



January 2014

Translation Of AADL To PNML To Ensure The Utilization Of Petri Nets

Amrita Chatterjee

Follow this and additional works at: <https://commons.und.edu/theses>

Recommended Citation

Chatterjee, Amrita, "Translation Of AADL To PNML To Ensure The Utilization Of Petri Nets" (2014). *Theses and Dissertations*. 1629.
<https://commons.und.edu/theses/1629>

This Thesis is brought to you for free and open access by the Theses, Dissertations, and Senior Projects at UND Scholarly Commons. It has been accepted for inclusion in Theses and Dissertations by an authorized administrator of UND Scholarly Commons. For more information, please contact zeinebyousif@library.und.edu.

TRANSLATION OF AADL TO PNML
TO ENSURE THE UTILIZATION OF PETRI NETS

By

Amrita Chatterjee
Bachelor of Science, Guru Nanak Institute of Technology, India, 2009

A Thesis

Submitted to the Graduate Faculty

of the

University of North Dakota

In partial fulfillment of the requirements

for the degree of

Master of Science

Grand Forks, North Dakota

August 2014

Copyright 2014 Amrita Chatterjee

This thesis, submitted by Amrita Chatterjee in partial fulfillment of the requirements for the Degree of Master of Computer Science from the University of North Dakota, has been read by the Faculty Advisory Committee under whom the work has been done and is hereby approved.



Dr. Hassan Reza (Chairperson)



Dr. Emanuel Grant

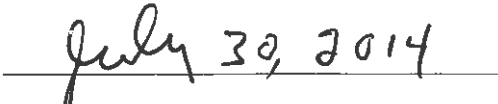


Dr. Wenchen Hu

This thesis is being submitted by the appointed advisory committee as having met all of the requirements of the School of Graduate Studies at the University of North Dakota and is hereby approved



Dr. Wayne Swisher
Dean of the School of Graduate Studies



Date

PERMISSION

Title Translation of AADL To PNML To Ensure the Utilization of Petri Nets
Department Computer Science
Degree Master of Computer Science

In presenting this thesis in partial fulfillment of the requirements for a graduate degree from the University of North Dakota, I agree that the library of this University shall make it freely available for inspection. I further agree that permission for extensive copying for scholarly purposes may be granted by the professor who supervised my dissertation work, or, in his absence, by the Chairperson of the department or the dean of the School of Graduate Studies. It is understood that any copying or publication or other use of this dissertation or part thereof for financial gain shall not be allowed without my written permission. It is also understood that due recognition shall be given to me and to the University of North Dakota in any scholarly use which may be made of any material in my thesis.

Amrita Chatterjee

July 25, 2014

TABLE OF CONTENTS

LIST OF FIGURES	ix
LIST OF TABLES	xi
ACKNOWLEDGEMENTS	xii
ABSTRACT	xiii
CHAPTER	
I. INTRODUCTION	1
1.1 Research Definition	1
1.1.1 Non-Functional Properties	3
1.1.2 Functional Properties.....	3
1.1.3 The Need for Petri Net Markup Language	4
1.2 Motivation	4
1.2.1 What is Correctness and Why Does it Matter in Safety-Critical systems?.....	5
1.2.2 How Does a Petri Net Address AAD Limitations in Safety-Critical Systems?	5
1.2.3 Why Extend AADL to PNML?.....	5
1.3 Approach	5
1.3.1 Case Study Elements of AADL.....	6
1.3.2 Mapping of AADL to PNML.....	6

1.4	Scope	6
1.4.1	Applicability of PNML	6
1.4.2	Traditional Approach	7
1.5	Expected Results	7
1.6	Thesis Structure	8
II.	BACKGROUND	10
2.1	Architecture Analysis Design Language	11
2.2	Petri Nets	13
2.2.1	Definition of Acronyms [Define the Terms]	15
2.2.2	Design Process	16
2.3	Transformation Techniques	16
2.4	Related Work	16
2.4.1	Petri Net Type Definition	18
2.5	PNML	19
2.5.1	Limitations and Benefits of PNML	21
2.5.2	The PNML Framework	21
2.6	Improving the Transition from AADL to PNML	22
2.6.1	Related Work	22
2.6.2	PNML Ontology	23
2.6.3	Petri Net Type Definition	23
2.6.4	AADL-OSATE	24
2.6.5	2009 Petri Net Workshop	25
2.6.6	PNML Core Model	25
2.6.7	Mapping of PNML Core Model to XML	26

	2.6.8 Summary of the Project	27
	2.6.9 Development Process	28
III.	METHODOLOGY	29
	3.1 Description of Methodology	29
	3.2 Steps of the Methodology	30
	3.3 Algorithms	32
	3.3.1 AADL Conversion	32
	3.3.2 PN Conversion	32
	3.3.3 Mapping.....	33
	3.3.4 Petri Net Markup Language	35
	3.3.5 XSLT.....	36
	3.3.6 High Level Petri Nets	36
	3.3.7 Data Types.....	36
	3.3.8 PNML Framework	36
	3.3.9 Applications of PNML Frameworks	37
	3.4 Development	38
	3.4.1 The Aarhus Petri Net Workshop	38
	3.4.2 Extensions of AADL.....	40
	3.4.3 Combining AADLs with High Level Petri Nets	40
	3.4.4 Strengths and Weaknesses of AADL	41
	3.5 Description of the Proposed Model	41
	3.5.1 Algorithm Explanation with Diagram.....	42
	3.6 Summary of the Methodology	43
IV.	CASE STUDY	44
	4.1 Case Elements	44

4.2	Analysis of the Case Study	47
4.2.1	Description of XSLT and How XSLT Works.....	50
4.2.2	Application of XSLT Script	53
4.3	Results of the Case Study	54
V.	CONCLUSION	56
5.1	Future Work	57
	APPENDICES	58
Appendix A	AADL Specifications	59
Appendix B	Java Programs.....	60
Appendix C	Petri Net Type Definitions.....	66
	REFERENCES	67

LIST OF FIGURES

Figure	Page
1. AADL Components	2
2. Functional and Non-Functional Properties	4
3. Approaches: a) Traditional Approach and b) Proposed Approach.....	7
4. AADL	11
5. A Petri Net and its Component Parts	14
6. PNML Core Model	26
7. The Transformation of AADL to AADL-XML	30
8. The Transformation of AADL-XML to PNML via Mapping Rules	31
9. The Conversion of AADL-XML to PNML via XSLT	32
10. AADL Text Conversion	32
11. PN Conversion	32
12. AADL-XML Conversion to PNML	33
13. PNML	39
14. AADL-XML to PNML- Architecture and Automated Version	42
15. Components of a Cruise Control System	45
16. AADL Text of Device “Wheel Rotation Sensor”	48
17. XML Version of AADL Text for “Device”	48
18. PNML Version of “Device”	50

19.	Graphical Version of the Translation	50
20.	The Structure of how the XSLT Works.....	51
21.	XML Input (AADL-XML)	52
22.	XSL script.....	52
23.	Output XML (PNML)	52
24.	XSLT Script Converting AADL-XML of Device to PNML.....	53
25.	Screenshot of Conversion of AADL-XML to PNML via XSLT	54
26.	The AADL Graphical Representation of Device.....	54
27.	The Whole Transformation from AADL-text to PNML	55

LIST OF TABLES

Table	Page
1. Correlation between AADL and System Architecture	12
2. Representation of Petri Net Components [6]	23
3. Mapping of PNML Core Element to XML [1]	27
4. AADL to PN Mapping	33
5. Mapping of AADL Text Elements to AADL-XML Components.....	48
6. Mapping of Major AADL Elements to PNML.....	49

ACKNOWLEDGEMENTS

I am grateful to my committee members Dr. Hassan Reza, Dr. Emanuel Grant, and Dr. Wen-Chen Hu for their support for the research for my thesis. Without the guidance and the time they have generously provided, this thesis would not have been possible. In particular, I thank Dr. Hassan Reza for agreeing to be my advisor, for introducing me to this fascinating topic, for all the invaluable lessons he has taught me about Architecture Analysis and Design Language and Petri Net Markup Language, and for all the useful comments and remarks he has contributed over the last couple of years. I appreciate the guidance and encouragement he has given me throughout my academic years at UND, and especially throughout the thesis process.

In addition, I am grateful to Dr. Emanuel Grant for agreeing to serve as a member of my committee and for the support and assistance he has provided throughout my academic program at UND. I would also like to thank Dr. Wen-Chen Hu for his support. I would particularly like to thank my family and my fellow students in the program at UND who have supported me throughout this entire process, both by keeping me harmonious and helping me put the pieces together.

ABSTRACT

Architecture Analysis and Design Language (AADL), which is used to design and analyze software and hardware architectures of embedded and real-time systems, has proven to be a very efficient way of expressing the non-functional properties of safety-critical systems and architectural modeling. Petri nets are the graphical and mathematical modeling tools used to describe and study information processing systems characterized as concurrent and distributed. As AADL lacks the formal semantics needed to show the functional properties of such systems, the objective of this research was to extend AADL to enable other Petri nets to be incorporated into Petri Net Markup Language (PNML), an interchange language for Petri nets. PNML makes it possible to incorporate different types of analysis using different types of Petri net. To this end, the interchange format Extensible Markup Language (XML) was selected and AADL converted to AADL-XML (the XML format of AADL) and Petri nets to PNML, the XML-format of Petri nets, via XSLT script. PNML was chosen as the transfer format for Petri nets due to its universality, which enables designers to easily map PNML to many different types of Petri nets. Manual conversion of AADL to PNML is error-prone and tedious and thus requires automation, so XSLT script was utilized for the conversion of the two languages in their XML format. Mapping rules were defined for the conversion from AADL to PNML and the translation to XSLT automated. Finally, a PNML plug-in was designed and incorporated into the Open Source AADL Tool Environment (OSATE).

CHAPTER I

INTRODUCTION

1.1 Research Definition

Architecture Analysis & Design Language (AADL) [9] is an effective approach to the model-based analysis and specification of complex real-time embedded systems such as those used in unmanned aerial vehicles. It is utilized to specify and analyze real-time embedded systems and complex systems, specializing in the performance of the system. It benefits from good semantics and is used for the verification of non-functional properties. AADL is a very specific language. AADL and its supporting toolset, the Open Source Architectural Tool Environment (OSATE)[51], were first issued in November 2004 as part of the standardization program run by SAE (the Society of Automotive Engineers in the United States) (SAE AS5506)[9]. Because code tests are generated from models, the design of the models plays an important role. AADL incorporates both textual and graphical notations for defining the software and hardware of the system, focusing on three main types of components:

- a) Software components,
- b) Hardware components, and
- c) System components.

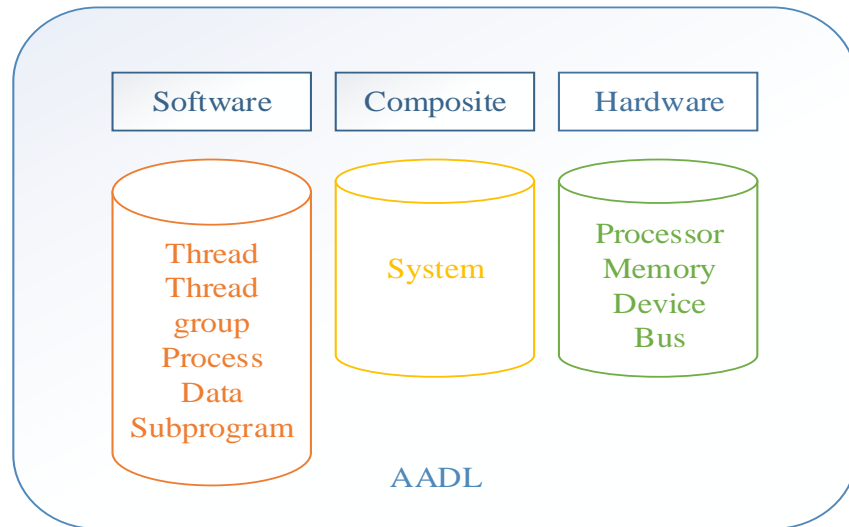


Figure 1. AADL components.

As Figure 1 shows, the software components include the process, thread, thread group, subprogram and data; the hardware components include the processor, memory, bus, and the device itself; and the system components describe how the software components are allocated to the hardware components.

The system components communicate through ports. A process consists of a thread or thread group, where a thread is an AADL component that represents a sequence flow of execution. Thread groups are a convenient abstraction for organizing threads, the data within a process. Interactions between components are performed via features and connections. Features are classified as ports, component access, subprogram calls and parameters. Ports are classified as event data ports, data ports and event ports. A port is

used to exchange data and event information. Connections are links which represent the communication of data between components.

AADL has become increasingly popular in recent years as it offers a good way to deal with safety-critical systems in avionics, automotive and other application domains. AADL is specifically designed for model-based engineering [31], describing how components such as data inputs and outputs are connected and the way the software and hardware components are allocated. AADL, which is based on the component-connector paradigm, enables designers to evaluate the reliability, availability, performance, schedulability and fault tolerance (non-functional properties) of a system and is generally used for modeling large scale systems.

1.1.1 Non-Functional Properties

AADL can be used to determine the non-functional properties of real-time embedded systems. These non-functional properties include important characteristics such as the availability, reliability, and schedulability of both the software and hardware components of real time systems. Originally developed for applications in the field of avionics, AADL contains constructs for modeling both the software and hardware of a system and can be represented both graphically and as text. The software architecture is modeled in terms of components and interactions.

1.1.2 Functional Properties

AADL cannot, however, determine the functional properties or the correctness of the system, both of which are vitally important in safety-critical systems. To address this deficit, Petri nets are a useful mathematical modeling tool that can be applied to determine the behavior of the system. The main shortcoming of AADL-OSATE is that it

is not easy to use to analyze the behavior of the components of a system in order to identify system conditions such as deadlock or livelock.

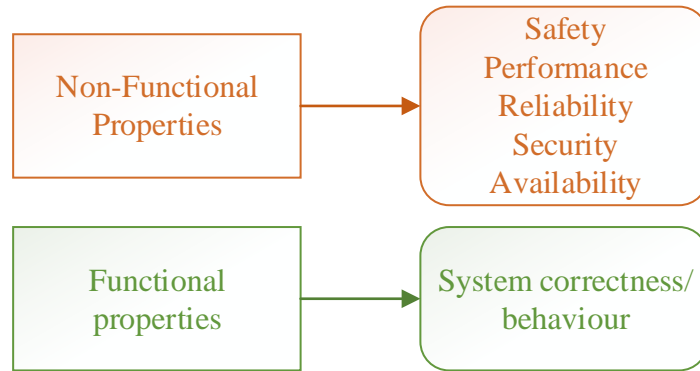


Figure 2. Functional and non-functional properties.

1.1.3 The Need for Petri Net Markup Language

Correctness of the system is essential for the security of the system, but AADL cannot be used to determine the functional properties of real-time embedded systems in its current form. Extending AADL to include the additional functionality offered by Petri nets will address this issue. In order to convert AADL to Petri nets, a new schema must be developed. ISO/IEC 15909-2 [1] is a standardized interchange format for Petri nets that defines the standard format to be the Petri Net Markup Language (PNML)[4], which is known for its interoperability. It facilitates the exchange of Petri nets among different Petri net tools and among different parties.

1.2 Motivation

Though AADL has proven to be an efficient language in the field of aeronautics and space science, it has some limitations. AADL with OSATE supports many of the features involved in analyzing and simulating systems but is unable to analyze their correctness, as AADL cannot detect either the functional properties or the behavior of the

system. Deadlock can also not be detected through AADL. One way to bridge this gap is to introduce a Petri net. It is difficult to convert directly from AADL to a Petri net without going through an intermediate format, so in order to ease the translation XML is used.

1.2.1 What is Correctness and why does it Matter in Safety-Critical Systems?

A safety-critical system requires a high degree of safety assurance. Thus, it is essential to determine the behavior of the components of the system. Petri nets can be used to determine the correctness or functional properties of the system in order to analyze conditions such as deadlock or livelock.

1.2.2 How does a Petri Net Address AADL Limitations in Safety-Critical Systems?

Petri nets are mathematical modeling tools that are used to represent discrete distributed systems. They determine the behavior or correctness (correction to specification or verification) of the system and are applied widely to study concurrent, non-deterministic, stochastic, asynchronous systems.

1.2.3 Why Extend AADL to PNML?

AADL defines the non-functional properties of a system but it is not meant to analyze functional properties. Petri nets have the capability to determine the behavior of a system. If AADL is extended to PNML (Petri Net Markup Language, the interchange language for Petri nets) then Petri nets can be incorporated to facilitate the analysis of other functional aspects.

1.3 Approach

There are several steps involved in extending AADL to the PNML version. First, the AADL text is converted to AADL-XML and the Petri net is converted to PNML (Part

2 of the ISO standard [1]). Once this has been accomplished, AADL-XML can be mapped to PNML, making it easy to then apply it to any kind of Petri net, as PNML is known for its universality [7].

1.3.1 Case Study Elements of AADL

As noted earlier, AADL is used to describe:

- a) Application software,
- b) Execution platform (the hardware), and
- c) Composite (the system).

In the case study described in Chapter IV, one of the hardware components of AADL, “Device”, is considered. The AADL text of “Device” is transformed to the XML version of AADL and then further transformed to its corresponding PNML version.

1.3.2 Mapping of AADL to PNML

Between the AADL text and AADL graphics, the AADL text is converted to AADL- XML and then mapped to the XML version used for Petri nets, PNML, according to the mapping rules[8].

1.4 Scope

1.4.1 Applicability of PNML

PNML ideally consists of the following features:

- a) The PNML core,
- b) The elements, which are basic to all Petri nets, and
- c) The corresponding PNML syntax.

1.4.2 Traditional Approach

In the traditional approach, AADL is directly converted to either Colored Petri Nets, Timed Petri Nets or Stochastic Petri Nets, as shown in Figure 3a. In [45] AADL is directly transformed to a Stochastic Petri Net [45] through the model transformation tool ADAPT to provide a Generalized Stochastic Petri Net (GSPN) in XM/XMI format.

1.5 Expected Results

In the new approach proposed for this project, which is shown in Figure 3b, AADL text is instead transformed to Petri Net Markup Language. Here, XML is utilized as the interchange format to bridge the gap between AADL-XML and PNML. AADL is extended to PNML (via XML) in order to permit the use of other Petri Nets.

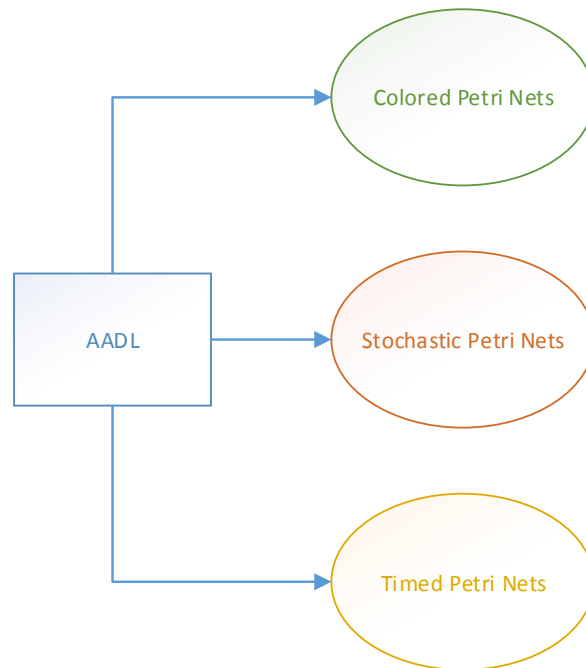


Figure 3(a). Traditional Approach.

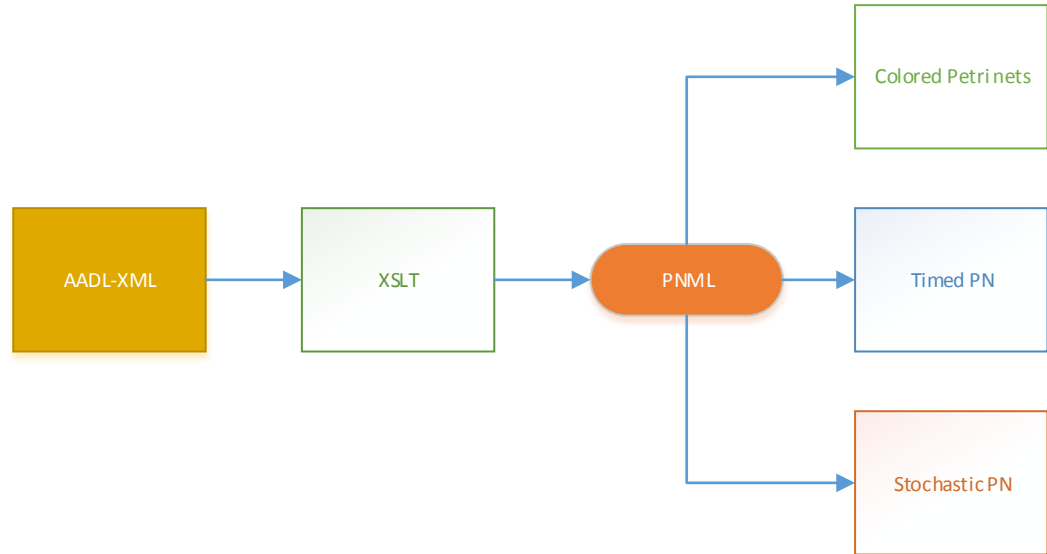


Figure 3(b). Proposed Approach.

1.6 Thesis Structure

This chapter has presented an overview of the whole thesis. AADL is an efficient way to determine the non-functional properties of real-time embedded systems but is unable to address the functional properties of safety critical systems. Instead, Petri nets offer a mathematical modelling tool that can determine the system behavior. PNML is the XML-based format for Petri nets and is known for its universality, making Petri nets interoperable via PNML. It therefore makes sense to extend AADL to PNML by incorporating Petri nets to perform this essential service. In order to do this effectively, an interchange format is required and XML was selected as the most appropriate interchange format between AADL and Petri nets. Once the components are in PNML format, they can be easily transformed into High-level Petri Nets, Symmetric Nets or Place/Transition Nets, as required. XSLT is utilized in the transformation of the XML version of AADL to the XML version of PNML.

The remainder of this thesis is organized as follows. Chapter 2 presents the theoretical background for the research and discusses the relevant literature. Chapter 3

describes the methodology and the development of the new algorithms, which are then tested in the case study presented in Chapter 4 and the results evaluated. The thesis concludes with a summary of the project in Chapter 5, along with suggestions for future research.

CHAPTER II

BACKGROUND

AADL is a textual and graphical language that supports model based engineering of real time embedded systems and is used to express the non-functional properties of a system. It is an important modelling language that is used to analyze system software by describing the dynamic behavior of real-time systems through formal notations, facilitating the design of the software and hardware involved. The model driven safety analysis technique is based on AADL. AADL components are specified in terms of component types and component implementations. SAE AADL is applicable in domains such as avionics, aerospace and automotive systems. AADL has the following advantages [26]:

- AADL makes it possible to apply a system engineering approach to software intensive systems.
- AADL enables architectures to be analyzed and reworked, decreasing their complexity.
- AADL is extendable and provides a good foundation for additional capabilities in automated system integration.

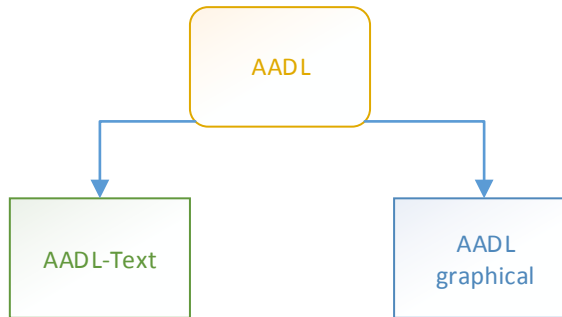


Figure 4: AADL.

Figure 4 depicts the two different forms of AADL, which can be represented by both textual and graphical formats.

2.1 Architecture Analysis Design Language

AADL provides analyses such as Flow Latency Analysis, Security and Safety Analysis, among others [48]. All these features are supported by a variety of specialized tools such as OSATE and Cheddar [48]. However, AADL lacks the flexibility to express some conceptual features of software architecture, although it does support components, connectors and configurations [8]. AADL has been designed to specify and analyze the non-functional aspects of a system, which include security, reliability, performance and safety, but is not intended to analyze the behavior of a system, especially problems like deadlock or livelock that may exist in different subcomponents of the system. Non-functional aspects of components (schedulability, performance, reliability) can be described within an AADL model, which is useful for examining items such as thread dispatching condition, interface specifications and interconnections between components. Elements of the architecture are described by the components and procedures are modeled by subprograms. The active part of an application is modeled by threads. AADL descriptions are hierarchical and AADL can analyze schedulability, error modeling, and

code generation, although it suffers from the absence of concrete operational semantics [15].

Table 1. Correlation between AADL and System Architecture.

AADL	Architecture
Components	Elements
Subprograms	Procedures
Thread	Active part of application
Processors	Processors and OS scheduler
Memories	Storage
Buses	Communication
Devices	Hardware
Process	Virtual Address Space

Table 1 represents the correspondence between the different hardware and software components of AADL and the architecture of a system. The hardware components of AADL consist of the processor, memory, and bus device, while the software components consist of the subprograms, data, thread and process.

An example of AADL- text for a cruise control system[11] is:

```

port group wheel_sensors_socket
features
wheel_pulse: in data port bool_type;
wheel_slippage: in data port real_type;
end wheel_sensors_socket;

```

2.2 Petri Nets

Petri nets are mathematical modeling tools that are used for studying the dynamic behavior and concurrency of a system and are capable of determining the correctness (correctness to specification or verification) of the system. A typical Petri net consists of place, arc and transition, and this graphical notation provides the basic primitives for modeling concurrency and synchronization [12]. The mathematical definition of high level Petri nets provided by the ISO [1] consists of a semantic model, and the graphical form of the technique is referred to as a High Level Petri Net Graph (HLPNG). Petri nets can be used to determine the correctness or the behavior of safety-critical systems. Petri nets can be used for both structural analysis and model checking to identify problems such as deadlock, which is a safety property, and livelock, which is a causal property. Petri nets were invented in 1962 to facilitate modelling the behavior of complex software systems. Parallelism, non-determinism, synchronicity and distributedness can also be modelled using Petri nets.

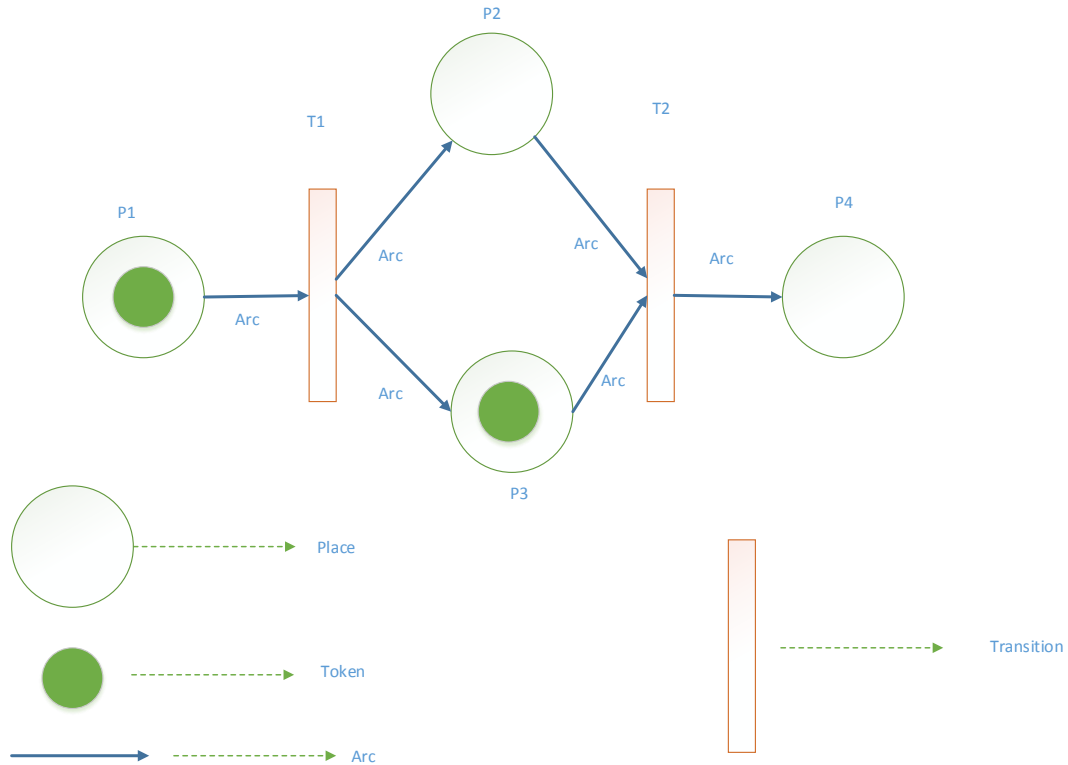


Figure 5. A Petri Net and its component parts.

Figure 5 shows a diagram of a Petri net. In the figure, circles represent places and bars represented transitions; the flow of information is represented by arcs. The dynamic of the net is represented by tokens (solid dots). The execution of tokens, known as a token game, models the dynamic of a system under construction.

A Petri net is a labeled, directed graph. A label represents all the specific information for a net. It may be associated with a node (places, transitions), arc or net. ISO/IEC 15909-2 defines UML packages for Place/Transition Nets, a package for Symmetric Nets and a package for High-level Petri Net Graphs. Symmetric Nets are formally known as Well-Formed nets and represent a subclass of High-level Petri Nets. Petri nets reflect system behavior. Transitions describe the behavior of the component.

The mathematical representation of Petri nets allows the quantitative analysis of deadlock detection. Because it can be executed, the dynamics of the system can be shown. Petri nets are considered to be a powerful modeling language [17].

Petri nets are often utilized in information systems development and system modelling. For practical applications of Petri nets, easy exchanges between different Petri net models are essential. To this end, PNML was defined by ISO/IEC 15909-2 and published on November 11, 2009 [1] to support various kinds of Petri nets. According to ISO/IEC 15909-2, PNML is defined as the transfer format between three versions of Petri nets, namely PT Nets, Symmetry Nets, and HLPN, as stated in Part 1 of ISO/IEC 15909-1. PNML supports any type of Petri net and is the XML based format for Petri nets. An XML representation is used to enable the exchange of models among different Petri net software applications.

Extensible Stylesheet Language Transformation (XSLT) supports the transformation and representation of data in XML format. It has become indispensable for XML data exchanges. XSLT consists of a set of templates. XSLT script is the specification of the output as a function of input.

2.2.1 Definition of Acronyms [Define the Terms]

The following acronyms are used in this document-

AADL: Architectural Analysis and Design Language

AADL-XML: XML version of AADL

PNML: Petri Net Markup Language

PN: Petri Nets

XSLT: Extensible Stylesheet Language Transformations

2.2.2 Design Process

XML is utilized as the interchange format for this extension of AADL to PNML. XSLT is employed as the language in order to automate the process of one XML file to another XML file. The XML version of AADL (AADL-XML) is taken as the input and PNML is the output via XSLT.

2.3 Transformation Techniques

In the transformation techniques required for this project, XML and XSLT are used. Java is also used to perform the same transformation.

2.4 Related Work

A number of studies have explored ways to extend the AADL standard to Petri nets. Part 1 of the International Standard (ISO/IEC 15909-1:2004(E)) [1] provides a formal protocol for the technique and defines the semantics and syntax used for Petri nets. This International Standard is applied in the design of systems and processes like air traffic control, banking, avionics, computer hardware architectures, distributed computing, nuclear power systems, operating systems, defense command and control, transport systems, legal processes, logistics, music telecommunications and workflow.

PNML enables compatibility and interoperability among different Petri net tools [2]. ISO/IEC 15909 defines Petri nets through a mathematical semantic model. It supports a number of different types of Petri nets. A software framework is proposed for the entire Petri net community in order to make PNML applicable and integrable at low cost. PNML relies on:

- A PNML Core Model,
- Concrete Petri net-type metamodels, and

- A Petri net-type Definition Interface.

Here, the PNML syntax is defined in terms of RELAX NG grammar. RELAX NG technology [15] is easy to handle and more flexible to use. It is a schema language for XML. The software framework is approached with the following main purposes:

- Providing tools for developers with standard APIs
- Making the ISO/IEC 15909-2 International Standard Applicable
- Easing the International Standard Applicability, and
- Enabling Petri net tool designers to add their own extensions.

PNML is intended to comprise three main capabilities: compatibility, extensibility, and applicability

Model Driven Development and model transformation have been used to introduce several successive models, with the most important being:

- Platform Independent Model (PIM): Here the model specification is not dependent on any particular platform technological requirements.[]
- Platform Specific Model (PSM): Here the model specification does take into account the specific platform utilized.[]

Model engineering techniques are concerned with model transformations. These techniques need metamodels in order to transform from a source meta-model to a target meta-model. PIMs can be transformed into other PIMs by considering the specific business concerns for a particular domain; PIMs can be transformed into PSMs by considering the platform specification; and PSMs can be transformed into PSMs by taking into account different platform specifications.

MathML[] is introduced in order to specify the annotations for higher level Petri net types other than PT(Place/Transition) Systems. These researchers structured the framework APIs into four virtual categories:

1. Create: While performing model transformation, framework based PNML models are created.
2. Save: The PNML models are written into a PNML file.
3. Load: After parsing the PNML file, the corresponding models are loaded.
4. Fetch: The user can fetch the elements.

The basic language structure of Petri nets consists of:

- a) PNTD (Petri Net Type Definition)
- b) PNML (Petri Net Markup Language), which is independent of the specific Petri net dialect.

2.4.1 Petri Net Type Definition

The Petri Net Type Definition provides the legal labels for a particular Petri Net Type. PNML provides a mechanism for defining Petri Net Types and for using labels from a conventions document, but PNTD (Petri Net Type Definition) contains additional features that are not included in PNML, including the extension of PNML via object-oriented principles. The PNML Core Model contains the basic structural definition of a Petri net as a labeled directed graph and is the primary building block upon which concrete Petri net types are defined. The basic structure of a PNML Document is defined in the PNML Core Model, which contains one or more Petri Nets. There may be graphical information with each object and this information may include its position, shape, color etc.

The important component parts of Petri Nets are places, transitions and arcs. Places contain tokens that carry data. According to the PNML Core Model it is legal to connect two places by arcs and two transitions by arcs in order to support the variations and extensions of the Petri Net types.

2.5 PNML

Petri Net formalism requires standardization in order to facilitate the work of researchers and permit data exchanges between different Petri Net tools [5]. PNML is introduced next in order to provide the syntax for the conversion of Petri net models between different applications [3].

Petri Nets are used to provide unambiguous specifications and descriptions of applications and thus support concurrency. Though PNML is an XML format of Petri Nets, it is independent from XML. A number of features make the ISO/IEC-15909-2 standard effectively a universal transfer syntax for Petri nets, including the following features:

- a) Flexibility,
- b) Universality,
- c) Mutuality,
- d) Readability,
- e) Compatibility, and
- f) Extensibility

However, there remains a challenge associated with the use of Petri nets for some systems. When Petri nets are described in the graphical form for the specification of complex systems, there are a huge number of elements involved. High Level Petri Nets

were introduced in order to overcome this by incorporating high level concepts.

Examples of High Level Petri Nets include Predicate-Transition nets and Colored Petri Nets. Standardization has been recommended as a better opportunity to improve organization in the Petri Net Community in order to address issues such as:

- The need for more support for researchers
- The need for a reference implementation capable of facilitating data exchange between various Petri net tools, and
- The need to develop future extensions on a stable common basis.

Part 1 of this standardization defines the mathematical definitions of High Level Petri Nets, while Part 2 defines a transfer format in order to support the exchange of High Level Petri Nets among different tools and Part 3 involves the standardization of Petri Net extensions. PNML consists of two aspects:

- The abstract syntax (structurally)
- The concrete syntax (textually)

The main Petri net types are defined using Unified Modeling Language. They are:

- The Core Model, which provides the foundation for further definitions of new Petri net types;
- P/T (Place/Transition) Systems metamodels built on extensions of the Core Model; and
- High-level Petri net metamodels built on extensions of P/T Systems, which require new labels and high-level functions to be defined.

Petri nets are graphical and mathematical modeling tools for processing systems or describing and studying information [12] and thus are ideally suited to visual

communication. Tokens are used in order to simulate the dynamic and concurrent activities of systems. Petri nets have a wide variety of applications due to their generality and permissiveness, especially for the analysis of large Petri Models. The graph in a Petri net consists of places, transitions and arcs. Arcs are labelled with their individual weights, and each place is assigned a non-negative integer. In modelling, places represent conditions and transitions represent events.

Colored Petri Nets are high level Petri nets and the tokens in CPN carry data. A discretization method has been used to convert CPN to Symmetric Nets [28]. Colored Petri Nets can be helpful in analyzing the modelling of a system, while Symmetric Nets are useful in representing the state space of large systems. A CPN model represents the states in a system and the events which cause the system to change its state.

2.5.1 Limitations and Benefits of PNML

Petri Net Markup Language is designed to support all kinds and variants of Petri Net types. It is categorized into specific levels: a) PNML Technology b) PNML Types and Features and c) PNML Documents. However, the main challenge facing PNML is its applicability.

2.5.2 The PNML Framework

In order to make the PNML standard applicable a PNML framework is used. This helps Petri net tools to remain updated and to comply with the standard [5]. The first release of the PNML Framework was published in March 2006. The PNML Framework has the following capabilities:

- Easy integration in Petri net tools,
- Efficient model-driven import and export tool for PNML models, and

- Standalone execution.

The PNML Framework has been designed using model engineering techniques. The flexibility of the PNML Framework facilitates the export and import of appropriate elements of Petri Net models and is designed on the basis of a model driven technique. The PNML Framework is open source software and is distributed under the Eclipse Public License.

2.6 Improving the Transition from AADL to PNML

AADL is designed in such a way that it can be extended to support other languages. It suffers from some shortcomings, including its inability to determine the behavior of the components of the system subject to deadlock, and thus cannot be used directly for safety critical systems as the correctness of the system requires a high degree of assurance that cannot be analyzed via AADL. Thus, the objective of the research reported in this thesis is to extend AADL to address this issue. In order to extend AADL to various kinds of Petri nets, an interchange format is required to translate AADL to an XML version of Petri nets, namely PNML. Once AADL has been extended to PNML, then it will be relatively straightforward to extend it to other Petri net tools.

2.6.1 Related Work

EPNML, the XML format of Petri nets, is used in many web applications [6]. Petriweb, a web application based on PNML, is used to manage Petri net repositories although it is independent of Petri net tools. PNML is defined with <pnml>, Petri nets by <nets>, places by <place>, and arcs by <arcs> (Table 2). Nets contain subnets, which are supported by EPNML. To represent a subnet EPNML uses Pages. A page occurs within a net. In order to specify the position and size of an element for layout, EPNML uses a

<graphics> element, which may include the subelements <position>, <offset>, and <dimensions>. The element <name> contains the name of the element, while the <description> element contains the description of the element. PNML specifications require an ID attribute.

Table 2. Representation of Petri Net Components [6].

Terms	PetriWeb
PNML	<pnml>
Petri Nets	<nets>
Places	<place>
Arcs	<arcs>

2.6.2 PNML Ontology

Guidelines for PNML ontology are provided in [16]. Renew uses XML in order to overcome the problem of model exchange with other Petri net types and PNML is accepted by the Petri Net Society. PNML consists of two parts, the first of which is independent of the specific Petri net dialect and the second of which is specific. The general part is referred to as PNML and the specific part as PNTD (Petri Net Type Definition). PNTD defines additional features that are not defined in PNML. PNML is used as the base for Petri Net Ontology and thus provides a mechanism for defining Petri Net types.

2.6.3 Petri Net Type Definition

XML Schema language is used for defining PNTDs. It defines the legal labels for a particular Petri Net type. In the ISO/IEC 15909 standard, for example, PNML and its

modular extension are represented as the most appropriate interchange format for Petri nets [1].

2.6.4 AADL-OSATE

AADL-OSATE was designed to specify and analyze the real time embedded systems used in safety critical systems [8]. AADL is used to analyze system-wide properties like safety, performance, reliability, security and availability. AADL-OSATE, which provides model representations for real-time embedded systems, utilizes the XML interchange format in order to allow model transformations. The formal theory of AADL is based on MetaH[], which is an extensible domain specific architectural description language. AADL and its tool OSATE support the features required for simulating, prototyping and analyzing the quality of a system. However, AADL-OSATE is not capable of analyzing the behavior of the components, and cannot detect either deadlock or livelock of the system [8]. This may result in serious problems, as the development of safety and mission critical systems requires the highest level of dependability.

In order to overcome these limitations, this project was designed to extend AADL-OSATE by incorporating Petri nets. Formal mapping rules are first established to determine the proper and precise relationship between the elements of Colored Petri nets and AADL. CPN was chosen because this is a formal modeling language and incorporates support tools such as model checker and editor. Colored Petri Nets are generally used to analyze both deterministic and non-deterministic systems and offer a useful way to identify the presence of unsafe states. Once AADL elements are mapped to CPN, the CPN models will make it possible to analyze the behavior of the system, in particular by verifying the absence of deadlocks and livelocks. Livelocks occur when a

process is starved of resources and thus is prevented from progressing, while in deadlock the processes are waiting for one another in order to start executing

2.6.5 2009 Petri Net Workshop

In 2000 the 10th Workshop on Coloured Petri Nets was held in Aarhus, Denmark [4]. As a result of the discussions held at this workshop, PNML was finally adopted as the ISO/IEC 15909-2 standard [1]. ISO/IEC 15909-2 defines a transfer syntax for High Level Petri Net graphs. Any kind of Petri net can be considered to be a labelled graph. The standard defines the transfer format of a number of different versions of Petri nets including Place/Transition nets, High Level Petri Net Graphs and Symmetric nets. All kinds of specific information of a net are represented in labels. A label may be associated with an arc, a net or a node.

2.6.6 PNML Core Model

Core concepts of Petri nets can be found in the PNML Core Model, which can be used to represent any kind of Petri net. Technically, this is a UML package. Though UML is a semi-formal modeling notation, it is utilized in this context due to its modularity, expressivity, and hierarchy. The use of the UML is suitable because it is generally accepted and standardized for analysis and modeling in software engineering [uml specific reference] and offers a convenient way to define the basic structure of Petri nets. This is a metamodel that defines the basic structure of a net graph, which is common to all versions of Petri nets. A Petri Net Document is a document that meets the requirement of a PNML Core Model. It is represented using UML class diagrams and can represent any kind of Petri net. The package PNML Core Model defines the basic structure of Petri nets.

A Petri net consists of one or more pages, each of which consists of objects. The most important objects in a Petri net are places, transitions and arcs. Places and transitions are referred to as nodes, and the nodes of a Petri net are connected by arcs. According to the PNML core model, it is legal to have arcs from one place to another place and from a transition to another transition because there are versions of Petri nets that support arcs. Figure 6 shows a schematic of a PNML Core Model.

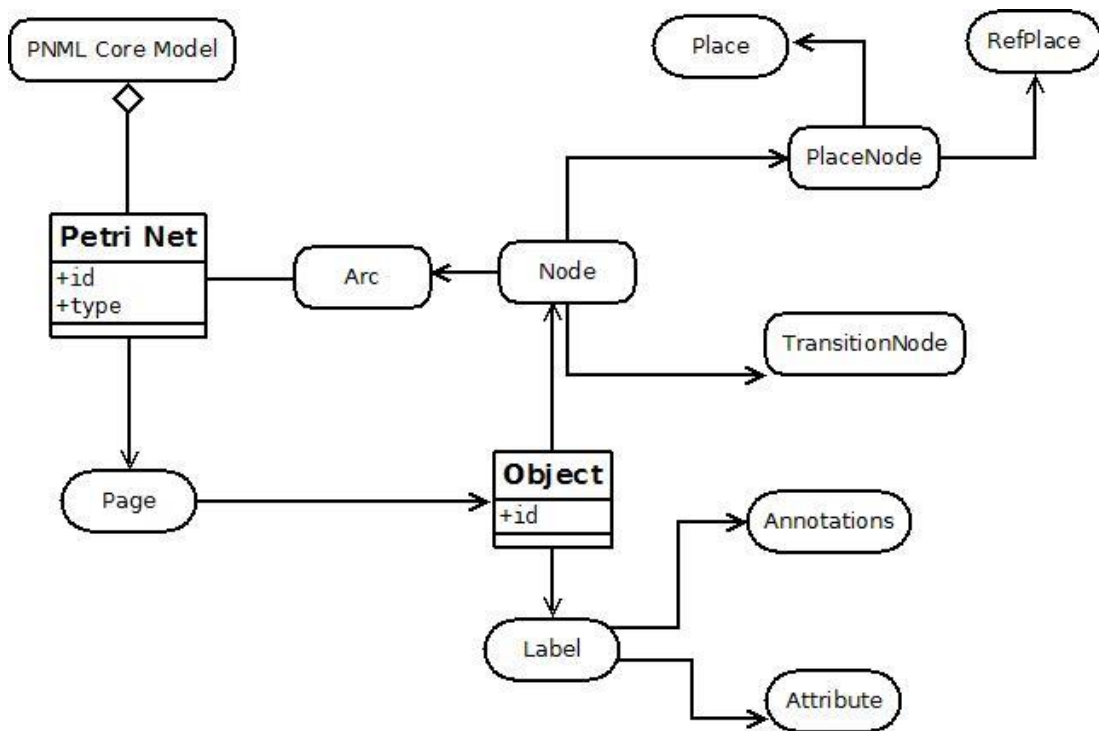


Figure 6. PNML core model.

2.6.7 Mapping of PNML Core Model to XML

Each concrete class in the PNML Core Model is mapped to an XML element. These XML elements are also called PNML elements. Mapping examples are shown below in Table 3.

Table 3. Mapping of PNML Core to XML Element [1].

Class	XML Element
Place	<place>
Transition	<transition>
Arc	<arc>
Petri Net	<pnml>

Structuring a Petri Net involves three objects: Pages, Reference places and reference transitions. A page can contain other pages, but an arc cannot connect nodes on different pages. The Petri net labels are distinguished as:

1. Annotation: Names, Initial Marking, Arc Inscriptions and transition guards are in this category. Annotation is represented as text near the object.

2. Attributes are represented in the form, style or color of the object. An attribute has its impact on the shape of the object.

2.6.8 Summary of the Project

Although AADL has proven to be an efficient way of determining the non-functional properties of the components of a system, it is not capable of determining its functional properties. Safety critical systems require a higher degree of safety assurance, so it is therefore essential to extend AADL to PNML in order to incorporate Petri Nets (mathematical modeling language) and thus gain this additional functionality. It is difficult to transform AADL to Petri nets directly and the use of an interchange language is proposed in order to ease the transformation. As manual conversion of this translation is both error prone and tedious, XSLT has been chosen to automate the process.

2.6.9 Development Process

The development and standardization of conventions is an ongoing process and requires further research.

CHAPTER III

METHODOLOGY

AADL is extended to PNML via a series of steps. Part 2 of the International Standard defines PNML as the preferred transfer format for High-level Petri Nets in order to support the exchange of High Level Petri Nets between various tools [1]. PNML is known for its universality and interoperability between different Petri net tools.

AADL is designed such a way that it can be extended to support other languages. The process begins when the AADL text version is transformed to the AADL-XML version. Simultaneously, the Petri net (PN) is converted to PNML, which is the XML format for Petri nets. AADL-XML is then transformed to the PNML version. In order to automate the process, XSLT script is employed. In the case study presented in Chapter IV, one of the hardware components of AADL is shown to illustrate the conversion of AADL text to PNML via XSLT script. Once the transition is made to PNML, it is then straightforward to transition from One Petri Net to another.

3.1 Description of Methodology

There are several important issues that must be considered when performing a transfer to a Petri Net format. It is difficult to convert AADL to various types of Petri nets without passing through any interchange format, so an appropriate interchange format must be chosen to perform the conversion. XML was chosen as the interchange format for this study since it is platform independent and Petri nets can be readily converted to XML format. This allows AADL XML to be mapped to the corresponding PNML

format, which then facilitates the transfer of one Petri net to another. XML is a widely used format. The methodology can be broken down into two distinct steps:

1. AADL text to AADL- XML
2. AADL-XML to PNML

This is followed by the conversion of AADL-XML to PNML via XSLT in order to automate the process.

Figure 3(b) gives a broad overview of what the methodology entails. Step 1 consists of the conversion of AADL text to AADL-XML, and Phase 2 comprises the conversion of AADL-XML to PNML-XML via the mapping rules.

3.2 Steps of the Methodology

In this section the steps involved in the translation of AADL- text to the xml version of PNML will be discussed in more detail. The AADL text version is first translated to the XML version of AADL, AADL-XML, to ease the translation. The resulting AADL-XML text is then mapped to PNML via the standard mapping rules.

Step 1:



Figure 7. The transformation of AADL to AADL-XML.

There are two possible versions of AADL, text and graphical, and the same procedure is followed for both. Figure 7 shows the process used to transform the textual

version of AADL into the XML format, which is the conversion of AADL text to AADL-XML.

The Mapping rules that are used in the Transformation of AADL-text to AADL-XML are as follows:

AADL -text	AADL-XML
Wheel_rotation_sensor	“wheel_rotation_sensor”
Wheel_pulse	“wheel_pulse”
Features	<Features> </Features>
Device	DeviceType name
data port	DataPort name
in	Direction=“in”
out	Direction=“out”

Step 2:



Figure 8. The transformation of AADL-XML to PNML via mapping rules.

After the AADL text has been translated to AADL-XML, AADL-XML is mapped onto PNML using the mapping rules for the conversion of AADL to Petri nets (Figure 8).



Figure 9. The conversion of AADL-XML to PNML via XSLT.

XSLT may also be used to convert AADL-XML to PNML-XML.

3.3 Algorithms

The textual version of AADL is transformed into the XML version of PNML using the following strategy.

3.3.1 AADL Conversion

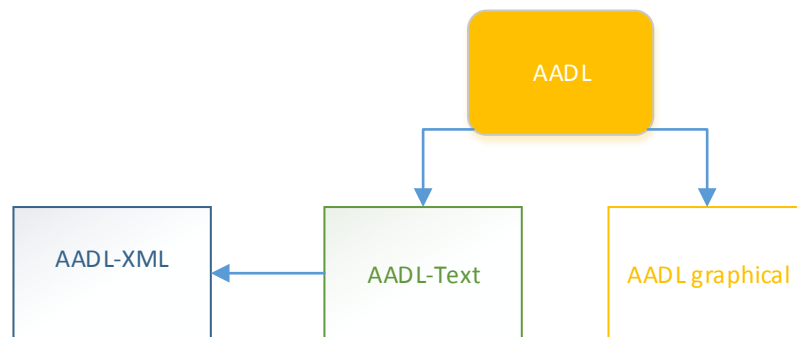


Figure 10. AADL text conversion.

AADL can be represented by either AADL text or AADL graphical notations.

AADL text can be converted to its XML format – AADL-XML (Figure 11).

3.3.2 PN Conversion

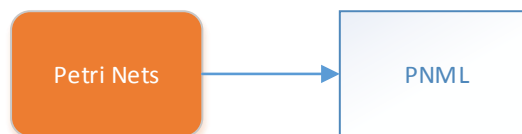


Figure 11. PN conversion.

The XML format of Petri Nets takes a standard format known as Petri Net Markup Language (PNML). The conversion process is shown in Figure 12.

3.3.3 Mapping



Figure 12. AADL-XML conversion to PNML.

AADL-XML is mapped to PNML using the following mapping rules:

Device \leftrightarrow transition

Port \leftrightarrow Place

For example, in the case of a wheel rotation sensor the mapping rules used for the translation of AADL-XML to PNML are:

Device name Type \leftrightarrow transition id class="data_flow"

Port name Type \leftrightarrow place id class="data_flow"

Table 4 shows how individual AADL components of AADL are mapped to the elements of a Petri Net.

Table 4. AADL to PN Mapping.

<u>AADL</u>	<u>PN</u>
In or Out Data Port	Place
In or Out Event Port	Place
Port Group	Place
Event Data Port	Place
Data Access	Place

Table 4. cont.

<u>AADL</u>	<u>PN</u>
System	Hierarchy transition or transition
Process	Hierarchy transition or transition
Thread	Transition
Device	Transition
Memory	Transition
Arc	Connection

The following scheme shows the steps involved in converting AADL to PNML for a Device “Component”:

AADL text (see also Figure 17 in Chapter 4):

```

Device wheel_rotation_sensor
  Features
  Wheel_pulse: out data port;
End wheel_rotation_sensor;
    
```

XML version of AADL Text (see also Figure 18 in Chapter 4):

```

<device Type name="wheel_rotation_sensor">
  <features>
    <data Port name="wheel_pulse"
      Direction="out">
    </features>
  </device>
    
```

PNM version of the above (see also Figure 19 in Chapter 4):

```

<transition id="wheel_rotation_sensor" class="data_flow">
  <place id="wheel_pulse" class="data_flow">
  </place>
</transition>
    
```


3.3.4 Petri Net Markup Language

PNML was designed based on the following properties: [2, 7]

- (a) Universality: It should be general enough to support any kind of Petri net with any kind of extension.
- (b) Readability: It should be convenient to read.
- (c) Mutuality: It should be able to extract information such as common principles and notations for Petri nets.
- (d) Extensibility: It should be extendable to future versions of Petri nets, apart from the types of Petri nets mentioned in Part 1 of the International Standard.
- (e) Compatibility: It should be compatible with all types of Petri nets.
- (f) Applicability: Engineers should be able to develop Petri net tool interfaces using any other high-level programming language.

Though AADL-XML is an XML-based format, it is independent from XML.

PNML conveys the structural information for Petri nets. PNML is standardized to enable the conversion from one Petri net type to another without loss of information. PNML relies on XML technology. The XML version of a Petri net is employed in [23], where it is used in a software tool to convert a Petri net to ruleML, the rule markup language, and then applied in a rule-based system. RuleML is the XML version of rule and is based on knowledge representation for knowledge-based systems. The PNML transfer format was designed to permit a convenient interchange of variants of Petri nets between different Petri net tools.

3.3.5 XSLT

XSLT facilitates the conversion of languages in their XML format. XSLT script is used in this study in order to automate the conversion of AADL-XML to PNML. The components which comprise XSLT are the templates. XSLT is used for transforming one XML document to another XML document. XSLT is used as a part of XSL, which is used for styling XML documents. The mechanism which processes the conversion of one XML document to another is the XSLT processor. More about XSLT is discussed later.

3.3.6 High Level Petri Nets

The basic features of a High-level Petri Net are the annotations of arcs, transitions and arcs. A term which is associated with the place denotes the initial marking and it must be associated with an appropriate term that defines the type of the tokens at a particular place. The term associated with an arc determines which tokens are to be added or removed on the firing of transitions. Each place in a Place/Transition (Petri) net is labeled with a natural number that indicates the initial marking. Each arc is labeled by the arc annotation, which denotes the arc weight. Examples of High Level Petri Nets include Predicate-Transition Nets and Colored Petri Nets [1]

3.3.7 Data Types

The data types used in different versions of High Level Petri Nets include: dots, Booleans, partitions, multisets, integers, strings, and lists, which may be finite enumerations, cyclic enumerations or finite integer ranges.

3.3.8 PNML Framework

The PNML Framework, released in 2006, is the API framework for standard types of Petri nets and is built following the Model Driven engineering technique .It is

introduced here in order to make use of the PNML standard applicable through automation.

3.3.9 Applications of PNML Frameworks

An API based framework helps tool developers to achieve conformance with the PNML standard [1]. This API framework is compatible with and works alongside the PNML standard. To support the wider application of PNML, its low-cost integration into the Petri net tools PNML framework is recommended. It relies on multi-platform technologies such as the Eclipse Modeling Framework, which runs on the Eclipse Java based platform. The Eclipse Modeling Framework (EMF) supports UML, code generation and model transformation and was designed to address several important issues, specifically:

1. To facilitate the applicability of PNML over a wide range.
2. To provide standard APIs for tool developers.

The developers of Petri net tools utilize API frameworks as a convenient way to support PNML documents. This in turn supports different kinds of Petri nets as this approach is based on a structured set of metamodels issued from the PNML standard and is designed to facilitate the export and import of Petri Net models. The framework provides a set of APIs to read and write PNML files and is implemented in Java. Its flexibility and ability to evolve supports the further development of the PNML standard. PNML framework code is automatically generated; an API is automatically generated from the PNML metamodels in EMF.

3.4 Development

Phase 1 of this methodology involves the description of the Conversion of AADL text to AADL- XML. The major aspect of Phase 1 is the transformation of Architecture Analysis Design Language from its textual version to an XML version. This conversion is required since it is difficult to convert AADL to Petri nets directly. XML was chosen as a convenient interchange format that acts as a bridge to facilitate the conversion of AADL to PN. The XML representations are specifically defined to ease the interoperability between tools.

3.4.1 The Aarhus Petri Net Workshop

In 2000, a workshop was held in Aarhus, Denmark, to formulate a definition for the standard transfer format of Petri nets [4]. It synthesized the independent work of a number of different research groups on XML-based Petri net formats and initiated the development of a standard interchange format for Petri Nets. The attendees were faced with several problems in trying to define an interchange format for Petri nets:

- Each Petri net tool had its own file format. A different parser was needed to implement each, which was tedious. There was no generally agreed file format for such a standard.
- Each Petri net tool supported different versions of Petri nets.

The decision to utilize XML sidestepped the above issues and the existing XML tools reduced the cost of the implementation of XML based interfaces. This made it possible to propose a document description of Petri nets [47] by treating them as having:

- a) a general part, which is independent of the specific type of Petri net, and
- b) a specific part, which is specific to the Petri net version.

The general part is known as Petri Net Markup Language (PNML) and the specific part is the Petri Net Type Definition (PNTD). PNML defines the features of all Petri Nets, while PNTD defines additional features which are not captured by PNML. PNTD defines the special features of a particular Petri net type. The general interchange format is expected to support all features of Petri net tools, both those already in existence and those yet to be developed.

Typically a Petri net consists of places, transitions, and arcs, where places are associated with marking. The typical features of Petri nets can now be extracted into PNML and the resulting PNML forms a uniform file structure for all kinds of Petri nets. PNML makes it possible to exchange nets among different compatible Petri net types. Figure 13 shows a representation of PNML.

```
<place id="P1">
  <name>A</name>
  <marking const="M">
  </place>

  <transition id="T1">
  <name>T</name>
  </transition>

  <arc id="A1" source="P1" target="T1">
  <annotation>
  <var ref="x">
  <var ref="y">
  </annotation>
  </arc>
```

Figure 13. PNML.

There are several reasons for choosing PNML for this project, the most important being that it is a convenient way to exchange content-only for non-graphic tools. Here, AADL-XML is treated as the input file and the PNML version is the output file.

AADL describes the architecture of the software and hardware components of a system and its functional interfaces [37]. AADL is designed to be extensible in order to accommodate analyses which the core language does not support. AADL is particularly efficient in architectural modeling and AADL models can be transformed into GSPN (Generalized Stochastic Petri Nets).

3.4.2 Extensions of AADL

AADL is used to describe functional interfaces and performance-critical aspects of components [15]. Here, AADL is transformed into BIP (Behavior Interaction Priority). BIP is a language used for the description and composition of components, as well as associated tools for analyzing models. BIP provides a framework for modeling heterogeneous real-time components. A general methodology is provided for translating AADL models into BIP models.

3.4.3 Combining AADLs with High Level Petri Nets

AADL is combined with High Level Petri Nets in order to perform model based testing. Traditional software testing suffers from interoperability requirements. Formal methods must be employed in order to overcome the limitations and improve the reliability and confidence of software products. Software testing is the final step before the software is released, and companies invest considerable amounts of time, effort, and money in their software testing activity. Safety critical systems require greater cost and attention in software testing since they require a higher degree of dependability. Many

techniques have been suggested to improve this process including, for example, architectural-based testing, model-based testing and hybrid-based testing. Of these, hybrid-based testing combines many of the benefits of formal verification testing. AADL is used to specify the system at the architectural level and then mapped to Petri nets. High Level Petri Nets have proven to provide an effective modeling notation for describing a system at different levels of abstraction.

Petri Net Models are built from AADL in order to evaluate the behavior or schedulability of Distributed Real-Time Embedded systems in order to check for deadlock [13] in particular. This objective of this study is to take advantage of the benefits offered by both AADL and Petri Nets for software testing by building an XML bridge between them to facilitate the conversion process. Since the qualitative analysis of AADL specifications is required, Symmetric Petri Nets are selected as being the most appropriate.

3.4.4 Strengths and Weaknesses of AADL

AADL is used to analyze the non-functional properties of a system but is unable to determine the functional properties of a system. The correctness of a system is of crucial importance in safety-critical systems, so this project aims to extend AADL to incorporate PNML functionality. PNML was selected as the interchange format due to its universality. Once AADL [11] can be transformed to PNML, then it can be easily transformed into any kind of Petri net.

3.5 Description of the Proposed Model

AADL will be converted to a Petri net via an interchange format, in this case XML, as it offers a convenient way to ease the interoperability between the tools

involved. The process begins by converting AADL to AADL-XML and the Petri net to PNML, the XML format for Petri nets. Once this is achieved, the resulting Petri net that is now in PNML can readily be transformed to another type of Petri net.

3.5.1 Algorithm Explanation with Diagram

AADL-XML is converted to PNML using the standard mapping rules. XSLT is utilized in order to automate the conversion of AADL-XML to PNML.

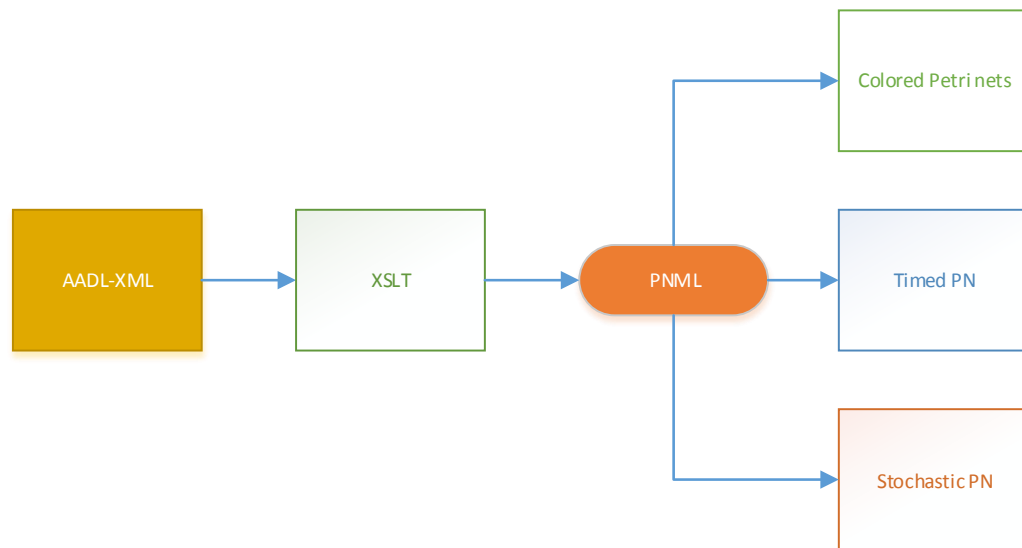


Figure 14. AADL-XML to PNML - architecture and automated version.

Figure 14 shows the architecture involved in moving from AADL-XML to PNML and from PNML to different Petri nets via the automated process based on XSLT. The XML version of AADL is translated to PNML via mapping rules and in an automated way through XSLT to make it easier to then switch between different Petri net tools once it has been translated to PNML. As the figure shows, AADL-XML is first translated to PNML and then once in PNML it can be translated to Timed Petri Nets, Stochastic or Colored Petri Nets relatively easily.

3.6 Summary of Methodology

In this methodology, the major aspects were divided into three phases, the conversion of AADL Text to AADL-XML, the conversion of Petri nets to Petri Net Markup Language, and the mapping of AADL-XML to PNML. This process can be automated by invoking another step, namely the conversion of AADL-XML to PNML via XSLT. Chapter 4 will present a case study in which the phases outlined in the methodology described in this chapter will be applied to a theoretical example as a proof of concept.

CHAPTER IV

CASE STUDY

This chapter presents a case study that demonstrates how one component of AADL is converted to PNML. The transformation is shown in several steps. The textual version of AADL is first converted to its XML version. The XML version of AADL is then transformed into the XML version of a Petri net. XSLT script is utilized for this transformation, taking as its input the XML version of AADL and generating as output the corresponding XML version for a Petri net, namely PNML. In this case, an automobile cruise control system is considered as the example to demonstrate how one component of a cruise control system is translated into the XML version of a Petri net in a step by step process. The “wheel rotation sensor” device is at the center of the process as in this case the concept “device” can represent the abstract version of a complicated system. Communication with the device consists of data and control commands.

4.1 Case Elements

This case study focuses on the operation of a vehicle cruise control system. To make the case more manageable, only a part of the cruise control system is used. It is translated to the XML version of PNML using the mapping rules discussed in the previous chapter. The “wheel_rotation_sensor” device is the part of the cruise control system that is used as the subject of this case study.

The cruise control system controls the speed of an automobile automatically. It is used by drivers in the case of steady traffic conditions such as those typically encountered

when driving long distances on an interstate. The cruise control system takes over the throttle of the car and helps to maintain a set speed for the vehicle that is specified by the driver. When the driver applies the brakes, the cruise control disengages.

The components of a cruise control system are shown below in Figure 15.

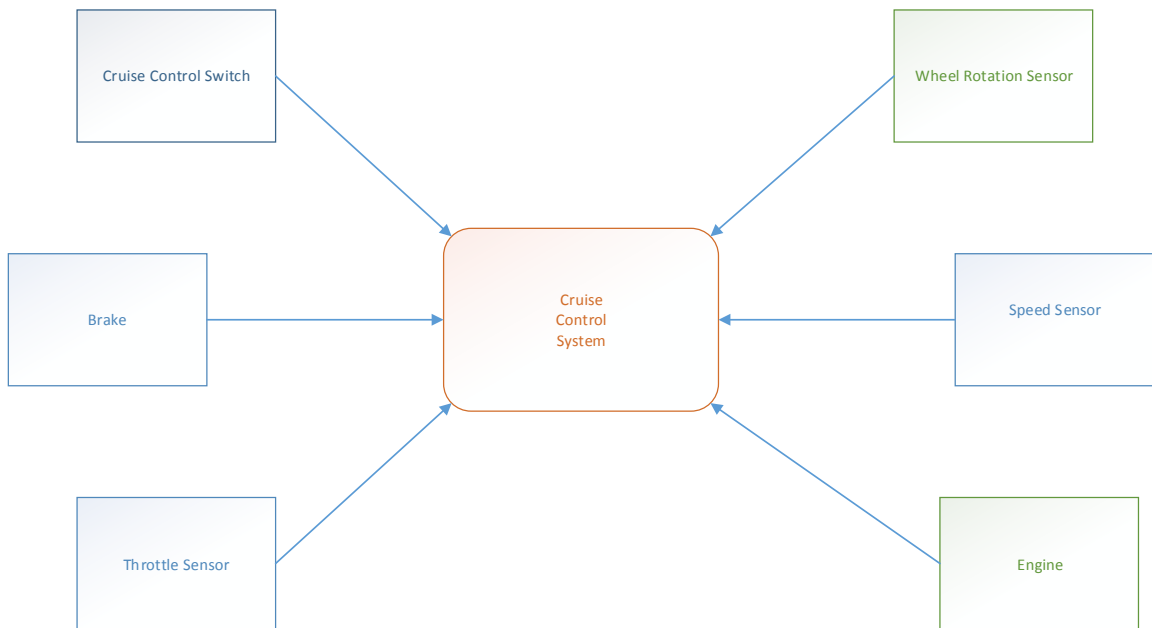


Figure 15. Components of a cruise control system.

The main components of a cruise control system are:

1. Cruise Control Switch
2. Brake
3. Engine
4. Wheel Rotation Sensor (Device)
5. Throttle Sensor
6. Speed Sensor

Cruise control switch. This comprises the cruise control “on/off” button, the “resume” button, and the “accelerate” and “set” options. Once a certain speed is attained in a vehicle and the cruise control switch is activated, then the vehicle is expected to run at a steady speed set by the driver.

Brake. The braking system deactivates the cruise control system. When the driver hits the brake, the vehicle slows down and no longer maintains the same steady pace. In order to resume the same speed the “resume” option must be activated.

Wheel rotation sensor. The “wheel rotation sensor” monitors the wheel rotation.

Device. An AADL device is generally used to represent a sensor. Here, “device” represents the abstraction of one or more complex entities. Data and data communication are both modelled via AADL.

Throttle sensor. The throttle sensor represents an electronic device. It is capable of receiving a signal representing the relationship between fuel and engine function.

“Device” can represent single function components and also more complicated components. Single function components include sensors (such as the `wheel_rotation_sensor`), while complicated components include GPS (Global Positioning System). A device thus represents those components that have an interface with the external environment. Communication of data from a device is configured via the “in data port” and “out data port”. A device interacts with the software and execution components of AADL via physical connections to processors and logical connections to applications such as software components.

In this case, one of the hardware components, “Device”, is used for the case study. In a series of steps (shown earlier in Figures 7, 8 and 9 in Chapter 3), the element

is transformed to the PNML version. The textual version of the “Device” component of AADL is transformed into Petri Net Markup Language. The component is first transformed into the XML version from the AADL textual version, after which it is mapped to the corresponding XML version of the Petri Net Core Model. XSLT is then used to transform the XML version into other XML versions.

In this case study, the “wheel_rotation_sensor” device is translated into the XML version of PNML. In the “wheel_rotation_sensor”, the wheel rotation is declared by wheel pulses. Data type is not declared in this scenario.

4.2 Analysis of the Case Study

AADL syntax represents the components, connections and behavior of a system. In this case study, the “Device” component is the focus of attention. Components have a component type that represents the component specification or the interface of the component. These can include, for example, “features”, “flows”, and “properties.” Device represents sensors, actuators, or other components. Devices have an interface with the external environment. “Features” are used to exchange control and data with other components through connections. Components contain “data ports”, which can be either in data ports or out data ports. Data is communicated to or from the “device” via the in data port or the out data port, respectively. Figures 17 through 19 show the transformation of the textual version of AADL (Figure 17) into the XML version of the Petri net (Figure 19). The hardware component “Device” is shown to be mapped to the corresponding XML version of PNML. In this case, the device is “wheel_rotation_sensor”. At this point, the data type is not known but untyped data declarations are supported.

The textual version of the “device” component of AADL, namely, wheel_rotation_sensor, is shown in figure 16.

```

Device wheel_rotation_sensor

Features

Wheel_pulse: out data port;

End wheel_rotation_sensor;

```

Figure 16. AADL text of device “wheel rotation sensor”.

The wheel rotation is modeled by the “device” (wheel_rotation_sensor). The corresponding xml version of the “Device” component of AADL (wheel_rotation_sensor) can be represented figure 17 using mapping rules in table X1.

```

<?xml version="1.0" encoding="UTF-8"?>
  <Test>
    <DeviceType name="wheel_rotation_sensor" />
    <features>
      <DataPort name="wheel_pulse" Direction="out" />
    </features>
  </Test>

```

Figure 17. XML version of AADL text for “device”.

Table 5. Mapping of AADL Text Elements to AADL-XML Components.

AADL -text	AADL-XML
Wheel_rotation_sensor	“wheel_rotation_sensor”
Wheel_pulse	“wheel_pulse”
Features	<Features> </Features>

Table 5. cont.

AADL -text	AADL-XML
Device	DeviceType name
data port	DataPort name
in	Direction="in"
out	Direction="out"

After the applying mapping rules from AADL to the Petri net, the corresponding PNML version is shown in Figure 18 below. The “Device” component is mapped to the “transition” component in the Petri net. In and out data ports are mapped to “place” in the Petri net. The mapping rules are shown below for the translation in this case study.

According to the mapping rules, “Device” in AADL corresponds to “transition” in the Petri net. Similarly, “in data port/out data port” corresponds to “place” in transition.

Table 6. Mapping of Major AADL Elements to PNML.

AADL-XML	PNML
DeviceType name	Transition id class="data_flow"
DataPort name	Place id class="data_flow"
SystemType name	Transition id class="data_flow"
MemoryType name	Place id class="data_flow"
ThreadType name	Transition id class="data_flow"
ProcessorType name	Transition id class="data_flow"
BysType name	Arc id class="data_flows"

After applying mapping rules from AADL to the Petri net, “Device” can be represented by Figure 18.

```
<?xml version="1.0" encoding="utf-8"?>
<transition id="wheel_rotation_sensor" class="data_flow">
  <place id="wheel_pulse" class="data_flow" />
</transition>
```

Figure 18. PNML version of “device”.

The graphical representation of the above example is as follows:

Device is the “wheel_rotation_sensor” in the following diagram, represented by transition in the Petri net. “Wheel_pulse” is represented by the place.

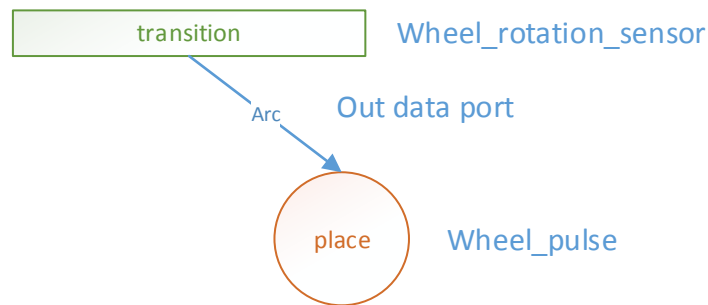


Figure 19. Graphical version of the translation.

4.2.1 Description of XSLT and How XSLT Works

XSLT (Extensible Stylesheet Language Transformations) is used for transforming one XML document to another XML document. Tree of nodes are generated as output. It uses the pattern matching template mechanism [49].

In the first step the XML document which is the input file (in our context id AADL-XML) and XSLT code are fed to the XSLT processor. It builds a source tree using Xpath from the input XML document. In the next step the source tree’s root nodes

are processed. Now, as it works with the pattern matching template mechanism it finds a matching template for that node in the style sheet. It proceeds to the next step after evaluation. In the following step it either creates nodes in the source tree or more nodes are processed as the source tree. XSLT uses XPath [50] in order to identify the subsets of the source document tree. Figure 20, 21, and Figures 22 show the example of XSLT conversion. More specifically, Figure 20 shows the original XML version of a document used as an input to XSLT. Figure 21 shows the XML template used to convert XML input to output. Figure 24 shows the output of the conversion. The figure X1 shows how the architecture of XSLT.

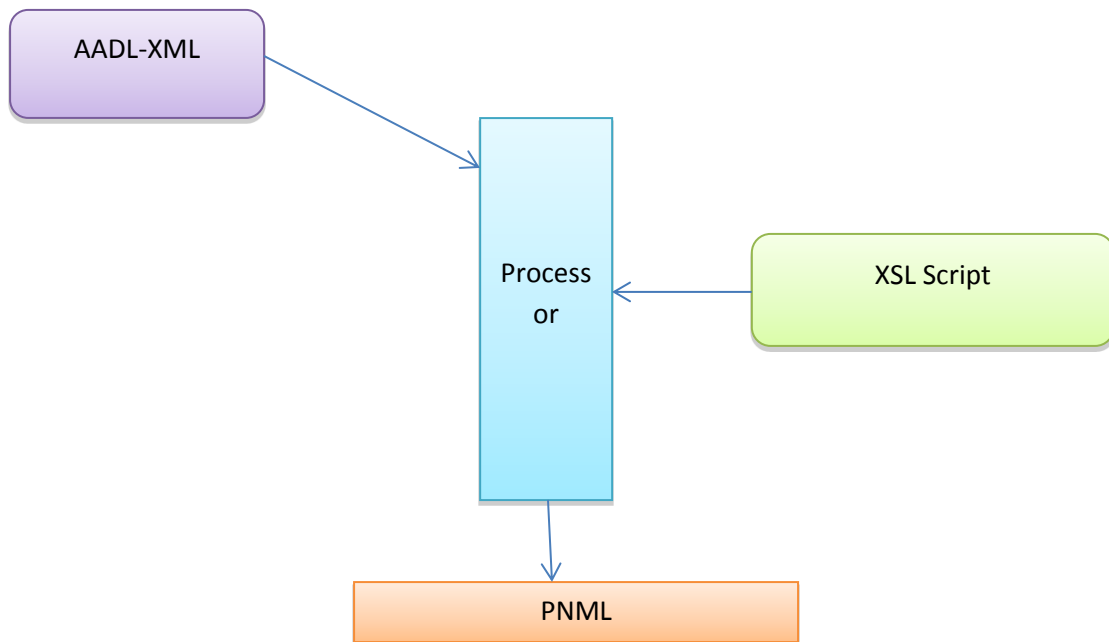


Figure 20. The structure of how the XSLT works.

```

<?xml version="1.0" encoding="UTF-8"?>
<Test>
  <DeviceType name="wheel_rotation_sensor" />
  <features>
    <DataPort name="wheel_pulse"
Direction="out" />
  </features>
</Test>

```

Figure 21. XML input (AADL-XML).

The XSLT stylesheet:

```

<?xml version="1.0" encoding="iso-8859-1"?>
<!-- Edited by XMLSpy® -->
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:template match="/">
    <xsl:element name="transition">
      <xsl:attribute name="id">
        <xsl:value-of select="Test/DeviceType/@name"/>
      </xsl:attribute>
      <xsl:attribute name="class">data_flow</xsl:attribute>

      <xsl:element name="place">
        <xsl:attribute name="id">
          <xsl:value-of select="Test/features/DataPort/@name"/>
        </xsl:attribute>
        <xsl:attribute name="class">data_flow</xsl:attribute>
      </xsl:element>
    </xsl:element>
  </xsl:template>
</xsl:stylesheet>

```

Figure 22. XSL script.

The Resulting XML document is as follows:

```

<?xml version="1.0" encoding="utf-8"?>
<transition id="wheel_rotation_sensor"
class="data_flow">
  <place id="wheel_pulse" class="data_flow" />
</transition>

```

Figure 23. Output XML (PNML).

4.2.2 Application of XSLT Script

XSLT is used for the conversion of one language in XML format to another language in its XML format. It can be employed for the conversion of AADL-XML to PNML (the XML format of Petri nets). The XSLT script takes the XML version of the “Device” component and generates the corresponding XML version of PNML.

```
<?xml version="1.0" encoding="iso-8859-1"?>
<!-- Edited by XMLSpy® -->
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:template match="/">
    <xsl:element name="transition">
      <xsl:attribute name="id">
        <xsl:value-of select="Test/DeviceType/@name"/>
      </xsl:attribute>
      <xsl:attribute name="class">data_flow</xsl:attribute>

      <xsl:element name="place">
        <xsl:attribute name="id">
          <xsl:value-of select="Test/features/DataPort/@name"/>
        </xsl:attribute>
        <xsl:attribute name="class">data_flow</xsl:attribute>
      </xsl:element>
    </xsl:element>
  </xsl:template>
</xsl:stylesheet>
```

Figure 24. XSLT script converting AADL-XML of “device” to PNML.

In the above XSL file, a template is created. For the “DeviceType” it generates as “transition id” followed by the class which is “data_flow”, for “DataPort” it generates as “place id”. According to the mapping rules (see TableX). Mapping of AADL-XML to PNML, the rest of the transformations will be done.

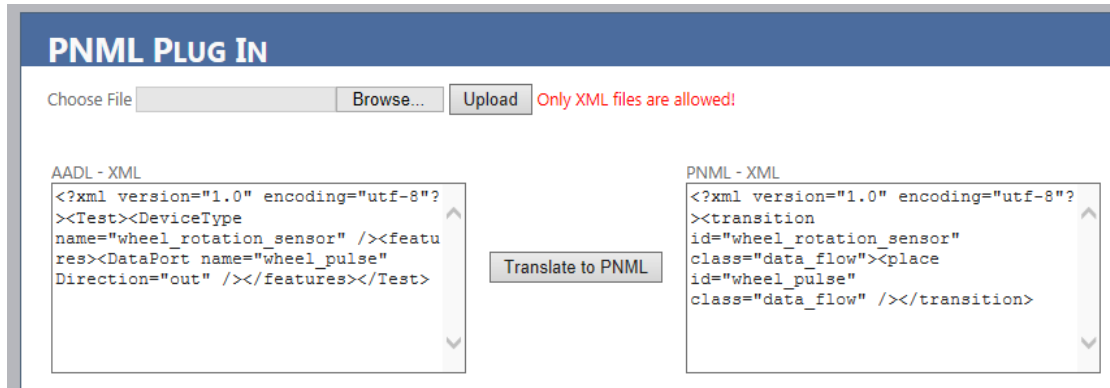


Figure 25. Screenshot of conversion of AADL-XML to PNML via XSLT.

The above screenshot shows the conversion of AADL-XML to PNML. On the left panel of the screen, AADL-XML is taken as the input and the PNML version is generated as the output. The AADL-XML text is translated to the XML version of PNML. This is done by the XSLT script on the right panel. Figure 26 shows the Graphical Representation of the Device named “wheel rotation sensor”.

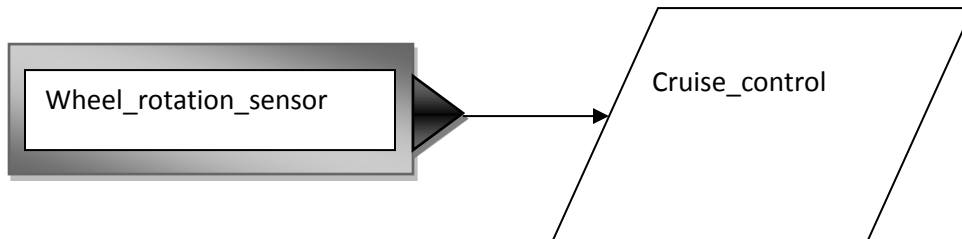


Figure 26. The AADL Graphical Representation of Device.

4.3 Results of the Case Study

The above case study demonstrates how one of the devices that make up the Cruise Control System is mapped from its original AADL textual version to the XML version of PNML using the standard mapping rules. Once the component is translated into PNML, it can then be easily translated into other types of Petri nets (for example, Place/Transition Nets, Stochastic Petri Nets, or High-level Petri Nets). However, some limitations remain to be addressed. PNML is known for its universality and

interoperability, but it cannot yet support the exchange of Petri net tools for every kind of Petri net. In the case study, one component of AADL is declared and translated into the XML version of PNML. If all the AADL components of the cruise control system are translated to PNML in the same way, then this would make it relatively straightforward to transform the components from one kind of Petri net to another. This methodology is shown in order to show the importance role played by XML in this translation.

Using PNML, it should now be possible to apply various types of Petri net modeling and tool reasoning related to system invariance and safety. These analyses could consist of simulations of the net and/or the ability to perform various types of analytical reasoning. This is because PNML is designed to support different variants of Petri nets based on its universality, flexibility, and extensibility. Figure 27 shows the complete process of converting AADL to PNML.

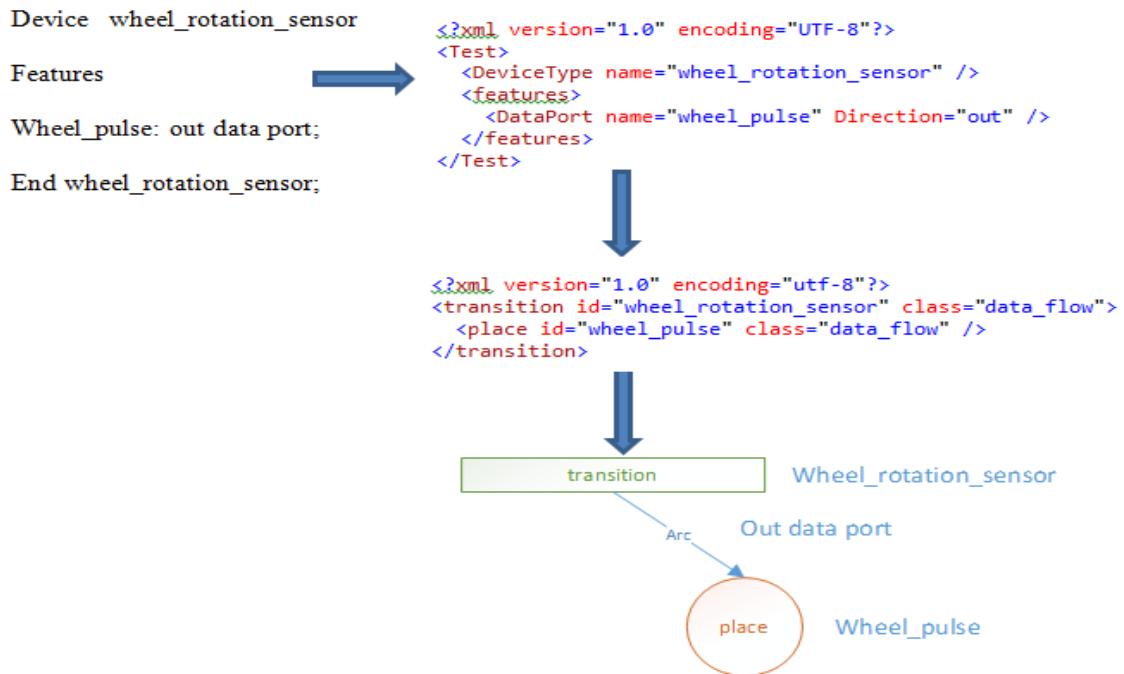


Figure 27. The whole transformation from AADL-Text to PNML.

CHAPTER V

CONCLUSIONS

A new paradigm for the conversion of AADL to Petri nets has been presented in this paper. In addition to their growing popularity due to the utility of their graphical notation in applications such as simulating dynamic systems, Petri nets can be used to determine the correctness of a system, which is not possible in AADL. In order to support this property of Petri nets, the main objective of this research was to extend the AADL - OSATE by incorporating Petri net functionality to facilitate the process of verifying the absence of deadlock. Since AADL cannot be converted to Petri nets directly without going through an interchange format, the XML version of Petri nets, PNML, was invoked. By first converting AADL to AADL-XML, its components can be mapped to their respective PNML elements. Once the PNML equivalent is created, this can be easily converted to different kinds of Petri net, thus permitting the incorporation and integration of discretized Petri net models with complex specifications.

PNML was chosen as the transfer format for Petri nets due to its universality. However, the XML format of Petri nets used for the manual conversion of AADL to PNML is error-prone and tedious, and would thus benefit considerably from automation. However, the main limitation of PNML is its applicability and tooling [2].

5.1 Future Work

The results of this study suggest several potentially very fruitful directions for future research. For example, research on the design of a PNML plug-in that can be incorporated in the OSATE environment is clearly indicated. Once the AADL text is given as input for the whole system, the corresponding PNML version should be generated automatically as output. In the future the addition of a Model Checker to analyze or verify the behavior of the system could prove very useful to facilitate the process of verifying the correctness of the system.

APPENDICES

APPENDIX A

AADL SPECIFICATION

```
AADL_specification::={AADL_global_declaration|
AADL_declaration}+
AADL_global_declaration::=package_spec|property_set
AADL_declaration::=component_classifier
|port_group_classifier|annex_library
Component_classifier::=component_type|
component_type_extension|
component_implementation|
component_implementation_extension
Port_group_classifier::=port_group_type|
port_group_type_extension
Component_type_extension::=component_category
Component_category::=software_category|
execution_platform_category|composite_category
Software_category::=data|subprogram|thread|
thread_group|process
Execution_platform_category::=memory|processor|
bus|device
Composite_category::=system
```

[16] suggest the use of PNML as a basis for Petri net ontology. PNML contains all the concepts found in each format. PNML is independent of specific Petri net dialects and contains most of the Petri net bases. This XML based interchange format of Petri nets provides interoperability between various Petri net tools. Based on these features of PNML, ISO/IEC is planning to introduce Part 3 in the near future, which will introduce more Petri nets.

APPENDIX B

JAVA PROGRAMS

Java program to convert AADL XML → PNML for the above translation.

The program will take AADL-XML as input and will generate PNML as output:

Place.java

```
package xmltest;

public class Place {

    private String id;
    private String className;

    public String getId()
    {
        return id;
    }

    @javax.xml.bind.annotation.XmlAttribute
    public void setId(String id)
    {
        this.id = id;
    }

    @javax.xml.bind.annotation.XmlAttribute(name
    = "class")
    public String getClassName()
    {
        return className;
    }

    public void setClassName(String className)
    {
        this.className = className;
    }
}
```

Transition.java

```
package xmltest;

@javax.xml.bind.annotation.XmlRootElement
public class Transition {

    private String id;
    private String className;
    private Place place;

    @javax.xml.bind.annotation.XmlAttribute
    public String getId()
    {
        return id;
    }

    public void setId(String id)
    {
        this.id = id;
    }

    @javax.xml.bind.annotation.XmlAttribute(name
    = "class")
    public String getClassName()
    {
        return className;
    }

    public void setClassName(String className)
    {
        this.className = className;
    }

    public Place getPlace()
    {
        return place;
    }

    public void setPlace(Place place)
    {
        this.place = place;
    }
}
```

Device.java

```
package xmltest;

@XmlRootElement
public class Device {

    private String typeName;
    private java.util.ArrayList<DataPort> features;

    @XmlAttribute
    public String getTypeName()
    {
        return typeName;
    }

    public void setTypeName(String typeName)
    {
        this.typeName = typeName;
    }

    @XmlElementWrapper
    @XmlElement(name = "dataPort")
    public java.util.ArrayList<DataPort> getFeatures()
    {
        return features;
    }

    public void setFeatures(java.util.ArrayList
    <DataPort> features)
    {
        this.features = features;
    }
}
```

DataPort.java

```
package xmltest;

@javax.xml.bind.annotation.XmlRootElement (name
= "DataPort")
public class DataPort {

    private String name;
    private String direction;

    @javax.xml.bind.annotation.XmlAttribute
    public String getName()
    {
        return name;
    }

    public void setName(String name)
    {
        this.name = name;
    }

    @javax.xml.bind.annotation.XmlAttribute (name
= "Direction")
    public String getDirection()
    {
        return direction;
    }

    public void setDirection(String direction)
    {
        this.direction = direction;
    }
}
```

Converter.java

```
package xmltest;

public class Converter {

    public static void main(String args[]) throws
        javax.xml.bind.JAXBException, java.io.
        IOException
    {
        javax.xml.bind.JAXBContext readContext = javax.
            xml.bind.JAXBContext.newInstance(Device.
                class);

        /*Device device = new Device();
        device.setType("wheel_rotation_sensor");
        java.util.ArrayList<DataPort> features = new
            java.util.ArrayList<DataPort>();
        DataPort port = new DataPort();
        port.setName("wheel_pulse");
        port.setDirection("out");
        features.add(port);
        device.setFeatures(features);

        javax.xml.bind.Marshaller m = readContext
            .createMarshaller();
        m.setProperty(javax.xml.bind.Marshaller.
            JAXB_FORMATTED_OUTPUT, Boolean.TRUE);
        m.marshal(device, System.out);

        java.io.Writer writer = null;
        try
        {
            writer = new java.io.FileWriter("aadl.xml");
            m.marshal(device, writer);
        }
        finally
        {
            try
            {
                writer.close();
            }
            catch (Exception e)
            {
            }
        }
    }
}*/
```

```

javax.xml.bind.Unmarshaller um = readContext.
    createUnmarshaller();
Device newDevice = (Device)um.unmarshal(new
java.io.FileReader("aadl.xml"));
Transition transition = new Transition();
transition.setId(newDevice.getTypeName());
transition.setClassName("data_flow");
Place place = new Place();
place.setId(newDevice.getFeatures().get(0).getName());
place.setClassName("data_flow");
transition.setPlace(place);

javax.xml.bind.JAXBContext writeContext =
javax.xml.bind.JAXBContext.newInstance
(Transition.class);
javax.xml.bind.Marshaller m = writeContext.
createMarshaller();
m.setProperty(javax.xml.bind.Marshaller.
JAXB_FORMATTED_OUTPUT, Boolean.TRUE);
m.marshal(transition, System.out);

java.io.Writer writer = null;
try
{
    writer = new java.io.FileWriter("pnml.xml");
    m.marshal(transition, writer);
}
finally
{
    try
    {
        writer.close();
    }
    catch (Exception e)
    {
    }
}

}
}

```

APPENDIX C

PETRI NET TYPE DEFINITIONS

Petri Net Type Definitions

Petri Net Type Definitions specify the legal labels for particular Petri Net Types. PNML provides a mechanism for defining Petri Net Types and for using labels from a conventions document. PNTD contains additional features which are not included in PNML. It is the extension of PNML by object- oriented principles. The PNML Core Model contains the basic structural definition of a Petri Net as a labeled directed graph. It is the primary building block upon which concrete Petri Net types are defined. The basic structure of PNML Document is defined in the PNML Core Model. It contains one or more Petri Nets. There may be graphical information with each object. The information may include its position, shape, color etc.

The important objects of Petri Nets are places, transitions and arcs. According to the PNML Core Model, it is legal to connect two places by arcs and two transitions by arcs. In order to support the variations and extensions of the Petri Net types, Model engineering techniques are chosen.

REFERENCES

- [1] ISO/IEC 15909-2:2011 Systems and software engineering - High-level Petri nets - Part 2: Transfer format.
- [2] L. Hillah, F. Kordon, L. Petrucci, N. Trèves, “Model engineering on Petri nets for ISO/IEC 15909-2: API framework for Petri net types metamodels.” Petri Net Newsletter, 69:22–40, 2005.
- [3] L.M. Hillah, F. Kordon, L. Petrucci, N. Trèves, “PNML framework: An extendable reference implementation of the Petri Net Markup Language.” In: Lilius, J., Penczek, W. (eds.) PETRI NETS 2010. LNCS, vol. 6128, pp. 318–327. Springer, Heidelberg (2010)
- [4] L. Hillah, E. Kindler, F. Kordon, L. Petrucci, N. Treves, “A primer on the Petri Net Markup Language and ISO/IEC 15909-2.” In: Jensen, K. (ed.) 10th Workshop on Coloured Petri Nets (CPN 2009), pp. 101–120 (2009)
- [5] L. Hillah, F. Kordon, L. Petrucci, N. Trèves, “PN standardisation: a survey.” In 26th International Conference on Formal Methods for Networked and Distributed Systems (FORTE'06), volume 4229 of LNCS, pages 307-322, Paris, France, September 2006. Springer Verlag.
- [6] J.M.E.M. van der Werf, R.D.J. Post, (2004). *EPNML 1.1 : an XML format for Petri nets*. (External Report). Eindhoven: Petriweb.org, 16 pp
- [7] M. Weber, E. Kindler (2002) “The Petri net markup language.” In: Ehrig H, Reisig W, Rozenberg G, Weber H (eds) Petri net technology for communication based systems, vol 2472 of Lecture Notes in Computer Science, pp 124–144
- [8] H. Reza, E.S. Grant, "Toward Extending AADL-OSATE Toolset with Color Petri Nets (CPNs)," *Information Technology: New Generations, 2009. ITNG '09. Sixth International Conference on* , vol., no., pp.1085,1088, 27-29 April 2009
doi: 10.1109/ITNG.2009.246
- [9] P. H. Feiler, D. P. Gluch, J. J. Hudak. The Architecture Analysis & Design Language (AADL): An Introduction. Technical report, 2006. CMU/SEI-2006-TN-011.
- [10] Welcome on PNML.org. <http://www.pnml.org/tutorialpn09.php> accessed on 11th July.

- [11] J. Hudak, P. Feiler, "Developing AADL Models for Control Systems: A Practitioner's Guide," Software Engineering Institute, Carnegie Mellon University, Pittsburgh, Pennsylvania, Technical Report CMU/SEI-2007-TR-014, 2007.
<http://resources.sei.cmu.edu/library/asset-view.cfm?AssetID=8437>
- [12] T. Murata, "Petri nets: Properties, analysis and applications," *Proceedings of the IEEE* , vol.77, no.4, pp.541,580, Apr 1989
- [13] X. Renault, F. Kordon, J. Hugues. From AADL architectural models to Petri Nets: Checking model viability. In 12th IEEE International Symposium on Object-oriented Real-time distributed Computing (ISORC'09), pages 313-320, Tokyo, Japan, March 2009. IEEE CS.
- [14] X. Renault, F. Kordon, J. Hugues, "Adapting models to model checkers, a case study: Analysing AADL using Time or Colored Petri Nets," Rapid System Prototyping, IEEE International Workshop on, vol. 0, pp. 26-33, 2009
- [15] M. Y. Chkouri, A. Robert, M. Bozga, and J. Sifakis. "Translating AADL into BIP - Application to the Verification of Real-time Systems." In Model Based Architecting and Construction of Embedded Systems, 2008.
- [16] D. Gasevic, V. Devedzic, "Petri Net Markup Languages and formats as guidelines for ontology development", *In Proceedings of the IADIS International Conference on E-Society*, pp 662-665, June 2003.
- [17] Z. Jin, A software architecture-based testing technique. Thesis for Doctor of Philosophy in Information Technology, George Mason University, Fairfax, Virginia (2000)
- [18] JAXB Tutorial, <http://www.vogella.de/articles/JAXB/article.html>
- [19] Java and XML Tutorial, <http://www.vogella.de/articles/JavaXML/article.html>
- [20] Generating XML from an Arbitrary Data Structure,
<http://download.oracle.com/javaee/1.4/tutorial/doc/JAXPXSLT5.html>
- [21] B. Jean-Paul, C. Raphaël, C. David, F. Mamoun, R. Jean-François, "A mapping from AADL to Java-RTSJ," International Workshop on Java Technologies for Real-time and Embedded Systems, pp. 165-174, 28/09/07 .
- [22] C. Dong, J. Bailey, "Static analysis of XSLT programs," Proceedings of the 15th Australasian database conference , vol. 27, pp. 151-160, 2004.
- [23] K. Tongprasert, S. Chittayasothorn, "An XML-based Petri Net to Rules Transformation Software Tool," The 14th World Multi-Conference on Systemics, Cybernetics and Informatics: WMSCI 2010 , pp. 1-4, 2010.

- [24] K. Jensen, L. M. Kristensen, L. Wells, "Coloured Petri Nets and CPN Tools for modelling and validation of concurrent systems," *International Journal Software Tools Technology Transfer*, vol. 9, pp. 213-254, 13 March 2007.
- [25] M. Hecht, C Vogle, A Lam, "Application of the Architectural Analysis and Design Language (AADL) for Quantitative System Reliability and Availability Modeling," *Aerotech 2009*, November, 2009.
- [26] F. Singhoff, J. Legrand, L. Nana, L. Marce, "Scheduling and memory requirements analysis with AADL," *Proceedings of the 2005 annual ACM SIGAda International Conference on Ada: The Engineering of Correct and Reliable Software for Real-Time and Distributed Systems using Ada and Related Technologies*, vol. XXV, no. 4, pp. 1 - 10, December 2005.
- [27] K.B. Lassen, M. Westergaard, "Embedding Java Types in CPN Tools," *Transactions on Petri Nets and Other Models of Concurrency*, pp. 1 - 19, October 2006.
- [28] F. Bonnefoi, C. Choppy, F. Kordon, "A Discretization Method from Coloured to Symmetric Nets: Application to an Industrial Example," *Transactions on Petri Nets and Other Models of Concurrency III Lecture Notes in Computer Science*, vol. 5800, pp. 159 - 188, 2009.
- [29] T. A. Henzinger, B. Horowitz, and C. M. Kirsch, "Giotto: A time-triggered language for embedded programming", *Proc. 1st Int. Workshop on Embedded Software*, vol. 2211, pp.166 -184 2001
- [30] J. Clark. RELAX NG Home Page. <http://www.relaxng.org/2003>
- [31] B. Berthomieu, "Formal Verification of AADL Specifications in the Topcased Environment." *Ada-Europe '09 Proceedings of the 14th Ada-Europe International Conference on Reliable Software Technologies*. (2009): 207 – 221
- [32] A.-E. Rugina, "System dependability evaluation using AADL (Architecture analysis and design language)", *Rencontres Jeunes Chercheurs en Informatique Temps-Réel (RJCITR)*, 2005
- [33] D. Hemer, Y. Ding, "Modelling Software Architectures Using CRADLE," in *World IMACS/MODSIM Congress*, 2009, pp. 404-410.
- [34] Z. Qureshi, "Formal Modelling and Analysis of Mission-Critical Software in Military Avionics Systems." *Proceedings of the eleventh Australian workshop on Safety critical systems and software*. (2006): 67-77.
- [35] L. Grunske, J. Han, "A Comparative Study into Architecture-Based Safety Evaluation Methodologies using AADL's Error Annex and Failure Propagation Models." *11th IEEE High Assurance Systems Engineering Symposium*. (2008): 283 - 292.

- [36] X. Renault, F. Kordon, J. Hugues. "From AADL architectural models to Petri Nets: Checking model viability." In 12th IEEE International Symposium on Object-oriented Real-time distributed Computing (ISORC'09), pages 313-320, Tokyo, Japan, March 2009. IEEE CS.
- [37] A.E. Rugina, K. Kanoun, M. Kaâniche. "A system dependability modeling framework using AADL and GSPNs." *Architecting Dependable Systems IV*. Springer Berlin Heidelberg, 2007. 14-38.
- [38] Systems and software engineering -- High-level Petri nets -- Part 1: Concepts, definitions and graphical notation. Switzerland : INTERNATIONAL STANDARD ISO/IEC 15909-1, 2004.
- [39] M. Magyar, I. Majzik, "Tool Supported Structures of dependability Evaluation of Redundant architectures in Computer based control systems," in IEEE Int. Conf. Quantitative Evaluation of Systems, pp. 95-96, 2009
- [40] A. Ghosh, L. Pereira, T. Yan, H. Cao. "Modeling Wireless Sensor Network Architectures using AADL." In *ERTS*, Toulouse, France, January 2008.
- [41] K. Jensen, L. Kristensen, L. Wells. "Coloured Petri Nets and CPN Tools for Modelling and Validation of Concurrent Systems." *Journal International Journal on Software Tools for Technology Transfer*. no. 3 (2007): 213-254. <http://dl.acm.org/citation.cfm?id=1266789> (accessed April 13, 2014).
- [42] H. Reza, F. Gu, B. Shafai. "Toward Model Based Testing: Combining AADLs with High Level Petri Nets." *Software Engineering Research and Practice* (2010): 619-623.
- [43] H. Reza, S. Lande, "Model Based Testing Using Software Architecture," *Information Technology: New Generations (ITNG), 2010 Seventh International Conference on*, vol., no., pp.188,193, 12-14 April 2010 doi: 10.1109/ITNG.2010.122
- [44] H. Reza, R. Marsh, M. Askelson. "A Fault Tolerant Architecture Using AADLs and Error Model Annex for Unmanned Aircraft Systems (UAS).. " *Software Engineering Research and Practice* (2010): 180-184.
- [45] A.E. Rugina, K. Kanoun, M. Kaâniche, "The ADAPT Tool: From AADL Architectural Models to Stochastic Petri Nets through Model Transformation," Proceedings of the 2008 Seventh European Dependable Computing Conference, p.85-90, May 07-09, 2008
- [46] E. Kindler, L. Petrucci, "A framework for the definition of variants of high-level Petri nets." *Proceedings of the Tenth Workshop and Tutorial on Practical Use of Coloured Petri Nets and CPN Tools (CPN '09)*. 2009. 121-137

[47] M. Jünger, E. Kindler, M. Weber, 2000. Towards a Generic Interchange Format for Petri Nets. In: Bastide, R., Billington, J., Kindler, E., Kordon, F. and Mortensen, K. H. (eds.): Meeting on XML/SGML based Interchange Formats for Petri Nets. 1--5, Århus, Denmark, 21st ICATPN.

[48] N. Muhammad, Y. Vandewoude, Y. Berbers, S. van Loo. 2009. Modelling composite end-to-end flows with AADL. In *Proceedings of the workshop on the definition, evaluation, and exploitation of modelling and computing standards for Real-Time Embedded Systems* (Dublin, Ireland. July 1--3, 2009)

[49] <http://en.wikipedia.org/wiki/XSLT>

[50] <http://en.wikipedia.org/wiki/XPath>