Journal of Software Engineering and Applications, 2014, 7, 920-933
Published Online October 2014 in SciRes. http://www.scirp.org/journal/jsea
http://dx.doi.org/10.4236/jsea.2014.711082

Scientific Research

Mapping AADL to Petri Net Tool-Sets Using PNML Framework

Hassan Reza, Amrita Chatterjee

Department of Computer Science, School of Aerospace Sciences, University of North Dakota, Grand Forks, USA
Email: reza@aero.und.edu

Received 23 August 2014; revised 18 September 2014; accepted 15 October 2014

Abstract

Architecture Analysis and Design Language (AADL) has been utilized to specify and verify non-functional properties of Real-Time Embedded Systems (RTES) used in critical application systems. Examples of such critical application systems include medical devices, nuclear power plants, aerospace, financial, etc. Using AADL, an engineer is enable to analyze the quality of a system. For example, a developer can perform performance analysis such as end-to-end flow analysis to guarantee that system components have the required resources to meet the timing requirements relevant to their communications. The critical issue related to developing and deploying safety critical systems is how to validate the expected level of quality (e.g., safety, performance, security) and functionalities (capabilities) at design level. Currently, the core AADL is extensively applied to analyze and verify quality of RTES embed in the safety critical applications. The notation lacks the formal semantics needed to reason about the logical properties (e.g., deadlock, livelock, etc.) and capabilities of safety critical systems. The objective of this research is to augment AADL with existing formal semantics and supporting tools in a manner that these properties can be automatically verified. Toward this goal, we exploit Petri Net Markup Language (PNML), which is a standard acting as the intermediate language between different classes of Petri Nets. Using PNML, we interface AADL with different classes of Petri nets, which support different types of tools and reasoning. The justification for using PNML is that the framework provides a context in which interoperability and exchangeability among different models of a system specified by different types of Petri nets is possible. The contributions of our work include a set of mappings and mapping rules between AADL and PNML. To show the feasibility of our approach, a fragment of RT-Embedded system, namely, Cruise Control System has been used.

Keywords

Model-Based Engineering, Petri Nets, AADL, PNML, Software Architecture, Formal Methods

How to cite this paper: Reza, H. and Chatterjee, A. (2014) Mapping AADL to Petri Net Tool-Sets Using PNML Framework. *Journal of Software Engineering and Applications*, **7**, 920-933. http://dx.doi.org/10.4236/jsea.2014.711082

1. Introduction

The Society of Automotive Engineers (SAE) developed Aerospace Standard AS5506 [1], the Architecture Analysis & Design Language (AADL) and its supporting toolset Open Source Architectural Tool Environment (OSATE), for the model-based engineering (MBE) of Real-Time Embedded Systems (RTES). AADL initially designed to assist avionic engineers to model hardware platform, software, and external environment such as physical devices connected to the systems (e.g., sensors).

AADL applicability in other industrial settings (e.g., military and aerospace) is increasingly growing in the recent years for many reasons. AADL, which is based on a component-connector architectural paradigm, supports many features such as multi-dimensional analyses, code generations, model-transformations, etc. These features are essential in specification and verification of the front-end and back-end engineering of RTES used in safety critical systems [2] [3].

Safety critical systems refers to the class of systems whose malfunctions may results in loss of human lives, significant loss of financial assets, or environmental damage [4]. Examples of safety critical systems include medical devices (peacemaker), avionics (unmanned aircraft system), software that controls nuclear power plants, etc. Using AADL and supporting tools developers can analyze a model of a safety critical system for different types of qualities. Examples of these qualities include performance, safety, security, etc.

Although AADL has proven to be an effective modeling notation for RTES deployed in safety critical systems, but the notation suffers from a number of limitations. For one thing, AADL and its supporting tool support Open Source AADL Tool Environment (OSATE) [5]-[7] are unable to verify the logical correctness of these systems. Examples of these properties include livelock and deadlock.

One the other hand, the theory of Petri Nets have been extensively applied to specify and verify the logical behaviors of complex, concurrent, and distributed systems [8]. The classical Petri nets, commonly known as flat Petri nets, provide the required primitives constructs and inscriptions (e.g., place, transitions, flows) needed for modeling and analyzing concurrency, synchronization, determinism, and distributeness used in inherently concurrent and distributed systems. Using model checker or simulations, Petri nets can be analyzed or simulated for logical correctness (e.g., livelock and deadlock).

The High Level Petri nets [9] extends the formal underpinning of the flat Petri nets with additional graphical constructs and annotations to support hierarchy, compositions, etc., these features, in turn, assist to manage the complexity and hence maintainability of large scale Petri net specifications. Therefore, it makes sense to extend AADL with the well-established theory of Petri nets to automatically analyze the behavioral correctness of the safety critical systems specified by AADL. Toward this goal, the AADL is translated to the Petri Net Markup Language (PNML) through an interchange format, XML. PNML is a standard defined by ISO/IEC 15909-2 in order to support interoperability various kinds of Petri nets tools [10]-[14]. According to ISO/IEC 15909-2, PNML is defined as the transfer format for the three types of Petri nets, Predicate Transition Nets (P/T nets), Symmetric Nets, and High Level Petri Nest (HLPN), but the framework is flexible and extensible enough to support any type of Petri net in future. PNML provides an interface that works as intermediate representation allowing automated translation among those currently supported by the PNML framework and other types of nets such as Colored Petri Nets, Stochastic Petri Nets and Timed Petri Nets [15].

Our approach to map AADL via PNML to Petri nets Tools is very efficient because it using intermediate representation [16]. More specifically, our work heavily influenced by the program compilation techniques in which AADLs represents high level architectural specification notations translated to low-level design specifications (e.g., Petri nets) through PNML framework. In our work, the PNML framework acts as intermediate representation [16]. The PNML framework is well-deigned and general enough to capture the capabilities common by all types of Petri nets. This feature, in turn, will make it possible not only to map AADL to Petri nets, but also other high level architectural notations such ADLs, SysmL, etc. to be mapped to different types of Petri nets.

In order to interface ADDL with other Petri net software solutions via PNML, XML has been used. XSLT (Extensible Style Sheet Language Transformation) [17] supports the transformation and representation of data in the XML format and has become indispensable for XML data exchanges. XSLT script is the specification of the output as a function of input and is used in this work to automate the translation of AADL-XML to PNML.

2. Related Work

AADL describes the architecture of the software and hardware components of a system and its functional inter-

faces [5]. AADL is used to analyze the non-functional properties of a system. The notation lacks formal semantics needed to perform safety and logical reasoning of ultra-complex systems. The correctness of such systems is of crucial importance in safety, security, and mission critical systems. A number of studies have explored the extension of AADL with formal semantics to allow the verification of logical properties of the RT-Embedded Systems used in avionics and aerospace systems. These approaches attempt to extend AADL include work performed by [3], [18] to analyze logical properties of complex systems. In [2], the authors extend AADL with Petri nets in order to perform safety analysis (*i.e.*, deadlock, livelock, etc.) using model-checking. In [18], AADL is used to describe functional interfaces and performance-critical aspects of components. To this end, AADL is transformed into BIP (Behavior Interaction Priority). Other study [19] proposed extending AADL-OSATE with Petri nets to facilitate a formal analysis using simulation.

Some aspects of AADLs have also been validated using both Colored and Timed Petri Nets [3]. In [3], the focus was on the behavioral analysis of AADL by means of Petri Net Models; this study provides a useful illustration of how to take advantage of Petri nets and utilize their capabilities for analyzing the properties such as deadlock detection and dimensioning. These researchers were seeking to ensure their AADL models were deadlock-free, livelock-free and bounded.

A study by [20] indicated that AADL cannot be easily mapped to some real time platforms. The researchers selected RTSJ for their study of the correctness of the mapping, concluding that the Java kernel implements the semantics of a subset of AADL. For example periodic, aperiodic and sporadic dispatch protocols are supported, in addition to thread dispatch and complete states. They concluded that their results are significant for programming with real-world systems and that AADL-type communication can be used for data, events and event data ports for threads when other components have not yet been implemented.

In [21], software tools were employed to transform Petri Net Markup Language to Rule ML. More specifically, the algorithm represented in [21] utilized a depth-first search; the results obtained were compared with those reported by earlier researchers. Inference engines in Java are mentioned as having the capability to infer rules in XML format. Rule language was assigned a high priority in this study owing to the knowledge representation for knowledge based systems and Petri nets were employed for the knowledge representation. Rule-based systems have gained in importance due to their relative simplicity. However, the removal of old rules and addition of new rules may lead to consistency issues; to avoid these issues, a modeling technique can be employed to check for consistency. In [19], AADL is extended by High Level Petri Nets for model based testing [22].

3. Background

3.1. Petri Nets and PNML

Petri nets provide a powerful modeling language [23] that is widely used to specify and verify the behaviors of complex, concurrent, and distributed software systems. For the practical application of Petri nets, interoperability between the various Petri net models and supporting tools is essential.

A flat Petri net (or classical net) can be considered to be a labeled directed graph where the labels represent all the specific information of the net. It may be associated with a node (places, transitions), arc or net. The mathematical representation of Petri nets allows analysis of deadlock and livelock. The main structural components of Petri nets are places, arcs and transitions and tokens. Places represent passive elements such as data/conditions; they are further classified into Input places and Output places. They are graphically represented by circles. Transitions represent active elements of a system such as events, tasks, actions; they are further classified into Input transition and Output transition. They are graphically represented by solid bars or rectangular. Arc represent relationships or flow of information between places and transitions [8].

Classical Petri nets suffer from combinatorial and complexity problems and hence are difficult to be used in modelling of complex systems with significant number of states. High level Petri Nets such as Colored Petri Nets (CPN) were developed to manage the complexity of nets by providing abstractions (super nodes), hierarchical, functional programming language constructs, and compositional features [24] [25]. CPN has been applied for state space and performance analysis, automatic simulation, visualization, etc.

The Petri Net Markup Language (PNML) is Extensible Markup Language-based meta-language allowing interoperability and exchangeability among different class of Petri Nets [9]-[13] [26] [27]. Currently, the PNML notation permit the exchange of Petri nets among toolset supported by three class of Petri Nets, namely, Place/Transition-Nets, High-level Petri Nets, and Symmetric Nets. To allow the exchangeability among tool supported

by the above nets, the PNML support universality, extensibility, readability, mutuality, etc.

Currently, PNML concepts fall into two major categories: core, and tool specific concepts. The PNML core concepts capture he common graphical and structural elements of the Petri nest. These concepts constitute the PNML core meta-model, which is defined in UML class diagrams [28]. Examples of these elements include place, transition, arc, and labels, which are used by all classes of Petri nets models and tools.

The PNML tool specific concepts maintain tool-specific information associated with specific objects used by specific class of Petri Nets [9]-[13]. Those information are meant for specific type of tool and nets, which has nothing in common with other information and features supported by other tools.

3.2. Software Architecture and Architectural Design Analysis TOOLS (AADLS)

Software architecture (SA) of software intensive systems plays pivotal role in success or failure of a system. SA has many definitions. The classical definition of SA refers to a set of components, connectors, constraints, and configurations from which the system under construction is modeled [29]. As a formal model of SA, architectural specifications can provide a concrete formal foundation for reasoning about system-wide properties. More precisely, the formal architectural model of a system allows behavioral and structural errors and inconsistencies be detected early on. For example, in [22] [30] [31], formal specifications of SA have been used to test a system.

Architecture Description Languages (ADLs) are formal modeling notations that provide precise syntax and precise semantics together with reasoning capability for characterizing software architecture using architectural constructs [32]. ADLs typically provide supporting tool-sets for editing/drawing, analyzing (type checking), or simulating architectural specifications.

The SAE-AADL is an architecture description language used in model-based engineering (MBE); it is aimed at modeling and analysis of critical systems deployed mainly in avionic and aerospace industries [33]. These systems are normally distributed, embedded real-time requiring anultra-degree of dependability (availability, security, safety, and performance) [33]. AADL core has been defined to support both textual and graphical views. The notations provides excessive set of modelling constructs needed to model hardware and execution platform, software components, constraints, and properties. A core AADL model can be annotated with additional information to quantitatively predict and analyze different operational qualities such as performance, security, safety, etc. For example, the Error Model Annex is an extension to AADLs standard to specify the failure modes of a system using formal automata. This annex, in turn, has been used as a risk mitigation method in Real-Time embedded system to increase the dependability of a system [34]-[36].

The AADL-SAE core is very attractive for modelling of safety critical systems for the following reasons:

- Provide an XML interchange format that allows model transformations, integrations, and/or navigations between different models (e.g., textual to graphical representation or vice versa);
- Provide a UML profile representing AADL as a specialized modeling notation within UML/MDD (Model-Driven Development);
- Allow integrate-ability with existing commercial and open source tool solutions and plug-ins (e.g., Open Source AADL Tool Environment (OSATE) developed by the SEI (Software Engineering Institute) and other institutes and agencies. Also, the AADL Meta model and XMI/XML model interchange format standard annex allows integration of AADL models with other models.

The underlining formal theory of AADL is based on MetaH, which is a domain specific ADL originally designed by US Army. The language had been used for specification and verification RTES deployed in avionic and aerospace industries, and to manage development cost and schedule over-runs [18]. Meta Hserves dual proposes of both being an ADL and a CASE tool by providing collections of constructs; these constructs can be used to specify and analyze hardware and soft ware components of a Real-Time Embedded Systems. Using MetaH and its supporting toolsets, designers are able to perform various analyses such as syntactic consistency checking, performance (schedulability), reliability, availability and safety aspects of a system. However, the important shortcoming of MetaH is that the notation does not support the analyzability needed to reason about the logical properties (e.g., deadlock and/or livelock) of a system.

OSATE tool support used by an AADL is an extensible open source platform that has been developed by the Software Engineering Institute (SEI) to provide efficient and effective solution to complex systems. OSATE supports, among other things, support simple integration of existing analyses using java-plug-ins deployed on Eclipse platform. For example, in [36] using OSATE, an engineer is able to specify operational quality such as

safety [35].

4. Method

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 [9]. 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 [17] script is employed.

There are several important issues that must be considered when performing the mapping to a PNML. 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 this conversion. To tackle this issue, 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. The PNML format, in turn, facilitates the interfacing of different types of Petri nets supported by the PNML standard.

The methodology consists of two distinct mappings as follows: 1) mapping of AADL text to AADL-XML; this conversion is supported by OSATE, which is an eclipsed-based tool supporting various types of quality-based analyses [5]; 2) mapping of AADL-XML to PNML via XSLT. Step one consists of the conversion of AADL text to AADL-XML shown in **Figure 1** to transform the textual version of AADL into the XML format. The subset of mapping rules to convert AADL-text to AADL-XML is shown in **Table 1**.

The second step is to map AADL-XML to PNML using XSLT [17] templates. XSLT processor facilitates the conversion of languages in their XML format. To this end, XSLT script attempts to automate the conversion of AADL-XML to PNML. The XSLT are the templates works with a set of criteria and XPaths [17] to perform the desired pattern matching. Using this mechanisms and XSL, XSLT is then enabled to convert one XML document to another XML document using XML-based presentations of place, transition, arc, label in XML tags are <place>, <transition>, <arc>, and <label> respectively defined in [21] [26].

Figure 2 depicts the mapping between ADDL-XML and PNML using XSLT. **Table 2** shows the corresponding translations of AADLs elements to PNML using two mapping steps.

Once the AADL converted to PNML, the designer should be able to utilize the analyzability and capabilities of offered by different types of Petri nets supported by PNML standard. **Figure 3** shows a simple representation of PNML using XML tags. In that figure, every individual structural elements of Petri Nets such as place, transition, and arc, identified by unique identifier.

Figure 4 shows the complete process by which AADL is mapped to PNML and the possible interoperability

Figure 1. The transformation of AADL to AADL-XML.

Table 1. Subset of mapping rules to convert AADL text to XML-format.

AADL-text	AADL-XML
devicedevice_name	<deviceType name = “device_name>…</devicType>
processprocess_name	<processType name = “process_name>…</orocessType>
features	<features>…</features>
dataPortdataPort_name,	<dataPort name = “dataPort_name direction = “out” | “in””/>,
eventDataeventData_name	<eventPort name = “eventPort_name” direction “out” | “in”/>,

Figure 2. The conversion of AADL-XML to PNML via XSLT.

Table 2. AADL to PNML mapping.

AADL	PNML
In or Out Data Port	Place
In or Out Event Port	Place
Port Group	Place
Event Data Port	Place
Data Access	Place
System	Super transition or regular transition
Process	Super transition or regular transition
Thread	Super transition or regular transition
Device	Super transition or regular transition
Memory	Super transition or regular transition
Connection, Bus	Arc (or flow)

```
<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>
</annotation>
```

Figure 3. Example of PNML representation of graphical elements of Petri nets.

among different types of Petri nets via PNML. Some of Petri nets shown in **Figure 4** (e.g., Colored Petri nets, Timed Petri net, etc.) are not yet supported by the current PNML standard, but the PNML framework is designed extensible and flexible enough to incorporate these types of high-level Petri nets.

5. Case Study: Cruise Control System (CCS)

In this section, we discuss a case study to demonstrate how one component of a system specified in AADL can be transformed to PNML. The transformation is shown using the method discussed in previous section. 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 PNML. XSLT script is utilized for mapping AADL to PNML in the second. For the purpose of this case study, an automobile Cruise Control System (CCS), which discussed in [11], is considered

as our running example to demonstrate how one component of a cruise control system (CCS) is translated to PNML.

In the simple form, the cruise control system (CCS) monitors the speed of an automobile under various conditions. CCS is an RTES used by drivers in the case of steady traffic conditions such as those typically encountered when driving long distances on an interstate under different type of weather condition. CCS takes over the throttle of the car and helps to maintain a set speed for the vehicle that is chosen by the driver. The main components of a cruise control system are shown in **Figure 5**.

The main components of a cruise control system are: Cruise Control Switch; Brake; Engine; Wheel Rotation

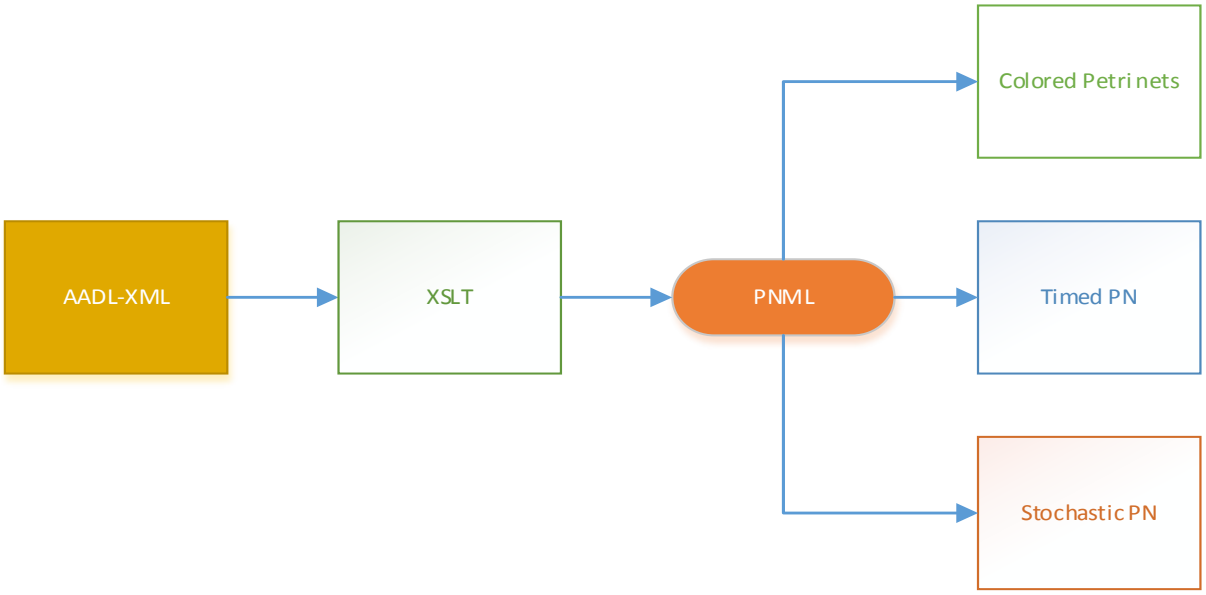

Figure 4. Possible extension of PNML with other types Petri Nets.

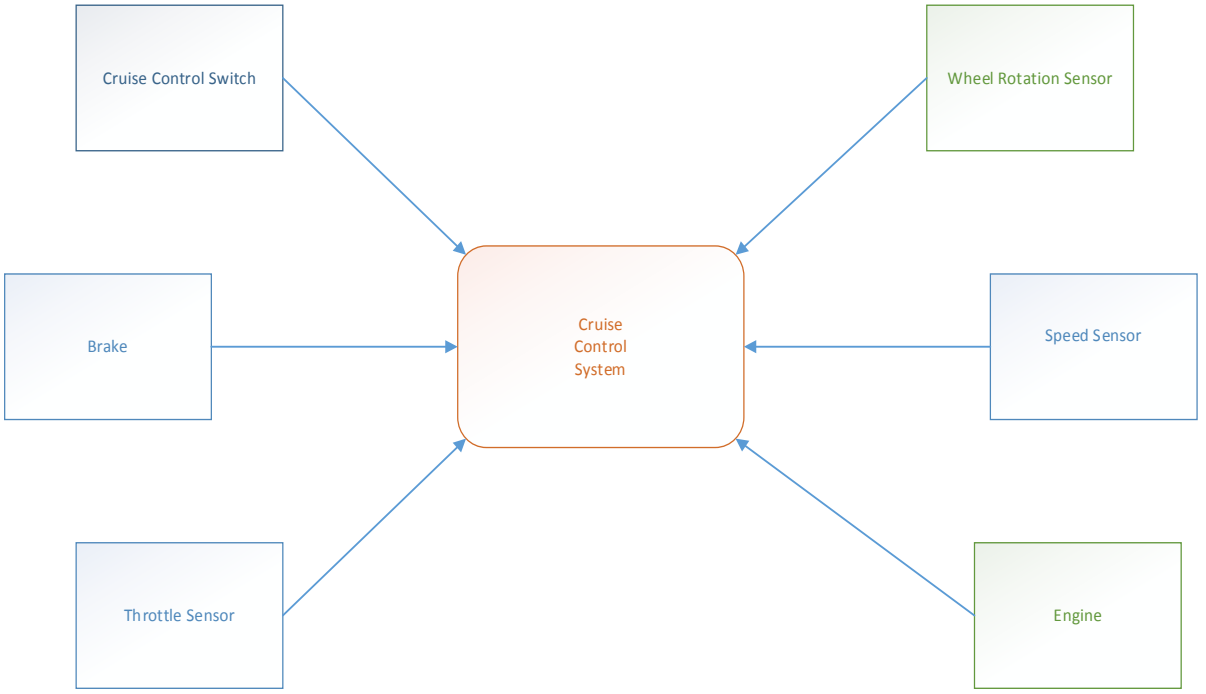

Figure 5. The context diagram representing external environment of a Cruise Control System (CCS).

Sensor (Device); Throttle Sensor; and Speed Sensor. Cruise Control Switch 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 is the braking system deactivates the cruise control system. The "wheel rotation sensor" monitors the wheel rotation.

To make the case more manageable, only one component of the cruise control system, namely, the "wheel_ rotation_sensor" deviceis used. In what follows, we how "wheel_rotation_sensor" device is translated to the XML version of PNML using the mapping rules discussed in the previous chapter. 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 this device consists of data and control commands.

An AADL device is generally used to represent various physical devices connected to the systems. Examples of these components include, camera, Global Positing System (GPS), sensors, etc. These components have an interfaces (ports) with the external environments. In general, device can represent single function components and/or more complicated components. Single function components include sensors such as the wheel_rotation_ sensor, while complicated components include GPS.

AADL syntax represents the components, connections and behavior of a system. More specifically, AADL 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. **Figure 6** shows the AADL specification of wheel_rotation_sensor. **Figure 7** shows the transformation of the textual version of AADL into the XML using **Table 3**. **Figure 8** demonstrates the corresponding PNML representation of "wheel_rotation_senor". In that figure, the data type is undefined. This is indicated by untyped data declarations. After applying mapping rules in **Table 4**, the corresponding PNML version is obtained. As

```
device wheel_rotation_sensor
features
    wheel_pulse: out data port;
end wheel_rotation_sensor;
```

Figure 6. AADL text of device "wheel_rotation_ sensor".

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

Figure 7. XML version of AADL text for "device".

Table 3. 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>
Device	DeviceType name
data port	DataPort name
in	Direction="in"
out	Direction="out"

Table 4. 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”

```
<?xml version="1.0" encoding="utf-8"?>
<transition id=“t1”>
    <graphics>
        <position x=”q” y=“w”/>
<graphics>
<name>
    <value> wheel_rotation_sensor </value>
<graphics>
        <offset x “-s” y=“-s’”/>
</graphics>
<arc id= “a1” source =“t1” target=“pl”>
<graphics>
<position x=“u” y=“o”/>
<position x=“u’” y=“o’”/>
<graphics>
<annotation>
    <value>out_data_port</value>
<graphics>
<offset x “-ss” y=“-ss’”/>
<graphics/>
</annotation>
</arc>
<place id="p1"/>
<graphics>
<position x “-sr” y=“-sr’”/>
</graphics>
<name>
<value>wheel_puls</value>
<graphics>
    <offset x=“-qr” y=“-qr’”/>
</graphics>
<name/>
<initialMarking>
<value>place</value>
<graphics>
    <offset x=“-r” y=“-r”/>
<graphics>
</initialMarking>
</place>
```

Figure 8. PNML Version of “wheel_rotation_sensor”.

can be seen from **Figure 8**, the XML version of AADL “Device” component of CCS is obtained by mapping it to the “transition” component in the PNML. In/out data ports are mapped to “place” in the Petri net according to the mapping rules.

The wheel rotation is modeled by the “device”, namely, wheel_rotation_sensor in AADL. The corresponding XML version of the “Device” component of AADL (*i.e.*, wheel_rotation_sensor) using mapping rules in **Table**

3 can be represented in Figure 7.

Figure 8 shows the PNML version of "wheel_rotation_sensor", which was obtained byapplying of the mapping rules documented in Table 4.

The generic Petri nets representation of "wheel_rotaion_sensor" defined in PNML is represented by Figure 9. In that figure, transition models the AADL device. Place, wheel_pulse, represents output interface (*i.e.*, output port) and arc represent flow of data leaving the sensor.

At the heart of the above translation from AADL to PNML is XSLT (Extensible Stylesheet Language Transformations) [5] that is used for transforming one XML document to another XML document. Figures 10-14 show how XSLT is used for this purpose. More specifically, Figure 10 shows the original XML version of a document used as an input to XSLT. Figure 11 shows the XML template used to convert XML input to output. Figure 8 shows the output of the conversion.

XSLT is used for the conversion of one language in XML format to another language using XML format. The XSLT script takes the XML version of the sensor component as an input and generates the corresponding XML version of PNML as an output.

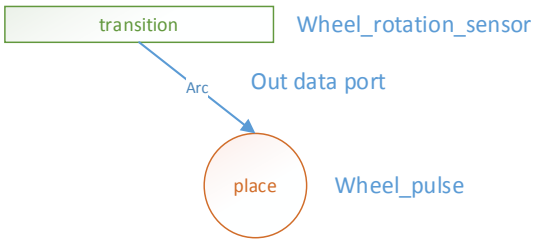

Figure 9. Petri nets representation of "device" using PNML.

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

Figure 10. XML input (AADL-XML).

```
<?xml version="1.0" encoding="iso-8859-1"?>
<!--Edited by XMLSpy®-->
<xsl:stylesheetversion="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 11. XSL script template.

```
<?xml version="1.0" encoding="iso-8859-1"?>
<!--Edited by XMLSpy®-->
<xsl:stylesheetversion="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 12. The XSLT script converting AADL-XML to PNML mapping.

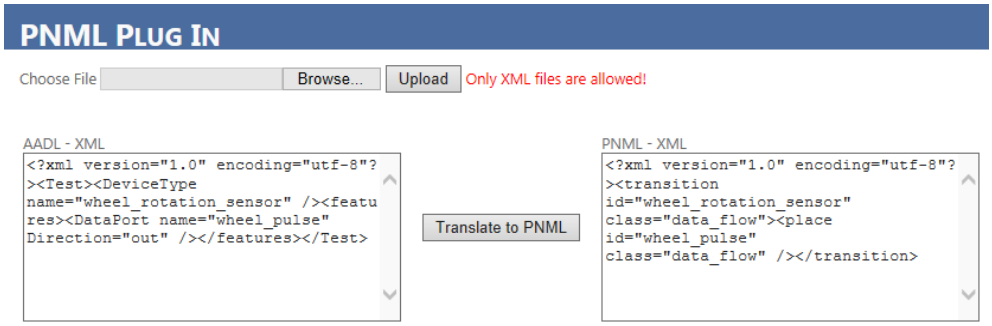

Figure 13. Screenshot of conversion of AADL-XML to PNML via XSLT.

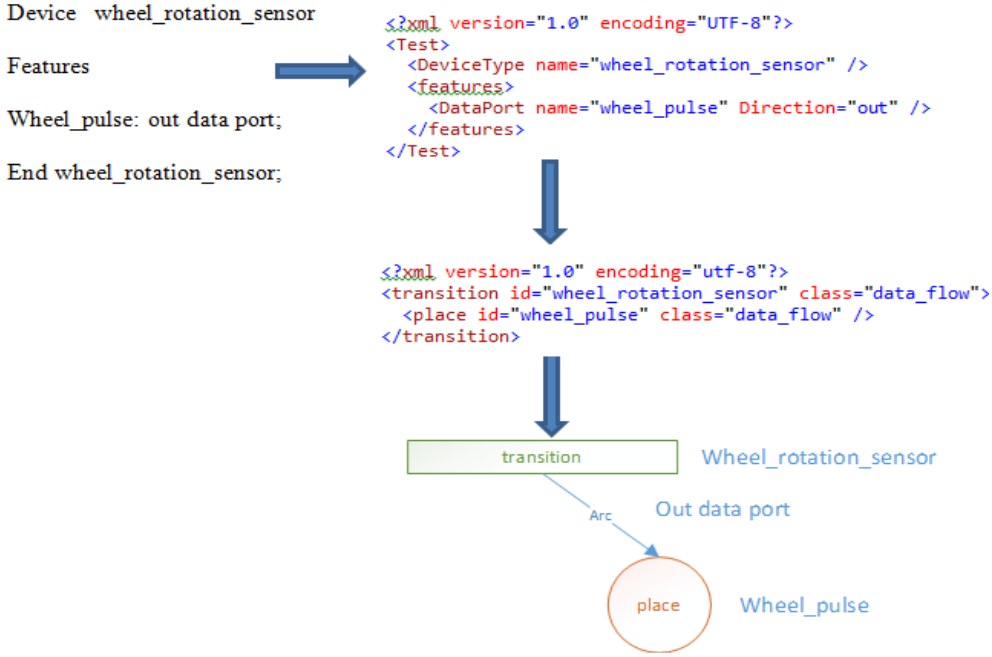

Figure 14. The complete transformation from AADL-Text to PNML.

Figure 12 depicts the XSLT Script used to map AADL-XML representation of wheel_sensorto PNML. In that figure, 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 **Table 4**).

Figure 13 represents the screen shot of a tool that automatically converts AADL-XML to PNML. In that figure, on left hand side, the panel shows AADL-XML, which is used as the input, and the right hand side shows the PNML version, which is generated as the output. This mapping is performed by the XSLT script shown in **Figure 11**.

The case study (CCS) discussed in this section 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 simple mapping rules. Once the AADL specifications are mapped to PNML, it then can 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 faithfully translated to PNML using the rules, then this would make it relatively straight forward to interface the AADLs to other types of Petri net via PNML. PNML makes it possible to apply various types of Petri net modeling and tool supports. For example, using Petri nets tool supports, a developer can simulate the net, or perform reach ability analysis, etc. **Figure 14** shows the complete process of converting AADL to PNML.

6. Conclusion and Future work

This study discussed how AADL representations can be augmented by capabilities supported by various types of Petri nets via PNML, which acts as intermediate representation (IR). There are some work that mapped AADL to specific type of Petri nets without using any IR. The main problem with these approaches is that the mapping process must be tailored for specific type of nets. PNML can be perceived as IR, because the framework poses important design attributes such as completeness (*i.e.*, complete set of elements common by all Petri nets modelling tools), extensibilities (*i.e.*, supported by Petri net Type definitions), and simplicity (*i.e.*, uses small set of constructs common to all types of Petri Nets) as suggested in [16].

Therefore, using PNML as intermediate representation (or language), we should be able to map AADL, UML, SysML to any type of nets (e.g., CPN, timed PN, etc.) that are currently supported or will be supported in future by the PNML framework. The PNML acting as an IR clearly separates front-end high level architectural languages from back-end specifications modelling languages with various tool support. This capability provided by PNML, in turn, provides a context in which front-end architectural descriptions languages can more efficiently mapped to back-ends modelling notations for various types of analyses. This is a significant improvement. For instance, for ADL architectural description languages and PN modelling specification supported by Petri nets, our approach required ADL+PN translations versus ADL×PN without PNML.

This study by no means completes. Therefore, it suggests several directions for future research. One important issue is the verification of XML code generated by XSLT. As discussed in [37], there are some anomalies (e.g., unreadable template, etc.) associated with using XSLT [37]. Therefore, we need to validate that the translation is faithful and correct.

The other important direction is the research on the design of a PNML plug-in that can be incorporated in the OSATE environment. Once the AADL text is given as input for the whole system, the corresponding PNML version should be generated automatically as output.

The other direction is the round trip engineering of safety critical systems using PNML, and AADL-OSATE. More specifically, how the developer can incorporate feedbacks from Petri nets models and tools to improve dependability of a system originally specified in AADL at architectural and how the AADL representation of the system at top level implemented and analyzed by low-level design using appropriate Petri Nets via PNML.

References

[1] ISO/IEC 15909-2 (2011) Systems and Software Engineering—High-Level Petri Nets—Part 2: Transfer Format.

[2] Hillah, L., Kordon, F., Petrucci, L. and Trèves, N. (2005) Model Engineering on Petri Nets for ISO/IEC 15909-2: API Framework for Petri Net Type Metamodels. *Petri Net Newsletter*, **69**, 22-40.

[3] Hillah, L.M., Kordon, F., Petrucci, L. and Trèves, N. (2010) PNML Framework: An Extendable Reference Implementation of the Petri Net Markup Language. In: Lilius, J. and Penczek, W., Eds., *Petri Nets*, *LNCS*, *Vol.* 6128, Springer, Heidelberg, 318-327.

[4] Hillah, L., Kindler, E., Kordon, F., Petrucci, L. and Treves, N. (2009) A Primer on the Petri Net Markup Language and ISO/IEC 15909-2. In: Jensen, K., Ed., *The* 10*th International Workshop on Practical Use of Colored Petri Nets and the CPN Tools* (*CPN* 2009), 101-120.

[5] Hillah, L., Kordon, F., Petrucci, L. and Trèves, N. (2006) PN Standardization: A Survey. 26*th International Conference on Formal Methods for Networked and Distributed Systems* (*FORTE*'06), Paris, 26-29 September 2006, 307-322.

[6] Van der Werf, J.M.E.M. and Post, R.D.J. (2004) EPNML 1.1: An XML Format for Petri Nets (External Report). Petriweb.org, Eindhoven, 16.

[7] Weber, M. and Kindler, E. (2002) The Petri Net Markup Language. In: Ehrig, H., Reisig, W., Rozenberg, G. and Weber, H. (Eds.), *Petri Net Technology for Communication Based Systems*, *Vol.* 2472 *of Lecture Notes in Computer Science* (*LNCS*), Springer-Verlag, Heidelberg, 124-144.

[8] Reza, H. and Grant, E. (2009) Toward Extending AADL-OSATE Toolset with Color Petri Nets (CPNs). *Proceeding of the Internal Conference on Information Technology*: *New Generations* (*ITNG*'09), Las Vegas, 27-29 April 2009, 1085-1088.

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

[10] XSL Transformations (XSLT) Version 1.1. http://www.w3.org/TR/xslt11/

[11] Hudak, J. and Feiler, P. (2007) Developing AADL Models for Control Systems: A Practitioner's Guide. Technical Report CMU/SEI-2007-TR-014, Software Engineering Institute, Carnegie Mellon University, Pittsburgh.

[12] Murata, T. (1989) Petri Nets: Properties, Analysis and Applications. *Proceedings of the IEEE*, **77**, 541-580. http://dx.doi.org/10.1109/5.24143

[13] Renault, X., Kordon, F. and Hugues, J. (2009) From AADL Architectural Models to Petri Nets: Checking Model Viability. 12*th IEEE International Symposium on Object-Oriented Real-Time Distributed Computing* (*ISORC*'09), Tokyo, 17-20 March 2009, 313-320.

[14] Renault, X., Kordon, F. and Hugues, J. (2009) Adapting Models to Model Checkers, a Case Study: Analyzing AADL Using Time or Colored Petri Nets. *IEEE International Workshop on Rapid System Prototyping*, Paris, 23-26 June 2009, 26-33.

[15] Chkouri, M.Y., Robert, A., Bozga, M. and Sifakis, J. (2008) Translating AADL into BIP—Application to the Verification of Real-Time Systems. In: Chaudron, M., Ed., *Model Based Architecting and Construction of Embedded Systems*, Springer-Verlag, Heidelberg, 5-19.

[16] Gasevic, D. and Devedzic, V. (2003) Petri Net Markup Languages and Formats as Guidelines for Ontology Development. *Proceedings of the IADIS International Conference on E-Society*, Lisbon, 3-6 June 2003, 662-665.

[17] Jin, Z. (2000) A Software Architecture-Based Testing Technique. Thesis for Doctor of Philosophy in Information Technology, George Mason University, Fairfax.

[18] Technical and Historical Overview of MetaH. http://aadl.sei.cmu.edu/aadl/documents/Technical%20and%20Historical%20Overview%20of%20MetaH.pdf

[19] Reza, H., Gu, F.F. and Shafai, B. (2010) Toward Model Based Testing: Combining AADLS with High Level Petri Nets. Software Engineering Research and Practice. CSREA Press, Las Vegas, 619-623.

[20] Reza, H. and Lande, S. (2010) Model Based Testing Using Software Architecture. *Seventh International Conference on Information Technology*: *New Generations* (*ITNG*), Las Vegas, 12-14 April 2010, 188-193.

[21] Jean-Paul, B., Raphaël, C., David, C., Mamoun, F. and Jean-François, R. (2007) A Mapping from AADL to Java-RTSJ. *International Workshop on Java Technologies for Real-Time and Embedded Systems*, Vienna, 26-28 September 2007, 165-174.

[22] Dong, C. and Bailey, J. (2004) Static Analysis of XSLT Programs. *Proceedings of the* 15*th Australasian Data Base Conference*, **27**, 151-160.

[23] Tongprasert, K. and Chittayasothorn, S. (2010) An XML-Based Petri Net to Rules Transformation Software Tool. *The* 14*th World Multi-Conference on Systemic, Cybernetics and Informatics*: *WMSCI* 2010, Orlando Florida, 29 June-02 July 2010, 1-4.

[24] Jensen, K., Kristiansen, L.M. and Wells, L. (2007) Colored Petri Nets and CPN Tools for Modelling and Validation of Concurrent Systems. *International Journal on Software Tools for Technology Transfer*, **9**, 213-254. http://dx.doi.org/10.1007/s10009-007-0038-x

[25] Hecht, M., Vogle, C. and Lam, A. (2009) Application of the Architectural Analysis and Design Language (AADL) for

Quantitative System Reliability and Availability Modeling. *Aerotech* 2009, Seattle, 30 August-2 September 2010, 1-15.

[26] Singhoff, F., Legrand, J., Nana, L. and Marce, L. (2005) 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*, **25**, 1-10.

[27] Lassen, K.B. and Westergaard, M. (2006) Embedding Java Types in CPN Tools. *Transactions on Petri Nets and Other Models of Concurrency*, Aarhus, 24-26 October 2006, 1-19.

[28] Bonnefoi, F., Choppy, C. and Kordon, F. (2009) A Discretization Method from Colored to Symmetric Nets: Application to an Industrial Example. *Transactions on Petri Nets and Other Models of Concurrency III Lecture Notes in Computer Science*, **5800**, 159-188.

[29] Reza, H., Gu, F. and Askelson, M. (2011) Model Based Engineering of Ground Based Risk Mitigation System. *Proceedings of the* 2011 *International Conference on Software Engineering Research & Practice*, Las Vegas, 18-21 July 2011, 260-265.

[30] Lewis, B. and Feiler, P. (2008) Multi-Dimensional Model-Based Engineering Using AADL. *The* 19*th IEEE/IFIP International Symposium on Rapid System Prototyping* (*RSP*'08), Monterey, 2-5 June 2008, 15-18.

[31] Knight, J.C. (2002) Safety Critical Systems: Challenges and Directions. *Proceedings of the International Conference on Software Engineering*, *ICSE* 2002, Orlando, 25 May 2002, 547-550.

[32] Shaw, M. and Garlan, D. (1996) Software Architecture: Perspectives on an Emerging Discipline. Prentice Hall, Upper Saddle River.

[33] Medvidovic, N. and Taylor, R. (2000) A Classification and Comparison Framework for Software Architecture Description Languages. *IEEE Transactions on Software Engineering*, **26**, 70-93. http://dx.doi.org/10.1109/32.825767

[34] Feiler, P. and Gluch, D. (2013) Model-Based Engineering with AADL: An Introduction to the SAE Architecture Analysis & Design Language. SEI Series, Addison Wesley, Boston.

[35] Reza, H., Marsh, R. and Askelson, M. (2010) A Fault Tolerant Architecture Using AADLs and Error Model Annex for Unmanned Aircraft Systems (UAS). Software Engineering Research and Practice, CSREA Press, Las Vegas, 180-184.

[36] Kindler, E. (2006) Concepts, Status, and Future Directions. In: Schnieder, E., Ed., *EKA* 2006, Braunschweig, 29-31 May 2006, 35-55.

[37] Chow, F. (2013) Intermediate Representation. *Communication of ACM*, **56**, 57-62. http://dx.doi.org/10.1145/2534706.2534720

Scientific Research Publishing (SCIRP) is one of the largest Open Access journal publishers. It is currently publishing more than 200 open access, online, peer-reviewed journals covering a wide range of academic disciplines. SCIRP serves the worldwide academic communities and contributes to the progress and application of science with its publication.

Other selected journals from SCIRP are listed as below. Submit your manuscript to us via either submit@scirp.org or Online Submission Portal.