

Carnegie Mellon University
Software Engineering Institute

AADLv2 library for SysMLv2

Jérôme Hugues

April 2023

TECHNICAL REPORT
CMU/SEI-2023-TN-001

SSD/ACPS

[DISTRIBUTION STATEMENT A] Approved for public release and unlimited distribution.

<http://www.sei.cmu.edu>



Copyright 2023 Carnegie Mellon University. This material is based upon work funded and supported by the Department of Defense under Contract No. FA8702-15-D-0002 with Carnegie Mellon University for the operation of the Software Engineering Institute, a federally funded research and development center. The view, opinions, and/or findings contained in this material are those of the author(s) and should not be construed as an official Government position, policy, or decision, unless designated by other documentation. This report was prepared for the SEI Administrative Agent AFLCMC/AZS 5 Eglin Street Hanscom AFB, MA 01731-2100 NO WARRANTY. THIS CARNEGIE MELLON UNIVERSITY AND SOFTWARE ENGINEERING INSTITUTE MATERIAL IS FURNISHED ON AN "AS-IS" BASIS. CARNEGIE MELLON UNIVERSITY MAKES NO WARRANTIES OF ANY KIND, EITHER EXPRESSED OR IMPLIED, AS TO ANY MATTER INCLUDING, BUT NOT LIMITED TO, WARRANTY OF FITNESS FOR PURPOSE OR MERCHANTABILITY, EXCLUSIVITY, OR RESULTS OBTAINED FROM USE OF THE MATERIAL. CARNEGIE MELLON UNIVERSITY DOES NOT MAKE ANY WARRANTY OF ANY KIND WITH RESPECT TO FREEDOM FROM PATENT, TRADEMARK, OR COPYRIGHT INFRINGEMENT. [DISTRIBUTION STATEMENT A] This material has been approved for public release and unlimited distribution. Please see Copyright notice for non-US Government use and distribution. Internal use:* Permission to reproduce this material and to prepare derivative works from this material for internal use is granted, provided the copyright and "No Warranty" statements are included with all reproductions and derivative works. External use:* This material may be reproduced in its entirety, without modification, and freely distributed in written or electronic form without requesting formal permission. Permission is required for any other external and/or commercial use. Requests for permission should be directed to the Software Engineering Institute at permission@sei.cmu.edu. * These restrictions do not apply to U.S. government entities. Carnegie Mellon® is registered in the U.S. Patent and Trademark Office by Carnegie Mellon University. DM23-0449

Table of Contents

Abstract	ii
A Appendix	1
A.1 Motivation for this work	1
A.2 Main package	2
A.2.1 AADL.sysml	2
A.3 Core Definitions	3
A.3.1 AADLPorts.sysml	3
A.3.2 AADLComponents.sysml	5
A.4 Component Categories	7
A.4.1 AADLAbstract.sysml	7
A.4.2 AADLBus.sysml	8
A.4.3 AADLData.sysml	9
A.4.4 AADLDevice.sysml	10
A.4.5 AADLMemory.sysml	12
A.4.6 AADLProcess.sysml	13
A.4.7 AADLProcessor.sysml	14
A.4.8 AADLSubprogram.sysml	15
A.4.9 AADLSubprogramGroup.sysml	16
A.4.10 AADLThread.sysml	17
A.4.11 AADLThreadGroup.sysml	19
A.4.12 AADLSystem.sysml	20
A.4.13 AADLVirtualBus.sysml	21
A.4.14 AADLVirtualProcessor.sysml	22
References/Bibliography	24

Abstract

The report is a proof-of-concept to defines elements of the Architecture Analysis and Design Language v2.3 (AADLv2) using the SysMLv2 textual syntax. SysMLv2 is the future revision of the SysML standard. It provides a lean syntax and rigorous static semantics to capture a system design. SysMLv2 also has the concept of library to specialize its building blocks for a specific domain.

SEI evaluated this capability to represent AADLv2 concepts using this alternate notation.

The appendix of this document provides the source code of this library, tested with the SysMLv2 pilot implementation. It shows that SysMLv2's power of expression is sufficient to capture AADLv2 concepts in a uniform way.

A Appendix

In this chapter, we publish the source code of SysMLv2 library that captures some aspects of the AADLv2.3 static semantics.

A.1 Motivation for this work

AADLv2.3 [1] is a mature standard, published by SAE International that provides a textual and graphical notation for the precise modeling and analysis of cyber-physical systems (CPS). Its semantics is well understood and implemented

SysMLv2 [2] is the future revision of the Systems Engineering Modeling Language. It is currently in draft status, entering finalization. There is a general interest in using SysMLv2 as a “lingua franca” to capture systems engineering artefacts from high-level concerns down to the precise architecture of CPS. In other words, to use SysMLv2 to also perform some of the modeling activities covered by AADLv2.3.

In this library, we demonstrate how to support AADL static constructs using SysMLv2. Instead of performing an exhaustive modelling of AADLv2.3 concepts, we cherry-picked elements that would exercise SysMLv2 syntax and semantics. Thus, our general approach is to capture the concepts of AADLv2.3 using the current draft of SysMLv2 at the time of writing, that is April 2023.

This library is organized in three parts:

- in section A.2, we introduce the main package to be imported by a user of this library;
- in section A.3, we introduce core definitions for central elements like AADL components, AADL ports, etc.
- in section A.4, we provide the definitions of each component category using the SysMLv2 syntax.

Summary: Through this work, we demonstrated the feasibility to use SysMLv2 to support AADL constructs. Future activities will consider completing the library to support all AADL constructs. This is an extension of the modeling patterns we have defined. Such activity will close the gap between systems engineering and the architecting of CPs.

A.2 Main package

AADL.SYSML is the main package to use this library. It bring all elements in the user namespace using SysMLv2 import mechanism.

A.2.1 AADL.sysml

```
/*
 * This is the root package of the AADLv2 library.
 * It exports all elements.
 */

package AADL {

    /* Generic concept definitions */

    import AADLPorts::* ;
    import AADLComponents::* ;

    /* Composite component categories */

    import AADLAbstract::* ;
    import AADLSystem::* ;

    /* Software component categories */

    import AADLProcess::* ;
    import AADLThread::* ;
    import AADLThreadGroup::* ;
    import AADLSubprogram::* ;
    import AADLSubprogramGroup::* ;
    import AADLData::* ;

    /* Hardware component categories */

    import AADLProcessor::* ;
    import AADLVirtualProcessor::* ;
    import AADLMemory::* ;
    import AADLDevice::* ;
    import AADLBus::* ;
    import AADLVirtualBus::* ;

}
```

A.3 Core Definitions

AADLPORTS.SYSML and AADLCOMPONENTS.SYSML provides the basic definitions used to capture AADL concepts as defined in [1], chapters 4 and 8.

- AADLPORTS.SYSML subsets SysMLv2 Port concept to build AADLv2's feature categories of ports and access.
- AADLCOMPONENTS.SYSML defines the concept of AADL component categories, Component_Category, and the basic component type, Component.

Every SysMLv2 representation of an AADL concrete component category subsets Component.

A.3.1 AADLPorts.sysml

```
/*
 * This package contains definitions for AADL ports
 */

package AADLPorts {

    /* SysMLv2 base library */
    import Ports::* ;
    import Connections::*;
    import CollectionFunctions::*;
    import BooleanFunctions::*;

    /* KerML base library */
    private import Base::Anything;

    doc /*
        * AADLv2 features categories
        */

    /*
        * SysMLv2 has the notion of port, called Port. This
        * conflict with the
        * term 'port' also used by AADL.
        *
        * First, we define an abstractFeature, as the root of
        * all features.
        * This type is then specialized for the various AADL
        * feature categories.
        */

    port def abstractFeature :> Port;

    doc /*
        * AADLv2 features categories
        */

    /*
        * First, we define the concept of port and derived
        * event/data/event
        */
}
```

```

    * data ports. These AADL ports have no subports.
    */

abstract port def AADLPort :> abstractFeature
{ assert constraint { isEmpty(subports) } }

abstract port def eventPort :> AADLPort;
abstract port def dataPort :> AADLPort;
abstract port def eventDataPort :> eventPort, dataPort;

/*
 * Then, we define the concept of AADL feature group,
 * i.e.
 * a group of ports.
 */

abstract port def featureGroup :> abstractFeature
{ assert constraint { notEmpty(ports) } }

abstract port def parameter :> abstractFeature;

abstract port def accessFeature :> abstractFeature;

abstract port def busAccess :> accessFeature;
abstract port def virtualBusAccess :> accessFeature;
abstract port def dataAccess :> accessFeature;
abstract port def subprogramAccess :> accessFeature;
abstract port def subprogramGroupAccess :>
    accessFeature;

doc /*
    * AADLv2 feature directions
    */

port def InPort :> abstractFeature
{assert constraint { true } }

/* An InPort has only one item, all in
*/

port def OutPort :> abstractFeature;
/* An OutPort has only one item, all out*/

connection def PortConnection :> BinaryConnection {
    end source : abstractFeature[0..*] :>>
        BinaryConnection::source;
    end target : abstractFeature[0..*] :>>
        BinaryConnection::target;

    assert constraint {
        /* See AADLv2.3 9.2 (L5) for the full list */
        ( source hastype eventPort & target hastype
            eventPort ) |

```

```

        ( source hastype dataPort & target hastype
          eventPort ) |
        ( source hastype dataPort & target hastype
          eventDataPort ) |
        ( source hastype dataPort & target hastype
          dataPort ) |
        ( source hastype eventDataPort & target hastype
          eventPort ) |
        ( source hastype eventDataPort & target hastype
          eventDataPort ) |
        ( source hastype eventDataPort & target hastype
          dataPort )
    }
}
}

```

A.3.2 AADLComponents.sysml

```

package AADLComponents {

    /* SysMLv2 base library */
    import SysML::*;
    import Parts::* ;
    import Ports::* ;
    import Items::* ;
    import Connections::*;

    /* KerML base library */
    private import Base::Anything;
    private import ScalarValues::*;
    private import ControlFunctions::*;
    private import CollectionFunctions::*;

    doc /* AADLv2 SysML library */
    import AADLPorts::*;

    doc /*
        * AADLv2 Component Categories
        */

    /* List of AADLv2 component categories */

    enum def Component_Category {
        enum Abstract;
        enum System;

        enum Process;
        enum Thread;
        enum ThreadGroup;
        enum Subprogram;
        enum SubprogramGroup;
        enum Data;
    }
}

```

```

        enum Processor;
        enum VirtualProcessor;
        enum Memory;
        enum Device;
        enum Bus;
        enum VirtualBus;
    }

    doc /*
        * AADL Component definition
        */

    /*
        * We define a general concept of a component,
        * it refines the notion of SysMLv2 part and adds the
        * notion
        * of component category.
        */

    abstract part def Component specializes Part, Item {
        attribute category : Component_Category;
    }
}

```

A.4 Component Categories

In this section, we show how to define each component category defined in the AADLv2 standard document using the SysMLv2 language.

They follow the same pattern: for component category CAT,

- We define `Cat` as a specialization of `Component`. This specialization has two sets of constraints on its ports and parts expressed as constraints.
- We define `checkCatPorts` (resp. `checkCatParts`) in two steps, first as an iteration on all ports (resp. parts), then as a check on one port (resp. part).

These definitions follow carefully the definition from the AADLv2.3 standard [1] in chapters 5, 6, and 7.

A.4.1 AADLAbstract.sysml

```
package AADLAbstract {  
  
    /* SysMLv2 base library */  
    import SysML::*;  
    import Parts::* ;  
    import Ports::* ;  
    import Connections::*;  
  
    /* KerML base library */  
    private import Base::Anything;  
    private import ScalarValues::*;  
    private import ControlFunctions::*;  
    private import CollectionFunctions::*;  
  
    /* AADLv2 SysML library */  
    import AADL::*;  
  
    /*  
     * Abstract Component Category  
     */  
  
    abstract part def Abstract specializes Component {  
        attribute redefines category =  
            Component_Category::Abstract;  
        assert constraint {  
            checkAbstractPorts (ownedPorts) &  
            checkAbstractParts (subparts)  
        }  
    }  
  
    /* Validity of ports of a abstract */  
    constraint checkAbstractPorts {  
        in p : Port[0..*];  
        p->forall { in x : Port ;  
            checkAbstractPort(x) }  
    }  
}
```

```

constraint checkAbstractPort {
    in p : Port;
    p hastype InPort
}

/* Validity of parts of a abstract */
constraint checkAbstractPart {
    in p : Port;

    p hastype Abstract |
    p hastype System |

    p hastype Process |
    p hastype Thread |
    p hastype ThreadGroup |
    p hastype Subprogram |
    p hastype SubprogramGroup |
    p hastype Data |

    p hastype Processor |
    p hastype VirtualProcessor |
    p hastype Memory |
    p hastype Device |
    p hastype Bus |
    p hastype VirtualBus
}

constraint checkAbstractParts {
    in p : Port;
    p->forall { in x : Part ;
        checkAbstractPart(x) }
}

}

```

A.4.2 AADLBus.sysml

```

package AADLBus {

    /* SysMLv2 base library */
    import SysML::*;
    import Parts::* ;
    import Ports::* ;
    import Connections::*;

    /* KerML base library */
    private import Base::Anything;
    private import ScalarValues::*;
    private import ControlFunctions::*;
    private import CollectionFunctions::*;

    /* AADLv2 SysML library */
    import AADL::*;
}

```

```

/*
 * Bus Component Category
*/

part def Bus :> Component {
    attribute redefines category =
        Component_Category::Bus;
    assert constraint {
        checkBusPorts
            (ownedPorts) &
            checkBusParts (subparts)
    }
}

/* Validity of ports of a Processor */
constraint checkBusPorts {
    in p : Port [0..*];
    p->forall { in x : Port ; checkBusPort(x) }
}

constraint checkBusPort {
    in p : Port;
    p hastype InPort
}

/* Validity of parts of a Processor */
constraint checkBusPart {
    in p : Port;

    p hastype Abstract |
    p hastype VirtualBus
}

constraint checkBusParts {
    in p : Port;
    p->forall { in x : Part ; checkBusPart(x) }
}
}

```

A.4.3 AADLData.sysml

```

package AADLData {

    /* SysMLv2 base library */
    import SysML::*;
    import Parts::* ;
    import Ports::* ;
    import Connections::*;

    /* KerML base library */
    private import Base::Anything;
}

```

```

private import ScalarValues::*;
private import ControlFunctions::*;
private import CollectionFunctions::*;

/* AADLv2 SysML library */
import AADL::*;

/*
 * Data Component Category
 */

part def Data specializes Component {
    attribute redefines category =
        Component_Category::Data;

    assert constraint {
        checkDataPorts (ownedPorts) &
        checkDataParts (subparts)
    }

}

/* Validity of ports of a Data */
constraint checkDataPorts {
    in p : Port [0..*];
    p->forall { in x : Port ; checkDataPort(x) }
}

constraint checkDataPort {
    in p : Port;
    p hastype InPort
}

/* Validity of parts of a Data */
constraint checkDataPart {
    in p : Port;

    p hastype Abstract |
    p hastype Data |
    p hastype Subprogram
}

constraint checkDataParts {
    in p : Port;
    p->forall { in x : Part ; checkDataPart(x) }
}
}

```

A.4.4 AADLDevice.sysml

```

package AADLDevice {

    /* SysMLv2 base library */
    import SysML::*;

```

```

import Parts::* ;
import Ports::* ;
import Connections::*;

/* KerML base library */
private import Base::Anything;
private import ScalarValues::*;
private import ControlFunctions::*;
private import CollectionFunctions::*;

/* AADLv2 SysML library */
import AADL::*;

/*
 * Device Component Category
 */

part def Device :> Component {
    attribute redefines category =
        Component_Category::Device;
    assert constraint {
        checkDevicePorts
            (ownedPorts) &
        checkDeviceParts
            (subparts)
    }
}

/* Validity of ports of a Processor */
constraint checkDevicePorts {
    in p : Port[0..*];
    p->forall { in x : Port ; checkDevicePort(x) }
}

constraint checkDevicePort {
    in p : Port;
    p hastype InPort
}

/* Validity of parts of a Processor */
constraint checkDevicePart {
    in p : Port;

    p hastype Abstract |
    p hastype Data |
    p hastype VirtualBus |
    p hastype Bus
}

constraint checkDeviceParts {
    in p : Port;
    p->forall { in x : Part ; checkDevicePart(x) }
}

```

```
}  
}
```

A.4.5 AADLMemory.sysml

```
package AADLMemory {  
  
    /* SysMLv2 base library */  
    import SysML::*;  
    import Parts::* ;  
    import Ports::* ;  
    import Connections::*;  
  
    /* KerML base library */  
    private import Base::Anything;  
    private import ScalarValues::*;  
    private import ControlFunctions::*;  
    private import CollectionFunctions::*;  
  
    /* AADLv2 SysML library */  
    import AADL::*;  
  
    /*  
     * Memory Component Category  
     */  
  
    part def Memory :> Component {  
        attribute redefines category =  
            Component_Category::Memory;  
        assert constraint {  
            checkMemoryPorts  
                (ownedPorts) &  
            checkMemoryParts  
                (subparts)  
        }  
    }  
  
    /* Validity of ports of a Processor */  
    constraint checkMemoryPorts {  
        in p : Port [0..*];  
        p->forall { in x : Port ; checkMemoryPort(x) }  
    }  
  
    constraint checkMemoryPort {  
        in p : Port;  
        p hastype InPort  
    }  
  
    /* Validity of parts of a Memory */  
    constraint checkMemoryPart {  
        in p : Port;  
  
        p hastype Abstract |
```

```

        p hastype Memory |
        p hastype VirtualBus |
        p hastype Bus
    }

    constraint checkMemoryParts {
        in p : Port;
        p->forAll { in x : Part ; checkMemoryPart(x) }
    }
}

```

A.4.6 AADLProcess.sysml

```

package AADLProcess {

    /* SysMLv2 base library */
    import SysML::*;
    import Parts::* ;
    import Ports::* ;
    import Connections::*;

    /* KerML base library */
    private import Base::Anything;
    private import ScalarValues::*;
    private import ControlFunctions::*;
    private import CollectionFunctions::*;

    /* AADLv2 SysML library */
    import AADL::*;

    /*
     * Process Component Category
    */

    part def Process specializes Component {
        attribute redefines category =
            Component_Category::Process;

        assert constraint {
            checkProcessPorts (ownedPorts) &
            checkProcessParts (subparts)
        }
    }

    /* Validity of ports of a process */
    constraint checkProcessPorts {
        in p : Port [0..*];
        p->forAll { in x : Port ; checkProcessPort(x) }
    }

    constraint checkProcessPort {
        in p : Port;
        p hastype InPort
    }
}

```

```

/* Validity of parts of a process */
constraint checkProcessPart {
    in p : Port;

    p hastype Abstract |
    p hastype Thread |
    p hastype ThreadGroup |
    p hastype Subprogram |
    p hastype SubprogramGroup |
    p hastype Data
}

constraint checkProcessParts {
    in p : Port;
    p->forall { in x : Part ; checkProcessPart(x) }
}
}

```

A.4.7 AADLProcessor.sysml

```

package AADLProcessor {

    /* SysMLv2 base library */
    import SysML::*;
    import Parts::* ;
    import Ports::* ;
    import Connections::*;

    /* KerML base library */
    private import Base::Anything;
    private import ScalarValues::*;
    private import ControlFunctions::*;
    private import CollectionFunctions::*;

    /* AADLv2 SysML library */
    import AADL::*;

    /*
     * Processor Component Category
    */

    part def Processor :> Component {
        attribute redefines category =
            Component_Category::Processor;
        assert constraint {
            checkProcessorPorts (ownedPorts) &
            checkProcessorParts (subparts)
        }
    }

    /* Validity of ports of a Processor */
    constraint checkProcessorPorts {

```

```

        in p : Port[0..*];
        p->forall { in x : Port ; checkProcessorPort(x)
        }
    }

    constraint checkProcessorPort {
        in p : Port;
        p hastype InPort
    }

    /* Validity of parts of a Processor */
    constraint checkProcessorPart {
        in p : Port;

        p hastype Abstract |
        p hastype VirtualProcessor |
        p hastype Memory |
        p hastype Bus |
        p hastype VirtualBus
    }

    constraint checkProcessorParts {
        in p : Port;
        p->forall { in x : Part ; checkProcessorPart(x)
        }
    }
}

```

A.4.8 AADLSubprogram.sysml

```

package AADLSubprogram {

    /* SysMLv2 base library */
    import SysML::*;
    import Parts::* ;
    import Ports::* ;
    import Connections::*;

    /* KerML base library */
    private import Base::Anything;
    private import ScalarValues::*;
    private import ControlFunctions::*;
    private import CollectionFunctions::*;

    /* AADLv2 SysML library */
    import AADL::*;

    /*
     * Subprogram Component Category
     */

    part def Subprogram specializes Component {
        attribute redefines category =
            Component_Category::Subprogram;
    }
}

```

```

        assert constraint {
            checkSubprogramPorts
                (ownedPorts) &
            checkSubprogramParts
                (subparts)
        }
    }

    /* Validity of ports of a Subprogram */
    constraint checkSubprogramPorts {
        in p : Port[0..*];
        p->forall { in x : Port ;
            checkSubprogramPort(x) }
    }

    constraint checkSubprogramPort {
        in p : Port;
        p hastype InPort
    }

    /* Validity of parts of a Subprogram */
    constraint checkSubprogramPart {
        in p : Port;

        p hastype Abstract |
        p hastype Subprogram |
        p hastype Data
    }

    constraint checkSubprogramParts {
        in p : Port;
        p->forall { in x : Part ;
            checkSubprogramPart(x) }
    }
}

```

A.4.9 AADLSubprogramGroup.sysml

```

package AADLSubprogramGroup {

    /* SysMLv2 base library */
    import SysML::*;
    import Parts::* ;
    import Ports::* ;
    import Connections::*;

    /* KerML base library */
    private import Base::Anything;
    private import ScalarValues::*;
    private import ControlFunctions::*;
    private import CollectionFunctions::*;
}

```

```

    /* AADLv2 SysML library */
import AADL::*;

/*
 * SubprogramGroup Component Category
*/

part def SubprogramGroup specializes Component {
    attribute redefines category =
        Component_Category::SubprogramGroup;

        assert constraint {
            checkSubprogramGroupPorts
                (ownedPorts) &
            checkSubprogramGroupParts
                (subparts)
        }

}

    /* Validity of ports of a SubprogramGroup */
constraint checkSubprogramGroupPorts {
    in p : Port[0..*];
    p->forall { in x : Port ;
        checkSubprogramGroupPort(x) }
}

constraint checkSubprogramGroupPort {
    in p : Port;
    p hastype InPort
}

    /* Validity of parts of a SubprogramGroup */
constraint checkSubprogramGroupPart {
    in p : Port;

    p hastype Abstract |
    p hastype Subprogram |
    p hastype SubprogramGroup |
    p hastype Data
}

constraint checkSubprogramGroupParts {
    in p : Port;
    p->forall { in x : Part ;
        checkSubprogramGroupPart(x) }
}
}

```

A.4.10 AADLThread.sysml

```

package AADLThread {

    /* SysMLv2 base library */

```

```

import SysML::*;
import Parts::* ;
import Ports::* ;
import Connections::*;

/* KerML base library */
private import Base::Anything;
private import ScalarValues::*;
private import ControlFunctions::*;
private import CollectionFunctions::*;

/* AADLv2 SysML library */
import AADL::*;

/*
 * Thread Component Category
 */

part def Thread specializes Component {
    attribute redefines category =
        Component_Category::Thread;

        assert constraint {
            checkThreadPorts
                (ownedPorts) &
            checkThreadParts
                (subparts)
        }

}

/* Validity of ports of a Thread */
constraint checkThreadPorts {
    in p : Port[0..*];
    p->forall { in x : Port ;
        checkThreadPort(x) }
}

constraint checkThreadPort {
    in p : Port;
    p hastype InPort
}

/* Validity of parts of a Thread */
constraint checkThreadPart {
    in p : Port;

    p hastype Abstract |
    p hastype Subprogram |
    p hastype SubprogramGroup |
    p hastype Data
}

```

```

        constraint checkThreadParts {
            in p : Port;
            p->forall { in x : Part ;
                checkThreadPart(x) }
        }
    }
}

```

A.4.11 AADLThreadGroup.sysml

```

package AADLThreadGroupGroup {

    /* SysMLv2 base library */
    import SysML::*;
    import Parts::* ;
    import Ports::* ;
    import Connections::*;

    /* KerML base library */
    private import Base::Anything;
    private import ScalarValues::*;
    private import ControlFunctions::*;
    private import CollectionFunctions::*;

    /* AADLv2 SysML library */
    import AADL::*;

    /*
     * ThreadGroup Component Category
     */

    part def ThreadGroup specializes Component {
        attribute redefines category =
            Component_Category::ThreadGroup;

        assert constraint {
            checkThreadGroupPorts
                (ownedPorts) &
            checkThreadGroupParts
                (subparts)
        }
    }

    /* Validity of ports of a ThreadGroup */
    constraint checkThreadGroupPorts {
        in p : Port[0..*];
        p->forall { in x : Port ;
            checkThreadGroupPort(x) }
    }

    constraint checkThreadGroupPort {
        in p : Port;
        p hastype InPort
    }
}

```

```

        /* Validity of parts of a ThreadGroup */
        constraint checkThreadGroupPart {
            in p : Port;

            p hastype Abstract |
            p hastype Thread |
            p hastype ThreadGroup |
            p hastype Subprogram |
            p hastype SubprogramGroup |
            p hastype Data
        }

        constraint checkThreadGroupParts {
            in p : Port;
            p->forall { in x : Part ;
                checkThreadGroupPart(x) }
        }
    }
}

```

A.4.12 AADLSystem.sysml

```

package AADLSystem {

    /* SysMLv2 base library */
    import SysML::*;
    import Parts::* ;
    import Ports::* ;
    import Connections::*;

    /* KerML base library */
    private import Base::Anything;
    private import ScalarValues::*;
    private import ControlFunctions::*;
    private import CollectionFunctions::*;

    /* AADLv2 SysML library */
    import AADL::* ;

    /*
     * System Component Category
     */

    part def System :> Component {
        attribute redefines category =
            Component_Category::System;
        assert constraint {
            checkSystemPorts (ownedPorts) &
            checkSystemParts (subparts)
        }
    }

    /* Validity of ports of a system */
    constraint checkSystemPorts {

```

```

        in p : Port [0..*];
        p->forall { in x : Port ; checkSystemPort(x) }
    }

    constraint checkSystemPort {
        in p : Port;
        p hastype InPort
    }

    /* Validity of parts of a system */
    constraint checkSystemPart {
        in p : Port;

        p hastype Abstract |
        p hastype System |

        p hastype Process |
        p hastype ThreadGroup |
        p hastype Subprogram |
        p hastype SubprogramGroup |
        p hastype Data |

        p hastype Processor |
        p hastype VirtualProcessor |
        p hastype Memory |
        p hastype Device |
        p hastype Bus |
        p hastype VirtualBus
    }

    constraint checkSystemParts {
        in p : Port;
        p->forall { in x : Part ; checkSystemPart(x) }
    }
}

```

A.4.13 AADLVirtualBus.sysml

```

package AADLVirtualBus {

    /* SysMLv2 base library */
    import SysML::*;
    import Parts::* ;
    import Ports::* ;
    import Connections::*;

    /* KerML base library */
    private import Base::Anything;
    private import ScalarValues::*;
    private import ControlFunctions::*;
    private import CollectionFunctions::*;

    /* AADLv2 SysML library */

```

```

import AADL::*;

/*
 * VirtualBus Component Category
 */

part def VirtualBus :> Component {
    attribute redefines category =
        Component_Category::VirtualBus;
    assert constraint {
        checkVirtualBusPorts
            (ownedPorts) &
        checkVirtualBusParts
            (subparts)
    }
}

/* Validity of ports of a VirtualBus */
constraint checkVirtualBusPorts {
    in p : Port [0..*];
    p->forall { in x : Port ;
        checkVirtualBusPort(x) }
}

constraint checkVirtualBusPort {
    in p : Port;
    p hastype InPort
}

/* Validity of parts of a VirtualBus */
constraint checkVirtualBusPart {
    in p : Port;

    p hastype Abstract |
    p hastype VirtualBus
}

constraint checkVirtualBusParts {
    in p : Port;
    p->forall { in x : Part ;
        checkVirtualBusPart(x) }
}
}

```

A.4.14 AADLVirtualProcessor.sysml

```

package AADLVirtualProcessor {

    /* SysMLv2 base library */
    import SysML::*;
    import Parts::* ;
    import Ports::* ;
    import Connections::*;
}

```

```

/* KerML base library */
private import Base::Anything;
private import ScalarValues::*;
private import ControlFunctions::*;
private import CollectionFunctions::*;

/* AADLv2 SysML library */
import AADL::*;

/*
 * VirtualProcessor Component Category
*/

part def VirtualProcessor :> Component {
    attribute redefines category =
        Component_Category::VirtualProcessor;
    assert constraint {
        checkVirtualProcessorPorts
            (ownedPorts) &
        checkVirtualProcessorParts
            (subparts)
    }
}

/* Validity of ports of a VirtualProcessor */
constraint checkVirtualProcessorPorts {
    in p : Port [0..*];
    p->forall { in x : Port ;
        checkVirtualProcessorPort(x) }
}

constraint checkVirtualProcessorPort {
    in p : Port;
    p hastype InPort
}

/* Validity of parts of a VirtualProcessor */
constraint checkVirtualProcessorPart {
    in p : Port;

    p hastype Abstract |
    p hastype VirtualProcessor |
    p hastype VirtualBus
}

constraint checkVirtualProcessorParts {
    in p : Port;
    p->forall { in x : Part ;
        checkVirtualProcessorPart(x) }
}
}

```

References/Bibliography

URLs are valid as of the publication date of this document.

- [1] AS2-C committee SAE International. Architecture Analysis & Design Language (AADL). Technical Report AS5506D, April 2022.
- [2] SysMLv2 SST. OMG Systems Modeling Language™ (SysML) v2 – draft. Technical report, April 2023.

REPORT DOCUMENTATION PAGE			<i>Form Approved</i> <i>OMB No. 0704-188</i>
Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington Headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188), Washington, DC 20503.			
1. AGENCY USE ONLY (Leave Blank)	2. REPORT DATE April 2023	3. REPORT TYPE AND DATES COVERED Final	
4. TITLE AND SUBTITLE AADLv2 library for SysMLv2		5. FUNDING NUMBERS FA8702-15-D-0002	
6. AUTHORS Jérôme Hugues			
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Software Engineering Institute Carnegie Mellon University Pittsburgh, PA 15213		8. PERFORMING ORGANIZATION REPORT NUMBER CMU/SEI-2023-TN-001	
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES) SEI Administrative Agent AFLCMC/AZS 5 Elgin Street Hanscom AFB, MA 01731-2100		10. SPONSORING/MONITORING AGENCY REPORT NUMBER N/A	
11. SUPPLEMENTARY NOTES			
12A. DISTRIBUTION/AVAILABILITY STATEMENT Unclassified/Unlimited, DTIC, NTIS		12B. DISTRIBUTION CODE	
13. ABSTRACT (MAXIMUM 200 WORDS) The report is a proof-of-concept to defines elements of the Architecture Analysis and Design Language v2.3 (AADLv2) using the SysMLv2 textual syntax. SysMLv2 is the future revision of the SysML standard. It provides a lean syntax and rigorous static semantics to capture a system design. SysMLv2 also has the concept of library to specialize its building blocks for a specific domain. SEI evaluated this capability to represent AADLv2 concepts using this alternate notation. The appendix of this document provides the source code of this library, tested with the SysMLv2 pilot implementation. It shows that SysMLv2's power of expression is sufficient to capture AADLv2 concepts in a uniform way.			
14. SUBJECT TERMS		15. NUMBER OF PAGES 29	
16. PRICE CODE			
17. SECURITY CLASSIFICATION OF REPORT Unclassified	18. SECURITY CLASSIFICATION OF THIS PAGE Unclassified	19. SECURITY CLASSIFICATION OF ABSTRACT Unclassified	20. LIMITATION OF ABSTRACT UL

NSN 7540-01-280-5500

Standard Form 298 (Rev. 2-89) Prescribed by ANSI Std. Z39-18
298-102