure 15: Hardware component categories.

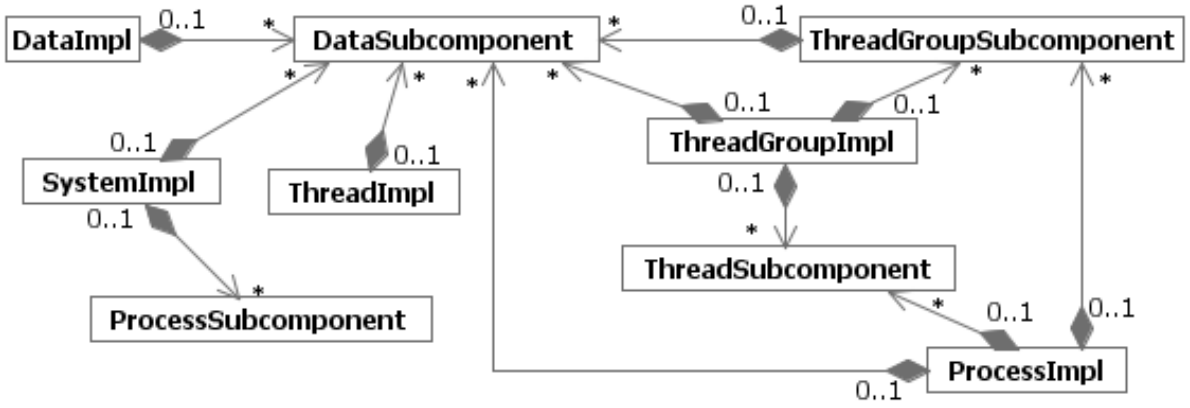


Figure 16: Relationships among software component categories.

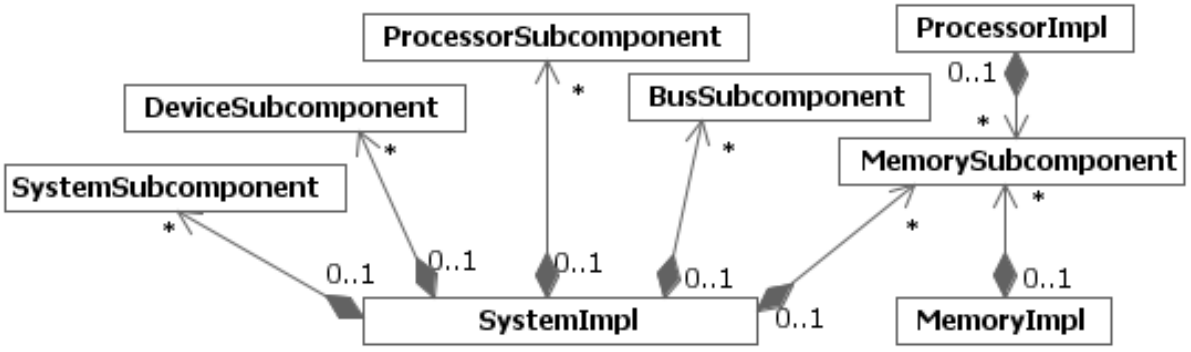


Figure 17: Relationships among hardware component categories.

B ExSAM model for the avionics case study

In this section we provide the SysML model for the Avionics case study. The SysML model is organized into packages similar to the packages in the AADL model. We have followed the mapping rules presented in this report to create the SysML model.

B.1 AppTypes

This package contains the declaration of several data components and port group types, which are modeled as blocks and FlowSpecifications, respectively. Figure 18 is the bdd structuring these blocks and FlowSpecifications.

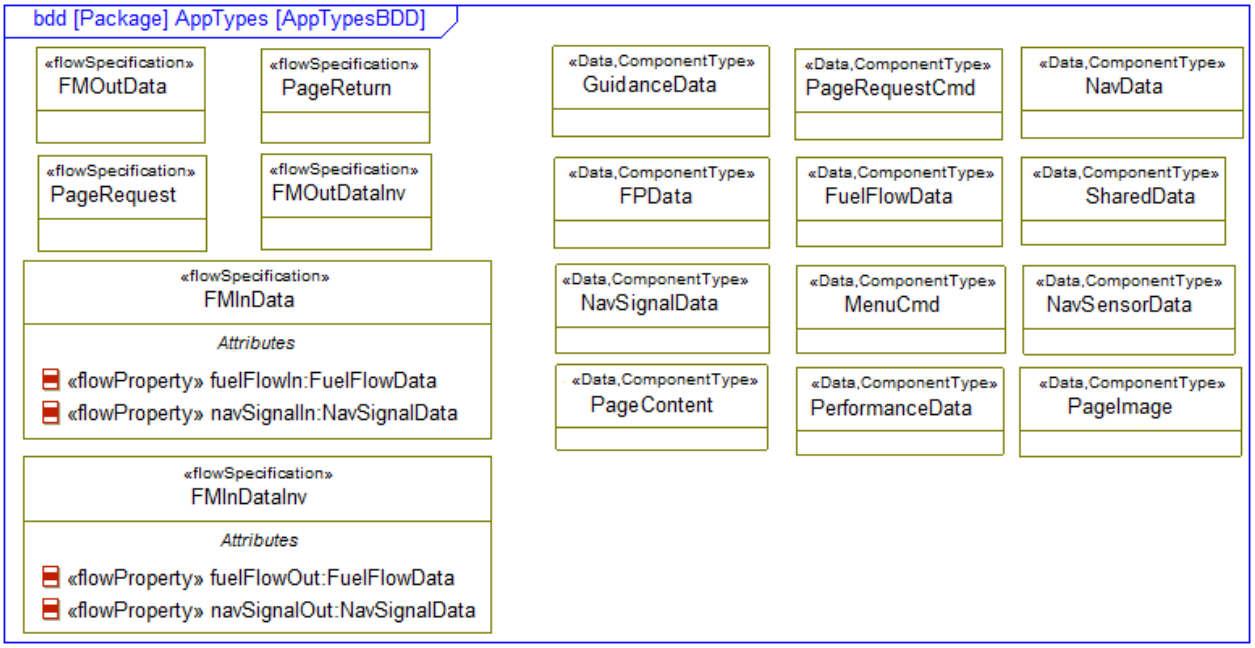


Figure 18: Interfaces and FlowSpecifications representing data components and port groups.

B.2 HardwareParts

Figure 19 shows a bdd structuring hardware parts of the Avionics case study. As it is shown in the figure, we use blocks stereotyped by «ComponentType» to represent component types, and blocks stereotyped by «ComponentImpl» to model component implementations. Category identifier stereotypes are also applied to these blocks to indicate their components categories. Note

that the realization relation between a block representing a component implementation and a block representing a component type is stereotyped by «AADL_Realization».

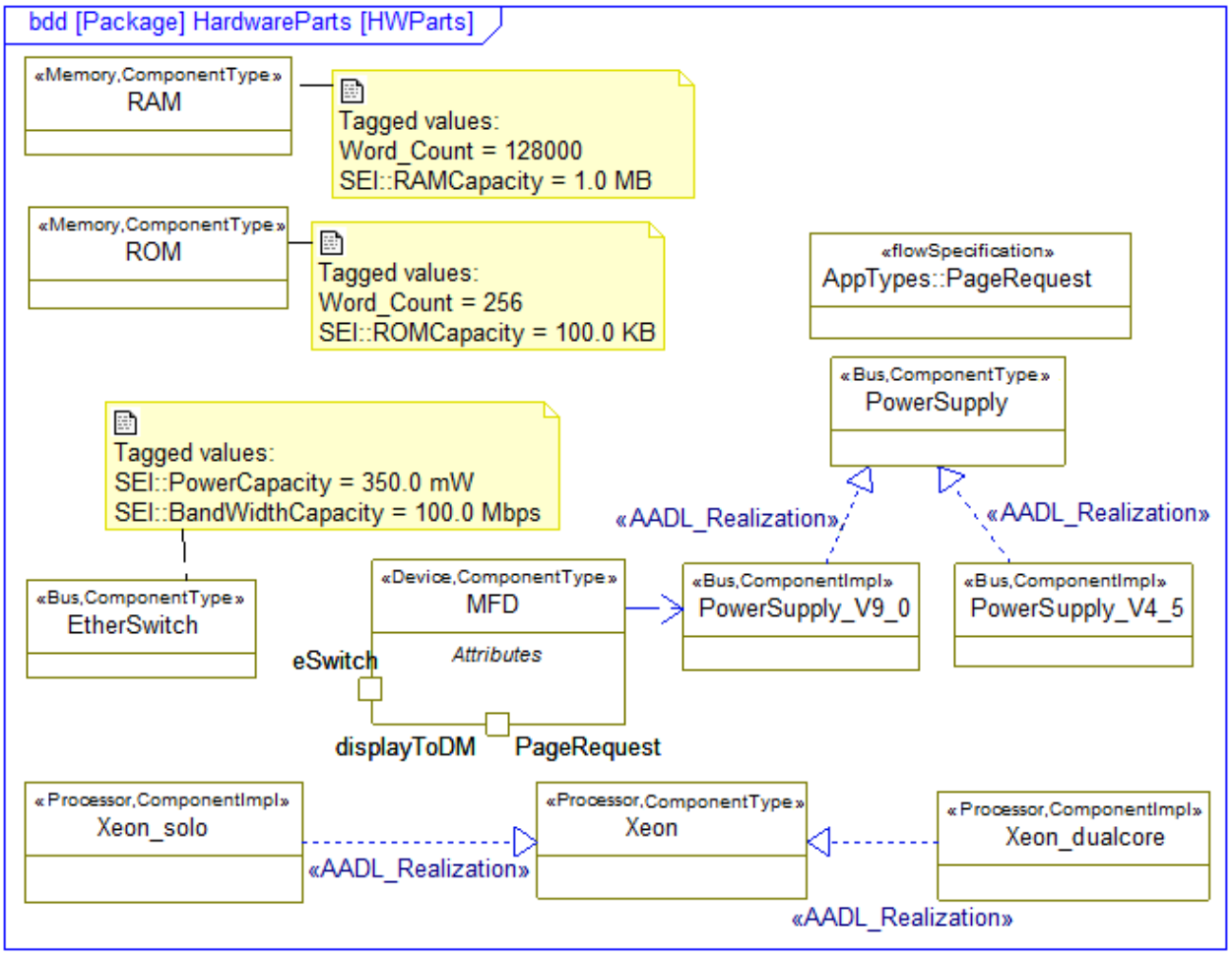


Figure 19: A bdd representing hardware parts.

B.3 AppSubSystems

Figure 20 shows a bdd containing SysML block representing AADL components modeling application subsystems.

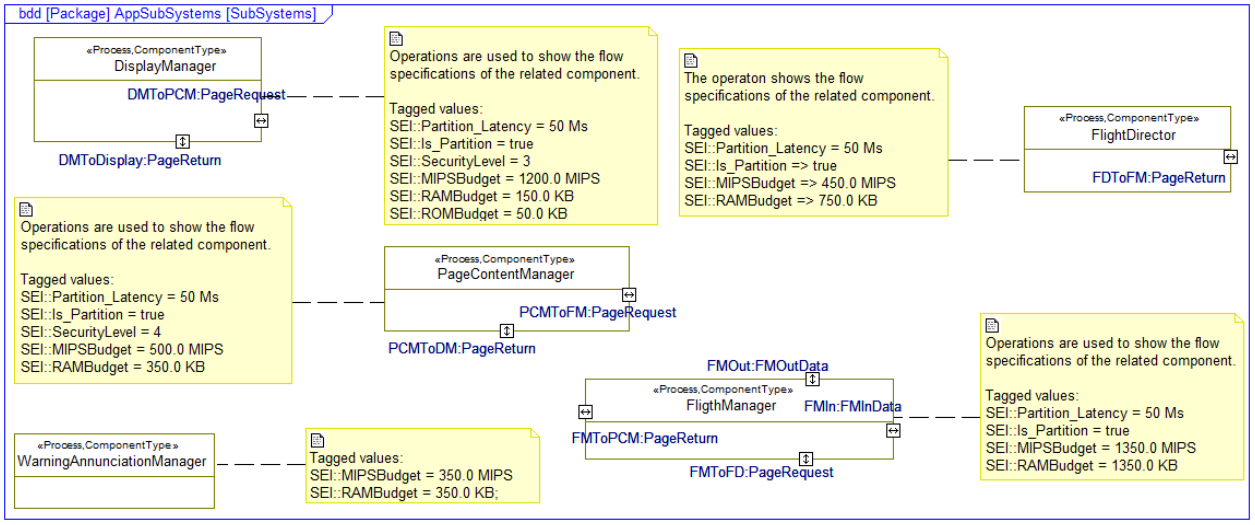


Figure 20: A bdd representing hardware parts.

B.4 FlightManager

The bdd structuring components of the package is given in Figure 21. The internal structure of one of the implementations of `FlightManager` (`FlightManager_noPIO`) is presented in Figure 22. The ibd representing the shared memory communication among subcomponents in `FlightManager` is given in Figure 23.

B.5 AppSystem

The bdd in Figure 24 models the component `EemeddedApp` and its four different implementations. Figure 25, Figure 26, Figure 27, Figure 28 show four ibds representing the four different implementations for the `EmbeddedApp` component.

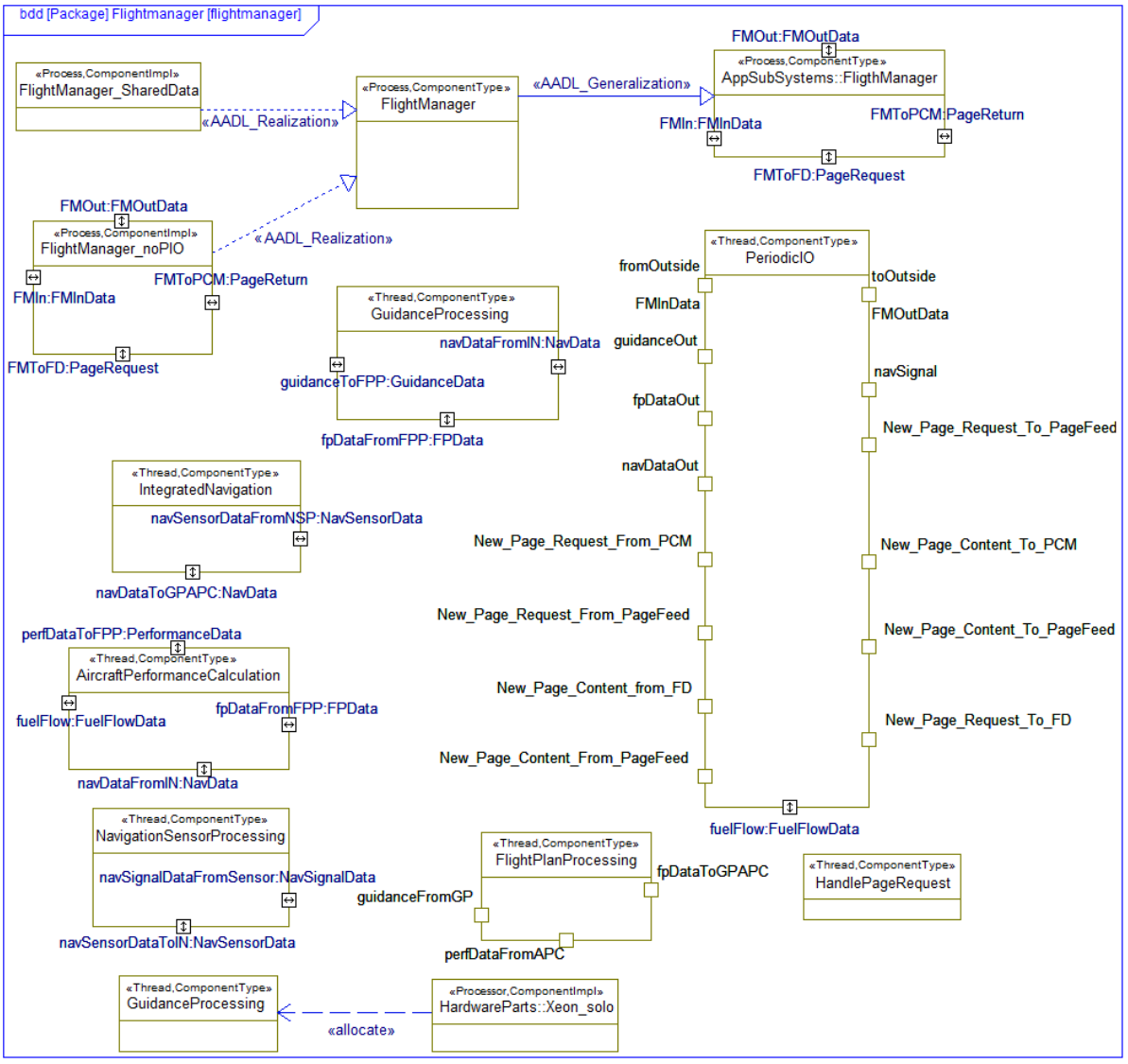


Figure 21: A bdd showing blocks representing the components used in the thread-based implementation of the flight manager.

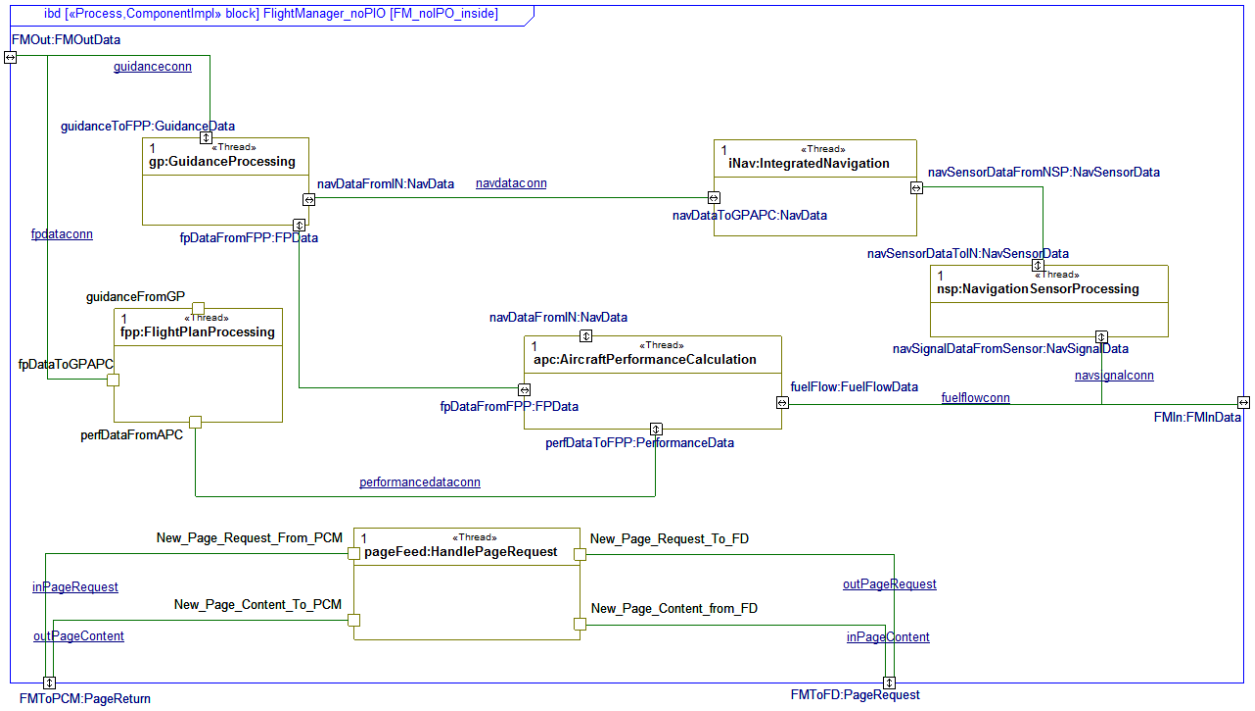


Figure 22: An ibd for FlightManager_noPIO, providing an implementation for flight manager.

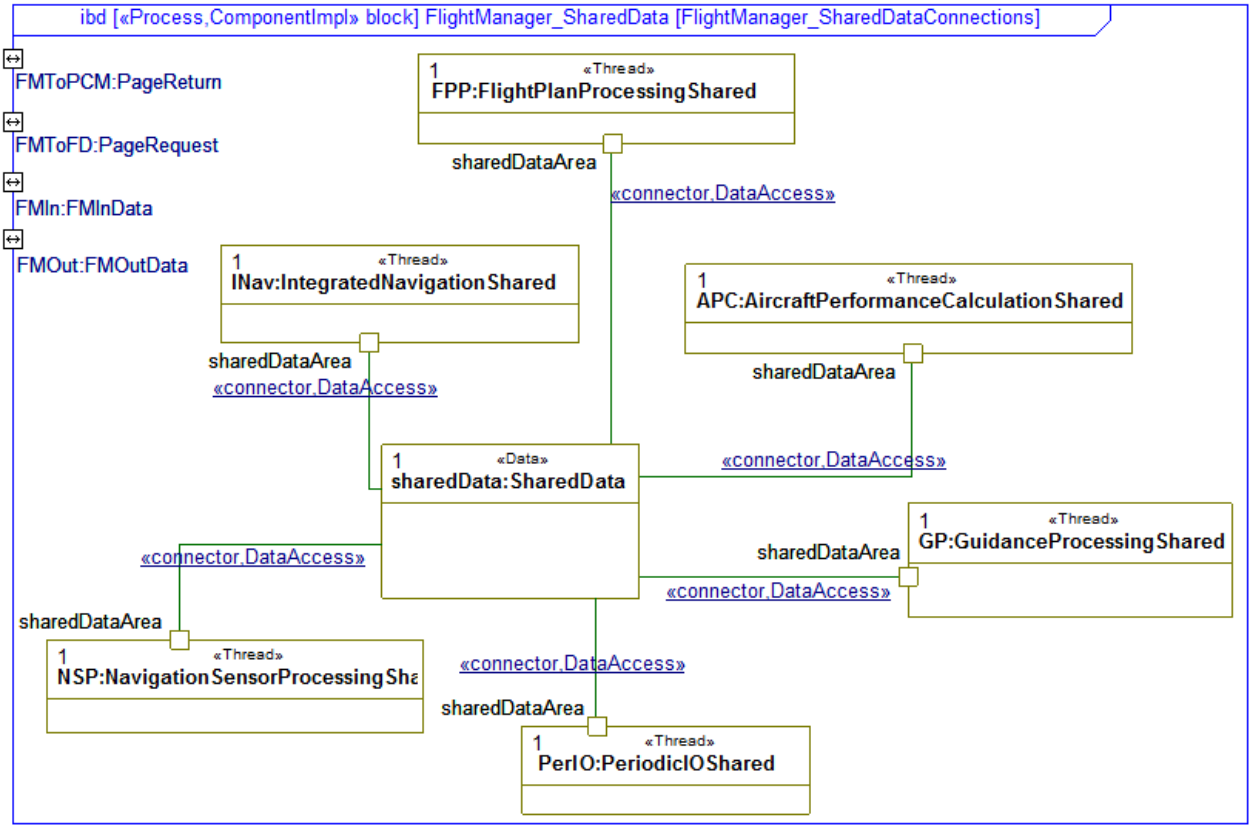


Figure 23: An ibd representing the shared memory communication among subcomponents of flight manager.

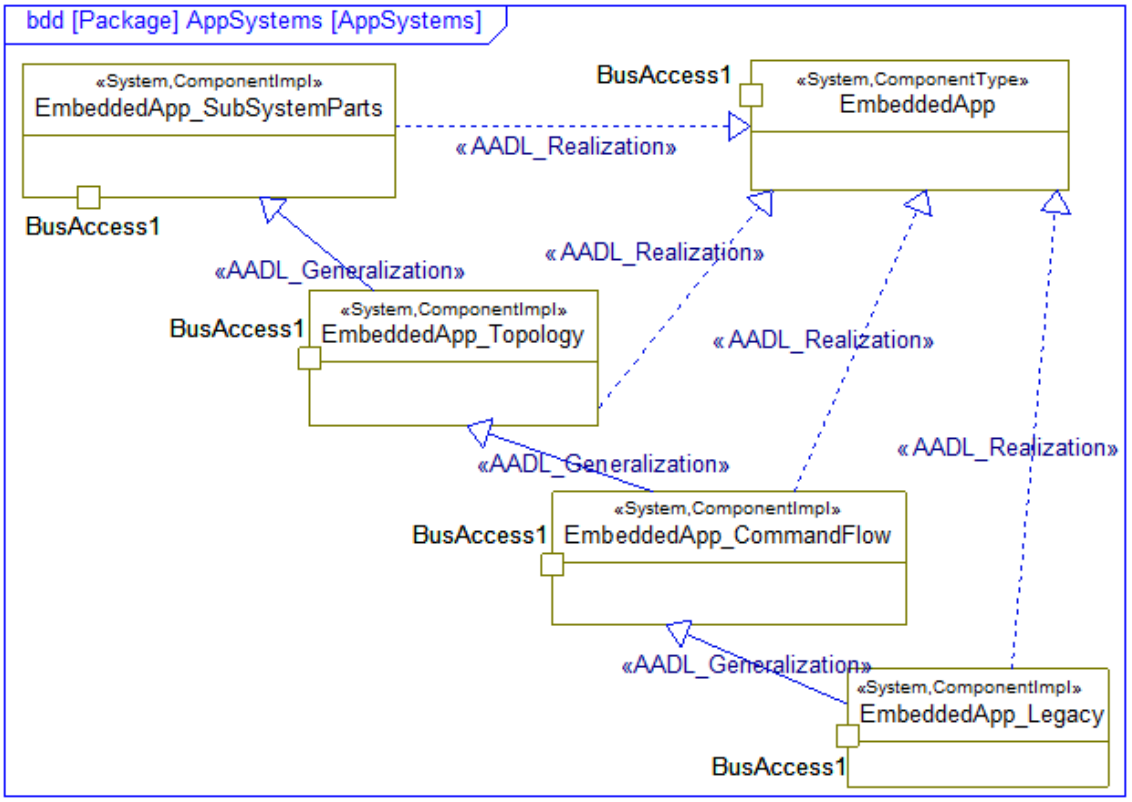


Figure 24: A bdd EmbeddedApp and four different implementations for it.

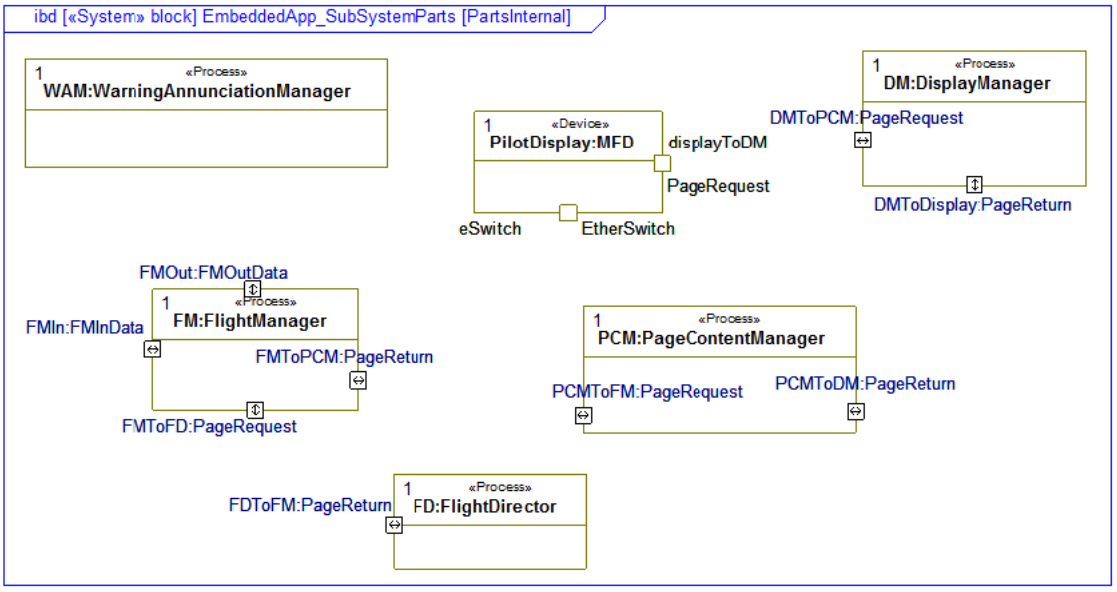


Figure 25: An ibd showing a possible implementations for EmbeddedApp. This implementation only declares the subcomponents (parts) of the component.

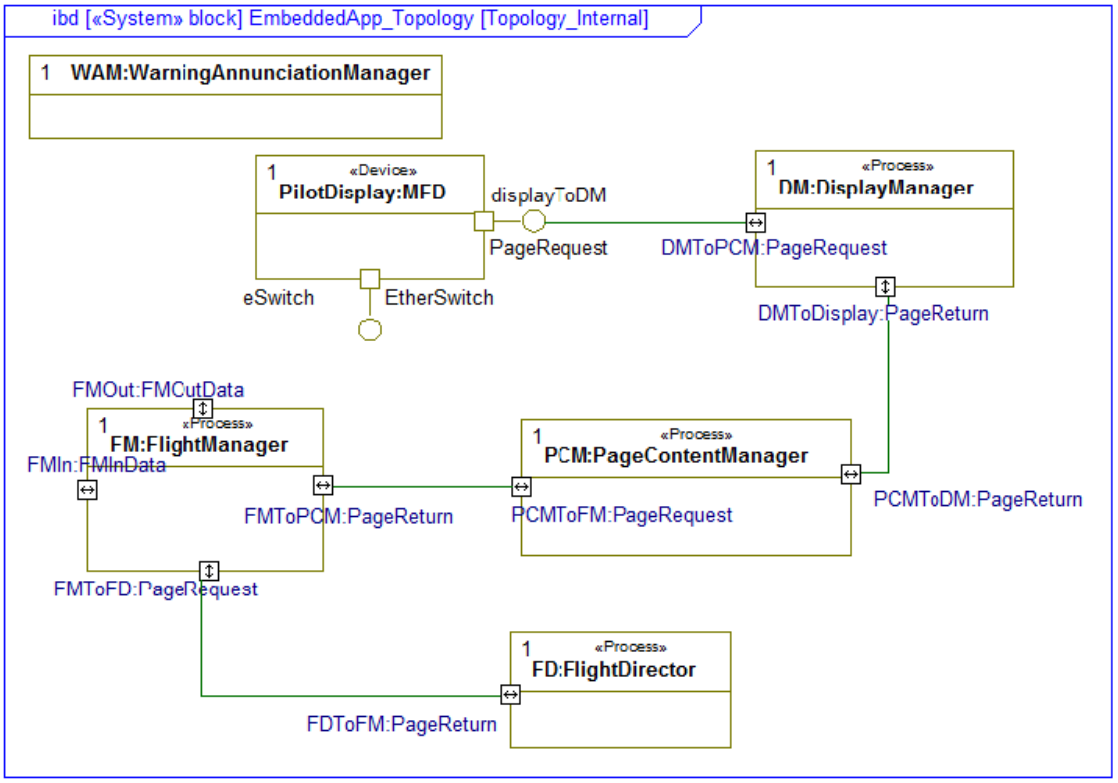


Figure 26: Another implementations for EmbeddedApp showing the connections between sub-components.

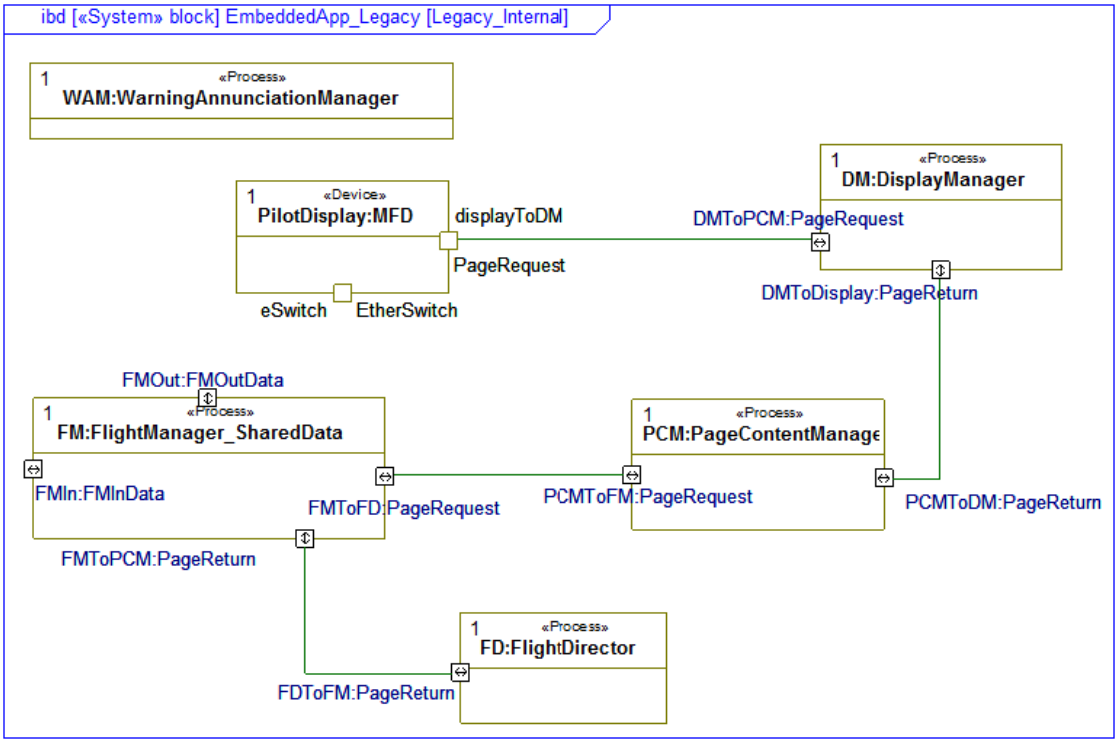


Figure 27: Another implementations for EmbeddedApp, EmbeddedApp_Legacy. In this implementation the part FM is replaced by one of its refinements FlightManager_SharedData.

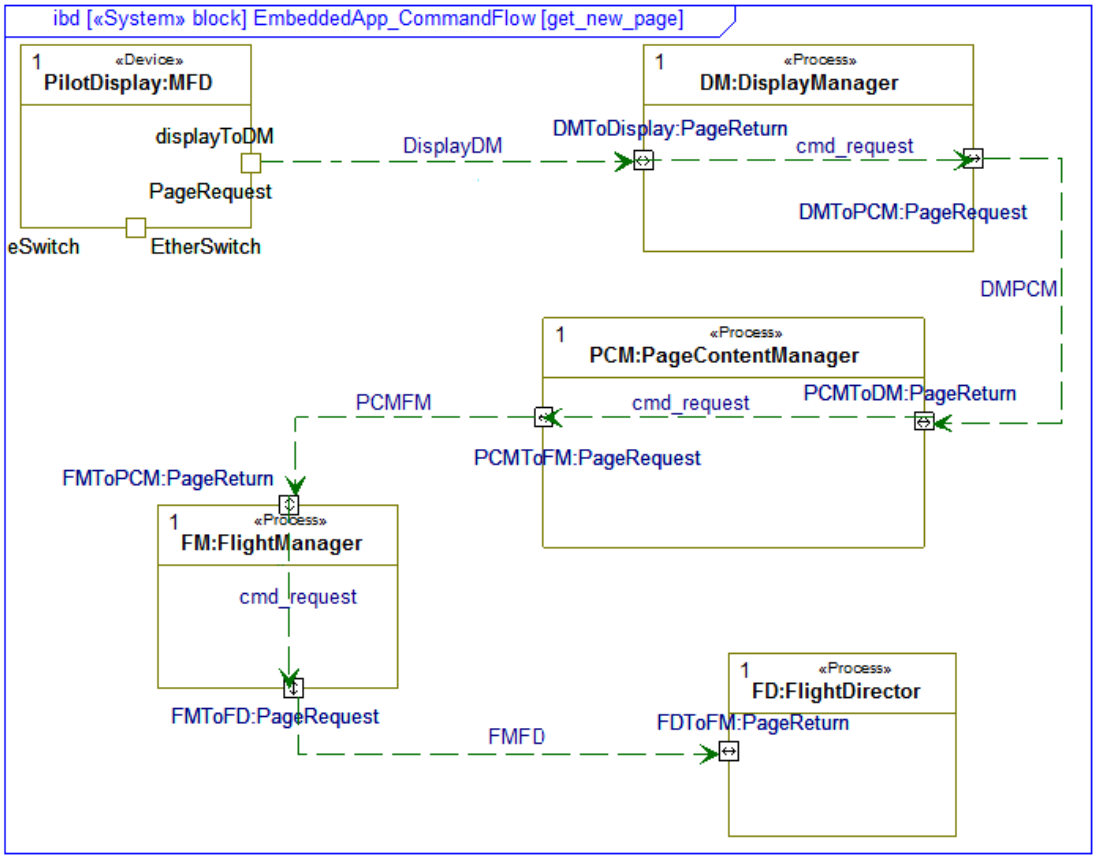


Figure 28: Another implementations for EmbeddedApp, EmbeddedApp_CommandFlow. This implementation describes one possible flow among the subcomponents.