AVM Component Model Specification
Version 2.5

Adam Nagel, Sandeep Neema, Mike Myers, Robert Owens, Zsolt Lattmann
Institute for Software Integrated Systems (ISIS)
Vanderbilt University
adam@isis.vanderbilt.edu

Dan Finke
Applied Research Laboratory (ARL)
Pennsylvania State University

Developed for the DARPA Adaptive Vehicle Make (AVM) Program

August 4, 2014

Contents

List of Figures

Chapter 1

Introduction

1.1 Adaptive Vehicle Make (AVM) Overview

The AVM program has developed tools (referred to as META tools)for designing Cyber Physical (or Mechatronic) Systems. Exemplified by modern Amphibious and Ground Military Vehicles, these systems are increasingly complex, take much longer to design and build, and are increasingly costlier. The vision of the AVM program is to revolutionize the design methodology of such systems and reduce the design time to 1/5th of the traditional systems engineering V methodology (MILSTD-499).

The META tools realize this vision by advancing a novel design flow geared around the following core concepts:

- **Component-Based Design** enables design cycle compression by reuse of existing technology and knowledge, encapsulated in integratable and customizable components that can be rapidly used in a design. Components in CPS are heterogeneous, span multiple domains (physical thermal, mechanical, electrical, fluid , .. and computational software, computing platforms), and require multiple models to soundly represent the behavior, geometry, and interfaces, at multiple levels of abstractions.
- **Design Space Exploration** using explicit representation of design choices and parameterized components. META tools enable a designer to systematically engineer a flexible and comprehensive design space for sub-systems and system that can be explored for satisfying requirements. The design spaces for subsystems and systems are assets that encapsulate design knowledge, which can be reused in a context different from which it was originally created. *Multi-Scale Design Space Exploration* incorporates multiple analysis methods that trade accuracy with computation time for exploring the large design spaces, and iteratively converge over to design points of interest.
- **Testbenches for Design Evaluation** capture requirements in a form that can be automatically evaluated for a system-under-test, using a large set of domain-specific analyses ranging from hybrid dynamics simulation, software platform timing simulation, geometric parameter evaluation, finite element analysis, probabilistic certificates of correctness, among others. The model composition tools included in META operate over defined testbenches and synthesize artifacts necessary for executing analysis in pertinent domain tools such as Dymola, Pro-e/Creo, Truetime, Qualitative Envisionment, Abaqus, etc..

1.2 AVM Component Overview

A central idea in engineering is that complex systems are built by assembling less complex components (building blocks). One approach, Component-based design offers advantages with respect to monolithic design such as reuse of design solutions, modular analysis and validation, reconfigurability and complexity management. In the context of component-based design the concept of component is much richer than the concept of containment. As opposed to monolithic design synthesis, it enables the formulation of the design

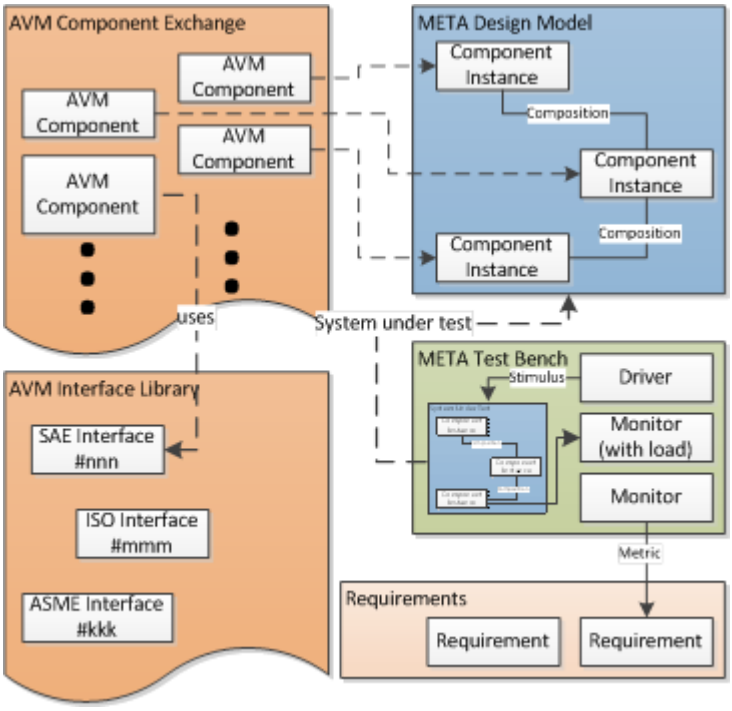

Figure 1.1: AVM Design Flow

problem the following manner: *design a system meeting the requirements using a set of components*, where requirements specify essential properties of the system.

Component-based design requires a composition framework that satisfies two requirements:

- *Composability*: meaning that components do not change their essential properties after composed with other components.
- *Compositionality*: meaning that the essential properties of the system can be computed from the properties of its components.

Composition frameworks enable constructivity: the synthesis of systems with predictable properties from components with known properties.

Within this context, AVM components are heterogeneous cyber-physical system components, spanning exclusively cyber, exclusively physical, as well as cyber-physical. Meaningful AVM components capture non-trivial and re-usable design knowledge, have interfaces for composition, and represent the design knowledge in a formalism that enables automated composition. AVM Components in general, could be composed of many parts and subparts; however, for the purposes of the AVM program and this document, AVM components are considered as atomic (black-box) units that can be acquired off-the shelf or custom built, and can be composed together into a system.

An AVM Component model is an integrated multi-model including an integration model (a model wrapper) that represents the key properties, parameters, compositional interfaces and their association with domain models; and a number of domain models that represent the dynamics, geometry, computational (cyber), and manufacturing of the component at multiple levels of abstraction and fidelity.

The figure 1.2 illustrates an AVM component model consisting of a set of properties (e.g. Weight, Height, Maximum Power, etc.), a set of parameters (properties that are user modifiable), a set of interfaces which can be atomic (e.g. Throttle Signal Port) as well as aggregate (e.g. Power Out Aggregate Interface consisting of a Rotational Power Port, and a Bell Housing Structural Port), a set of embedded domain models (e.g. High-Fidelity Modelica Dynamics Model, FEA-Ready CAD Model, etc.) representing different fidelities and abstraction of the component dynamics and geometry. The domain models are developed and represented in external tools and languages (i.e. Modelica/Dymola, CAD/Pro-E). The integration model

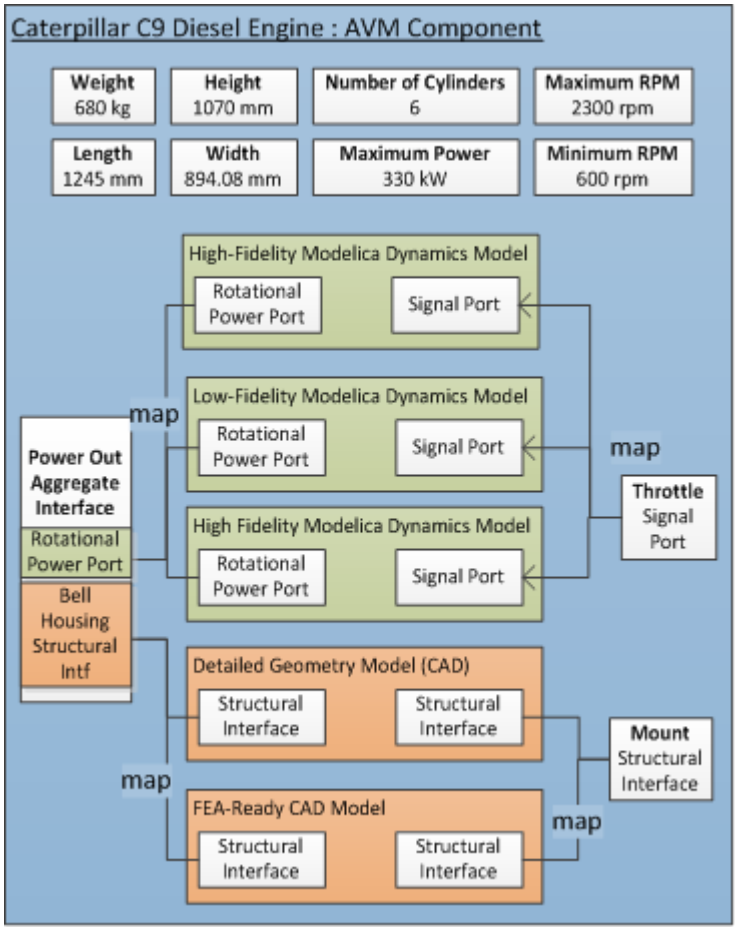

Figure 1.2: AVM Component Model Conceptualization

contains references to the domain models (through a URI) and explicitly represents domain-model specific interfaces (e.g. Rotational Power Port, Structural Interface, etc.) that are mapped into the external interfaces of the Component.

1.3 Scope

The scope of this document is limited to the specification of AVM Component Model. The AVM Design Models that are composed together using AVM Components, and the META Language, Tools, and Design Flow are considered outside the scope of this document. The domain modeling tools (such as Dymola, Pro/E, ...)and languages (Modelica, CAD,) are outside the scope of this document, however, modeling guidelines that must be adhered to when creating domain models are included in this document.

1.4 Purpose

The purpose of this document is to serve as a reference guide for AVM component developers enabling them to rapidly author valid components that can be used within the META Tools. The document purports to define and explain AVM Component Models, with specific emphasis on the integration model, and introduce methods to author and validate AVM Component models.

1.5 Document Organization

The rest of this document is organized as follows.

1. Chapter 1 provides a very brief overview of the AVM tools and rationale for the AVM component model
2. Chapter 2 provides a detailed description of the syntax and semantics of the AVM Component Model and its constituent parts
3. Chapter 3 describes a framework for authoring AVM Components
4. Chapter 4 describes a validation suite and automated testing framework for validating component designs

Chapter 2

AVM Component Model

2.1 AVM Component by Example

In order to better frame the description example AVM components from the ground vehicle domain are introduced. These will be used in the rest of this document to exemplify concepts described.

2.1.1 Engine

An Engine (see figure 2.1) serves as a good reference for an AVM component, as it spans cyber (electronic control) and physical, and encompasses multiple physics domains including mechanical, thermal, electrical, and acoustics. The Engine has physical interfaces through which it transfers energy (SAE-1/2 flywheel and flywheel housing), has mount points for installation, and has electronic interfaces for control. The Engine component has non-trivial dynamics and mode-based behavior that can be parameterized and modeled to predict the torque output, fuel consumption, and thermal output, as a function of mode, efficiency, engine-RPM, control settings, and a number of other parameters. The Engine has a complex geometry that can be captured with CAD models to analyze space claims, center-of-gravity, mass distribution, and mount point stresses using finite-element analyses. The Engine is also an example of a component which in itself is a highly complex system and made of several parts and sub-parts, but being a Commercial Off The Shelf (COTS) discrete component is treated as a black-box atomic component. The manufacturing model of such COTS components in AVM simply characterizes discrete attributes such as Cost, Lead Time, Shipping Volume, and Weight.

2.2 Component Model Organization

The AVM Component Model being a multi-model is represented and persisted as a package containing all the artifacts (integration model, domain models, documentation, test articles) needed to use a component within the AVM tools. The integration model is the primary (and mandatory) artifact, and is often referred to as the AVM Component Model (with the acronym ACM) in the rest of this document.

An AVM Component Package may include:

- AVM Component Model *(required)*
- Domain Models, such as:
  - CAD models
  - Modelica models
  - Manufacturing models
- Supplemental artifacts such as:
  - Images

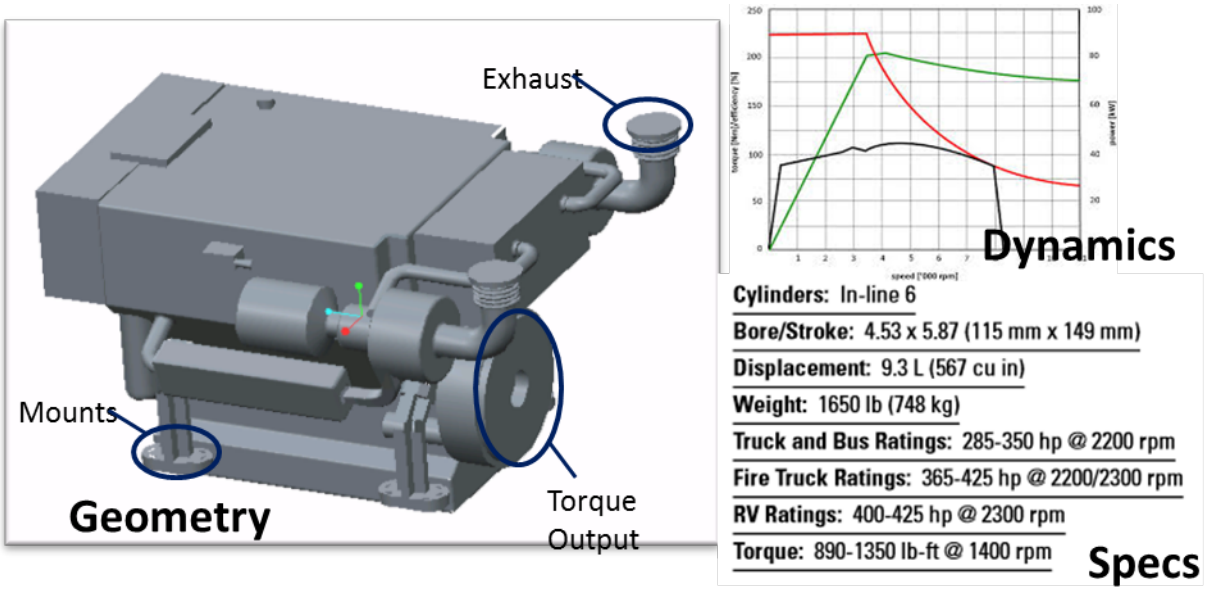

Figure 2.1: Example Engine Component Models, Interfaces, and Specs

Figure 2.2: Example Component Package contents

Diagrams

Documentation

Data Sheets

An AVM Component Package consists of a set of files that support a system designer's use of the AVM Component. These files may be arranged into a folder structure, but these folders must share a common root.

The ACM file serves dual purpose within the AVM Component package. It is the integration model that describes the component interface, references to domain models, and association with domain model ports, but it also serves as the package descriptor. Thus, an ACM file describing the component **must** be included within the root folder. It **must** have a file extension of ".acm" and be the only file with that extension in the root folder. No file other than the ACM is strictly required, but more files are typically included. All other files to be included in the component package **must** be referenced by "ResourceDependency" tags in the ACM file, with the exception of the ACM file itself.

Details of component distribution restrictions, licensing, and more must also be included in the ACM file. The ACM file is syntactically represented as an XML, and its schema is discussed in the following section.

2.3 AVM Component Model Schema

2.3.1 Overview

The **AVM Component Model** captures data and metadata about an AVM Component. From a system designer's perspective, it is a "black box," below which the designer does not need composition details.

An AVM Component Model serves several purposes:

- Capture the metadata of the component, including information on its dependent artifacts
- Define the interfaces and properties of the component for use in composing system models
- Serve as a wrapper for integrating various domain models of the component

2.3.2 Core Concepts

2.3.2.1 Properties and Parameters

The core metadata about a component is expressed using properties and parameters. Properties represent attributes of a component that can not be directly altered by the user of a component, while Parameters represent designed variability of a component. The variability of a component can range from geometric (e.g. dimensions of a parametrized fuel tank), to behavioral (e.g. gain constants of a PID controller), to interface variability (e.g. structural connector types).

Name	Value	Data Type	Unit	Dimensions
efficiency	0.85	Real	N/A	1
case_heat_transfer_area	3.84	Real	m$\hat{2}$	1

Table 2.1: Example Properties of an Engine Component

Name	Def. Value	Min Value	Max Value	Data Type	Unit	Dimensions
output_mount_position_x	1.0	0.4	10.0	Real	N/A	1

Table 2.2: Example Parameters of a Parametric Drive Shaft

2.3.2.2 Connectors

The AVM Component Specification allows multiple types of interfaces currently instantiated as Connectors and Ports. There are different types of Ports such as Power ports, Signal ports, and Structural ports. The Connectors are an aggregation of Ports. Connectors allow component designers to meaningfully group different Ports. For example, a power flow through a component will in most cases will require a physical interface, so it can be meaningfully represented with a Connector that aggregates a Power port, and a Structural port. When the component is used in a design, a single connection made with the Connector automatically and consistently results in the two separate Power and Structural connections. When individual Ports are used as interfaces it becomes the responsibility of the designer to ensure that connections are made in a consistent manner, and creates an implicit requirement from the component designer to the component user and is bound to result in problems. Therefore, the specification recommends usage of Connectors when ports need to be connected simultaneously.

Connector Name	Connector Roles	Port Map	Type
eng_torque_out	Z_Axis	cad.EXT_TORQUE_OUT_axis_z	Axis
	YX_Plane	cad.EXT_TORQUE_OUT_plane_xy	Plane
	YZ_Plane	cad.EXT_TORQUE_OUT_plane_yz	Plane
	brg_02	mod.conn_brg_02	Modelica.Mechanics.Multibody. Interfaces.FlangeWithBearing

Table 2.3: Example Connector of an Engine Component

2.3.2.3 Domain Model Sublanguage

A **Domain Model** is a domain-specific model that describes the component in that domain. Examples of these include CAD models, Modelica behavioral models, cyber models, manufacturing models, and more.

Each **Domain Model** may include instantiations of **Ports**, **Parameters**, and **Metrics**. **Ports** are composition points specific to the domain. **Parameters** are variables of the domain model that may be set prior to the model's use in a composition (simulation, geometric model, etc). **Metrics** are variables that may be checked after a model's use in a composition (mass calculation from a parametric CAD model, max force in a physics simulation, etc).

Domain-specific models are defined as *extensions* of the abstract *DomainModel* baseclass. Composition points are modeled as extensions of the *DomainModelPort* abstract baseclass, and likewise with Parameters and Metrics. They are typically defined within tool-specific namespaces to avoid name collisions (e.g.: *avm.cad*, *avm.modelica*).

Current examples include ModelicaModel, CADModel, CyberModel, and ManufacturingModel, although the spec is extensible to include more domains.

If more than one of a given class of DomainModel is included in the Component package, then these models are considered to be *alternative* representations of the complete component. These representations may vary in fidelity or suitability for a specific type of analysis.

When performing an analysis, users will select between alternative Domain Models by inspecting the Name field.

2.3.2.4 Composition Semantics

Components are designed to be composed with other components via their **Connectors**. When two component connectors are composed, then their corresponding **Role** elements are also matched, and the **DomainPorts** so mapped will be connected together in a generated domain model.

In the example given in **Figure 2.3**, two components each have embedded Domain Models of type **ModelicaModel**. They also each feature **Connector** objects that share a common definition. The **role** objects within each **connector** instance are mapped to the **modelica connectors** of each component's Modelica model. In the generated Modelica model, the corresponding Modelica class representing each component is instantiated, and their connectors are joined by following the *Modelica Connector→Role→Connector→Connector→Role→Mode Connector* chain from the source AVM META composition.

2.3.2.5 Value Sublanguage

The specification includes a sublanguage for expressing values, as seen in the "Values" ClassDiagram (see Figure 2.8). Within components, many elements have values, and these values may be fixed, derived from other sources, parametrically variable by the user, or probabilistic. This sublanguage for expressing values allows a high degree of flexibility in establishing dependent relationships between values in a model.

Each **Value** container contains a characterization of its unit (e.g., meter, mph), datatype (e.g., integer, boolean), and dimensionality (e.g., scalar, vector, matrix). The Value container then contains one **ValueExpression**, which describes how the value should be determined. There are several variations, including fixed, calculated, derived, parametric, and probabilistic. For many of these, key attributes may themselves be expressed using ValueExpressions.

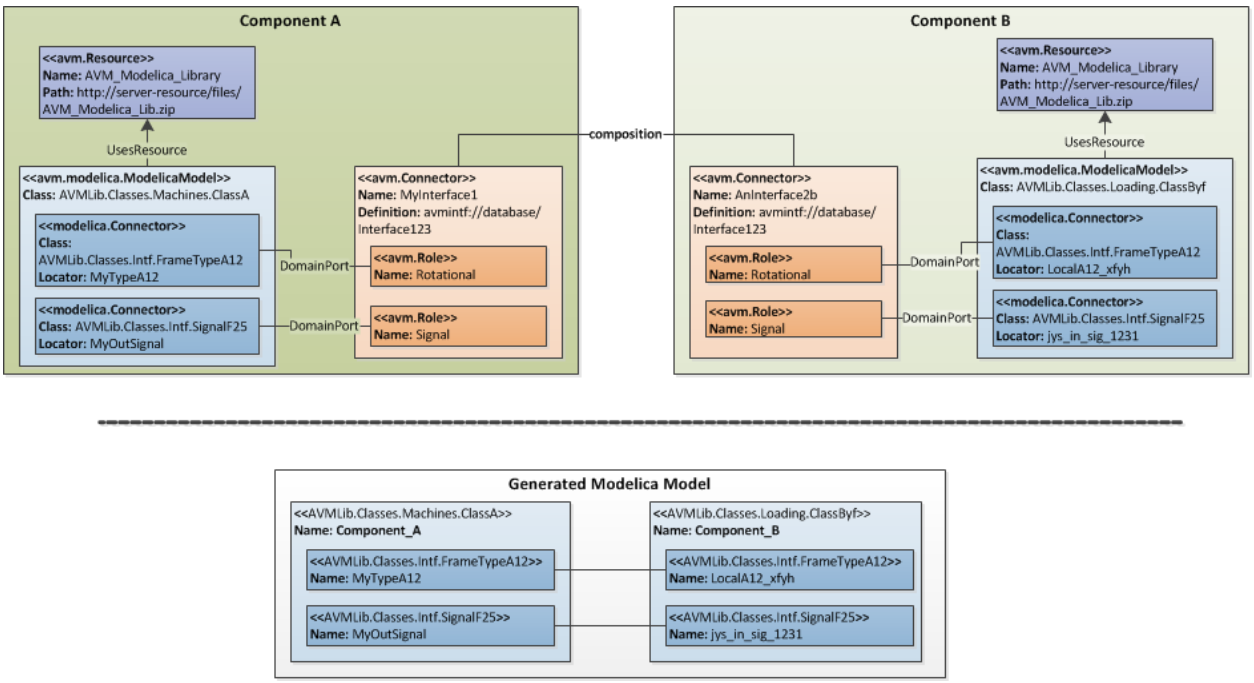

Figure 2.3: Example illustrating composition semantics. At top, the composition of two Component models within an AVM META design tool. At bottom, the resulting composition within a Modelica model generated by those tools.

2.3.3 Implementation

2.3.3.1 File Format

AVM Component Models are stored in files with an extension of .ACM. Their contents are in XML format. An XML schema (in XSD format) enforces the syntax of the spec.

2.3.3.2 Software Library

The AVM Component Model is also supported by software libraries that can parse, validate, and export AVM Component Models. These libraries provide classes specific to AVM Component concepts.

- **Python** An initial library implementation in Python is provided. It was generated using **PyXB**, a tool which generates Python classes from an XML Schema.

  An example model-building program is provided (*test_builder.py*).

- **Java** An initial library implementation in Java is provided. It was generated using **JAXB**, a tool which generates Java classes from an XML Schema.

- **C# for .NET** An implementation in C#, for use with the .NET framework, is provided. It was generated using **Xsd2Code**, a tool which generates C# classes from an XML Schema.

2.3.3.3 Schema

An XML Schema file is provided for checking conformance to the ACM file format.

2.3.3.4 Extending the ACM Format

Although the AVM Component Model format was designed to be extensible, the schema does not yet support the use of new tags. A set of best practices for extending the format remains to be defined.

2.3.4 Classes

This section documents and provides detail description of different classes in the AVM component model schema. The classes are organized in multiple namespaces to avoid name collisions. The core concepts that apply to the component are included in the **avm** namespace, while concepts relevant to domain model wrappers are included in the **cad, modelica, cyber, and manufacturing** namespaces respectively.

2.3.4.1 avm Namespace

The classes in avm Namespace are described below, and their relations are depicted in class diagrams in figures below.

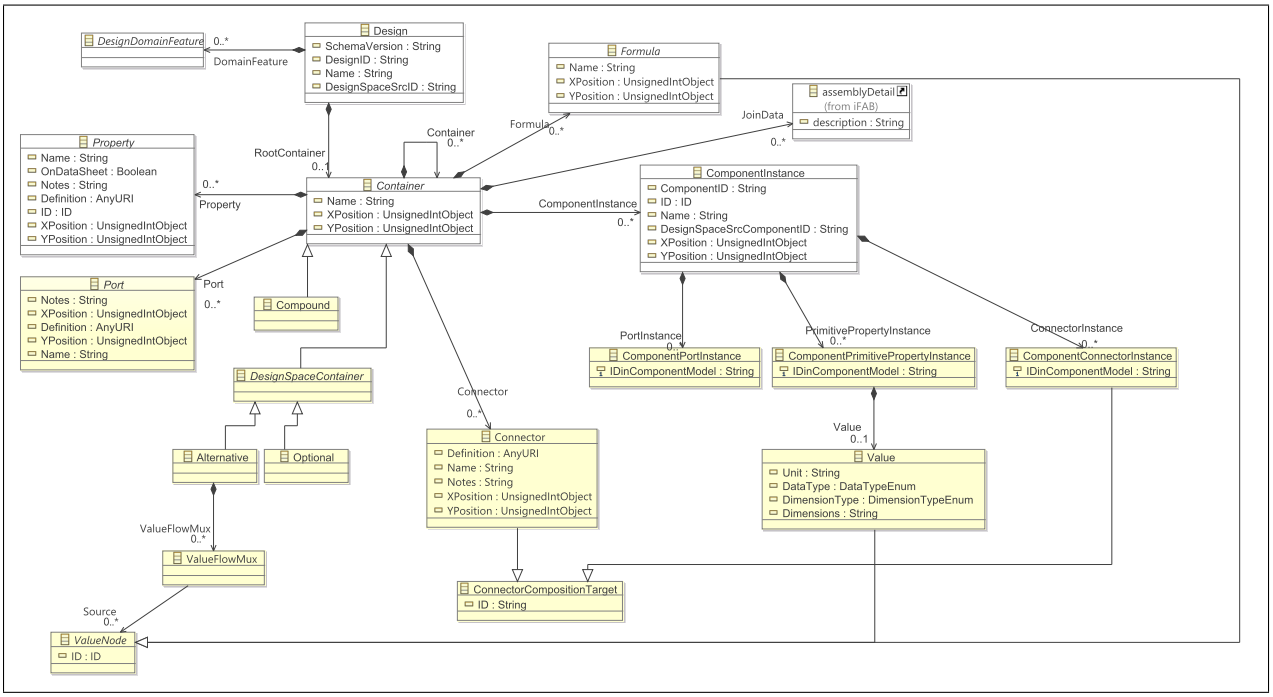

Figure 2.4: avm Namespace: Design diagram

2.3.4.1.1 avm.Component

Component is the root element of an AVM Component definition.

Attribute	Type	Description
Name	String	The name of the component
Version	String	The version number for the component
SchemaVersion	String	The version of the AVM Component schema to which this component conforms
ID	String	A string that uniquely identifies this Component model
Classifications	AnyURI[]	A list of URIs that identify classes to which this component belongs.
Supercedes	String[]	A list of IDs of Components that this component supercedes (is a newer version of).

```
<ns1:Component
    xmlns:ns1="avm" xmlns:ns2="manufacturing"
    xmlns:ns3="modelica" xmlns:ns4="cad"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  Name="engine_compression_ignition_diesel__cat__c9_600hp"
  Version="10_10_2013_17_23"
  SchemaVersion="1.0"
  ID="AVM.Component.2e61b1472b"  >
    <Classifications>Engine_Compression_Ignition_Diesel</Classifications>
    ...
</ns1:Component>
```

2.3.4.1.2 avm.DistributionRestriction

(Abstract) Baseclass for a number of expressions that describe distribution restrictions that apply to the component.

Attribute	Type	Description
Note	String	A note providing details of the restriction.

2.3.4.1.3 avm.SecurityClassification

Subtype of avm.DistributionRestriction
Describes a security classification that applies to the component. If no SecurityClassification element is found, the component is assumed to be unclassified.

Attribute	Type	Description
Level	String	The classification level that applies

2.3.4.1.4 avm.Proprietary

Subtype of avm.DistributionRestriction
Describes an organization that claims this component as proprietary.

Attribute	Type	Description
Organization	String	(Required) The organization claiming that the component is proprietary.

```
  <DistributionRestriction Organization="Ricardo Inc"
     xsi:type="ns1:Proprietary"/>
```

2.3.4.1.5 avm.ITAR

Subtype of avm.DistributionRestriction
Describes an ITAR restriction that applies to this component. If this ITAR element is found, the component is assumed to be ITAR. If no ITAR element is are found, the component is assumed to be non-ITAR.

```
  <DistributionRestriction Level="ITAR" xsi:type="ns1:ITAR"
Notes="Notes about this ITAR restriction."/>
```

2.3.4.1.6 avm.DoDDistributionStatement

Subtype of avm.DistributionRestriction
Describes a Department of Defense (DoD) distribution statement that applies to this component.

Attribute	Type	Description
Type	DoDDistributionStatementEnum	(Required) The DoD distribution statement that applies to this component.

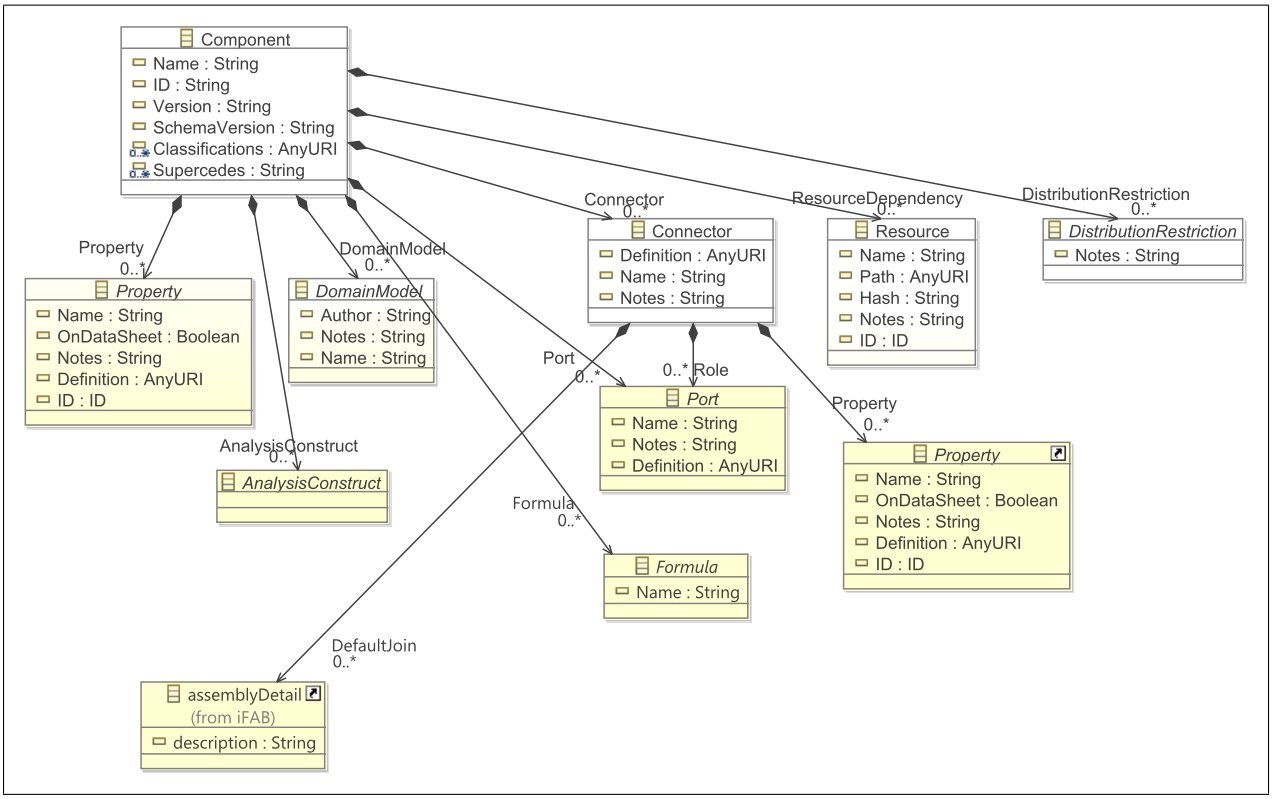

Figure 2.5: avm Namespace: Component diagram

2.3.4.1.7 avm.Property

(Abstract) Captures a property of a component. When instantiated into a component, it may have a fixed value, have a value described by a probability distribution, be parametrically variable at design time, or be derived from other values.

Attribute	Type	Description
Definition	AnyURI	If this property is strongly typed and instantiated from a definition database, this is its definition URI
Name	String	The name of the property, as instantiated
OnDataSheet	Boolean	Selects whether this property should be prominently displayed on "data sheet"-style views of the component
Notes	String	Any notes describing the property
ID	ID	The ID of this property, unique within the scope of the component model

2.3.4.1.8 avm.PrimitiveProperty

A property that is described by a value.

Relation	Type	Description
Value	Value	The Value container for this property

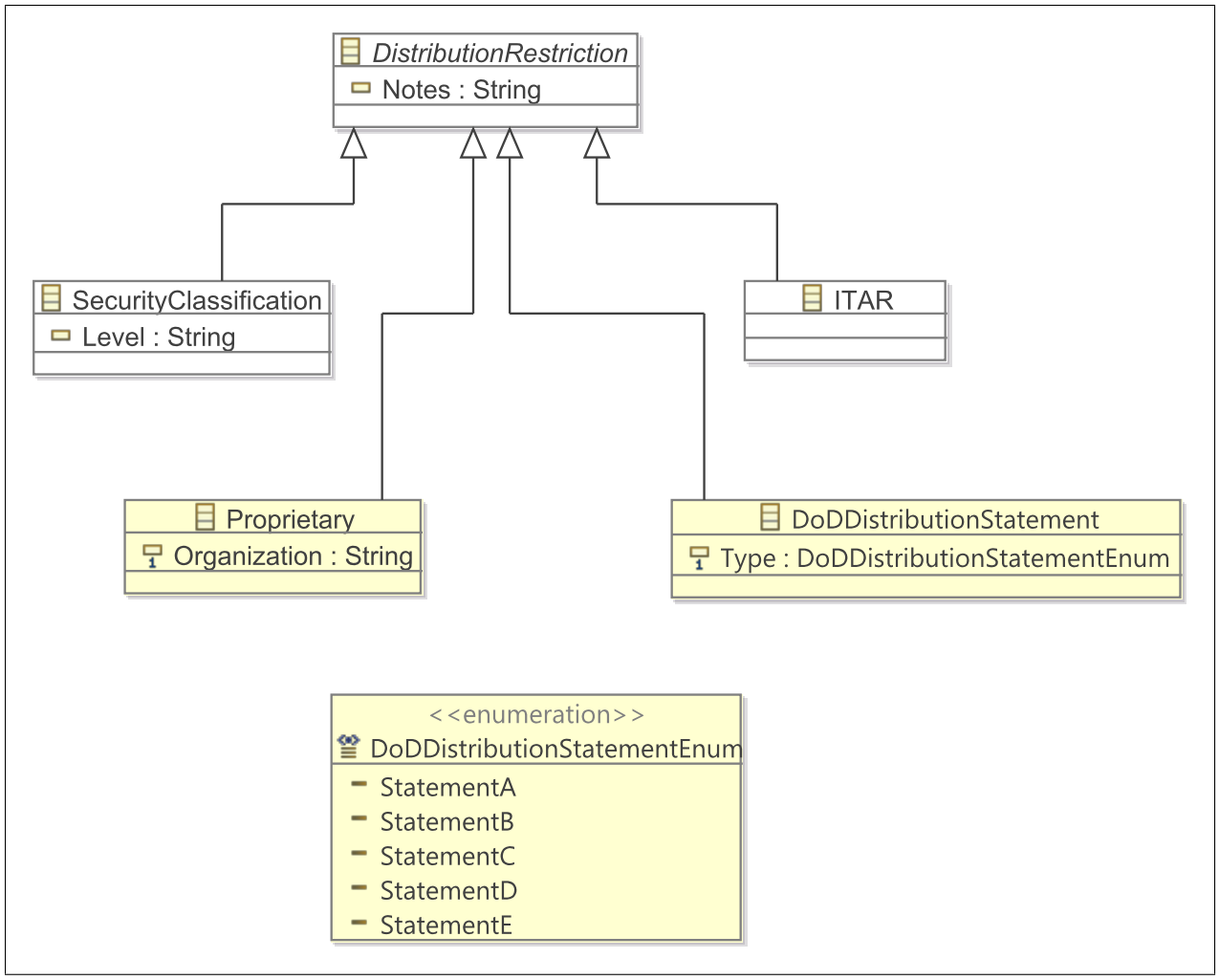

Figure 2.6: avm Namespace: Distributionrestrictions diagram

```
<Property
  Definition="CML_Physical_Quantity\cml_zodb.cml_basic_physicals.Efficiency"
 ID="primitive.cac_performance" Name="cac_performance"
 xsi:type="ns1:PrimitiveProperty">
  <Value>
    ...
  </Value>
</Property>
```

2.3.4.1.9 avm.CompoundProperty

A property that is a container for other properties.

Relation	Type	Description
PrimitiveProperty	PrimitiveProperty	Primitive Property children of this Compound Property
CompoundProperty	CompoundProperty	Compound Property children of this Compound Property

```
<Property Definition="CML_Mapping\cml_zodb.cml_component.CML_Common_Info"
  ID="compound.common_info" Name="common_info"
  xsi:type="ns1:CompoundProperty">
   <CompoundProperty
   Definition="CML_Mapping\cml_zodb.cml_extended_physicals.Bounding_Box"
   ID="compound.common_info.overall_dimensions"
   Name="overall_dimensions">
     <PrimitiveProperty
       Definition="CML_Physical_Quantity\cml_zodb.cml_basic_physicals.Length"
       ID="primitive.common_info.overall_dimensions.height"
       Name="height">
       <Value>...</Value>
     </PrimitiveProperty>
     <PrimitiveProperty
       Definition="CML_Physical_Quantity\cml_zodb.cml_basic_physicals.Length"
       ID="primitive.common_info.overall_dimensions.length"
       Name="length">
       <Value>...</Value>
     </PrimitiveProperty>
     <PrimitiveProperty
       Definition="CML_Physical_Quantity\cml_zodb.cml_basic_physicals.Length"
       ID="primitive.common_info.overall_dimensions.width"
       Name="width">
       <Value>...</Value>
     </PrimitiveProperty>
   </CompoundProperty>
```

2.3.4.1.10 avm.ValueNode

Abstract baseclass representing an object that carries a value, whether as a Value or Formula object.

Attribute	Type	Description
ID	String	(required) An ID, unique within the scope of the component model, used for referring to this value container.

2.3.4.1.11 avm.Formula

Abstract baseclass representing an object that describes a calculation. The result of the calculation must be passed to any **DerivedValue** object that has this calculation as a *ValueSource*, or to any **Formula** that has this calculation as an *Operand*.

Attribute	Type	Description
Name	String	The Name of the formula.
XPosition	Unsigned Int	The X Position of the element, if rendered graphically within its parent model
YPosition	Unsigned Int	The Y Position of the element, if rendered graphically within its parent model

2.3.4.1.12 avm.SimpleFormula

A **SimpleFormula** represents a basic calculation. A number of **ValueNode** objects serve as operands, with the operation determined by the *Operation* attribute.

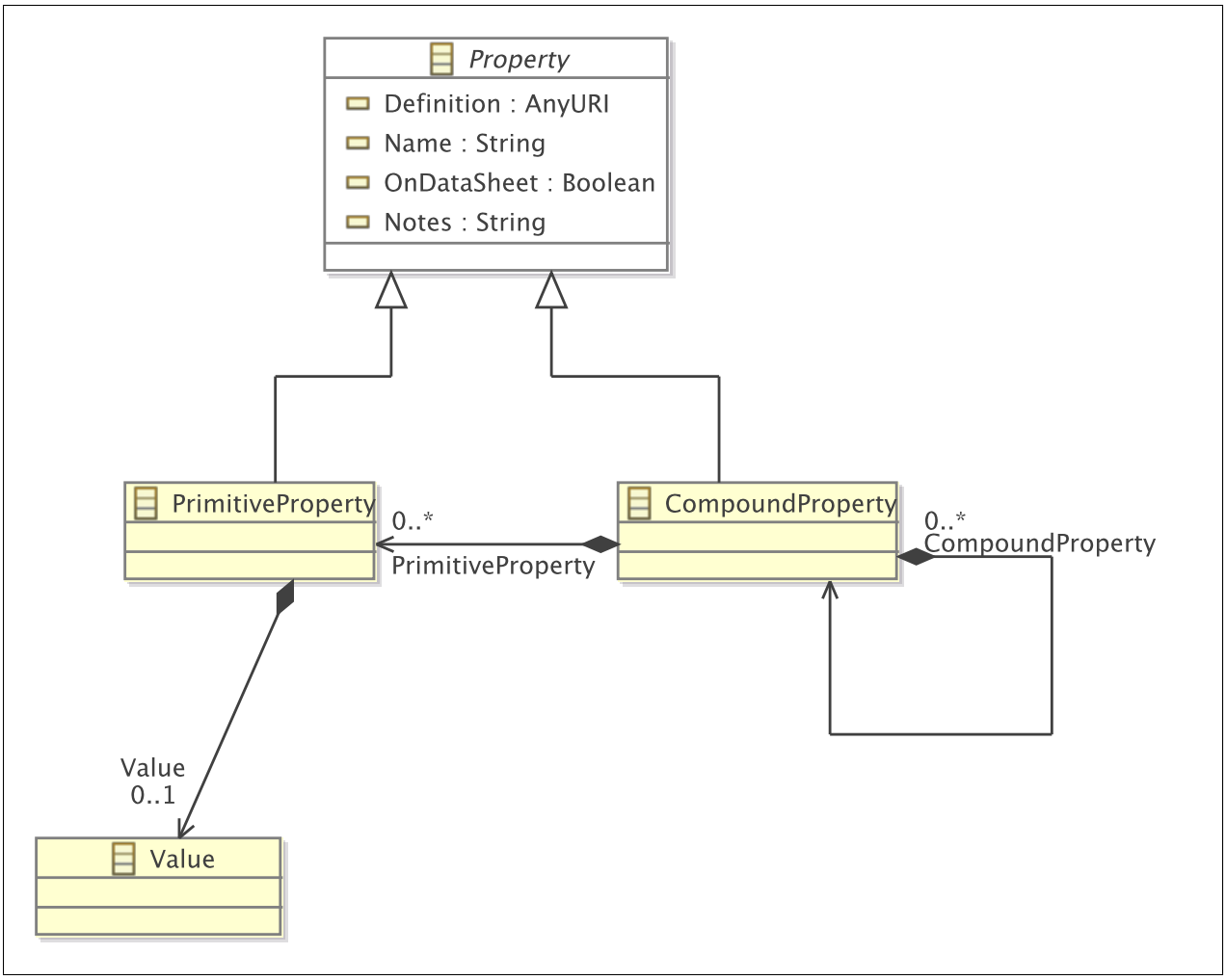

Figure 2.7: avm Namespace: Property diagram

Attribute	Type	Description
Operation	SimpleFormulaOperation	The operation to be performed on the operands.

Relation	Type	Description
Operand	ValueNode	A ValueNode that will serve as an operand to the calculation.

2.3.4.1.13 avm.ComplexFormula

A **ComplexFormula** represents a calculation described by an *Expression*.

Attribute	Type	Description
Expression	String	The expression to be evaluated with the included operands. Must obey ComplexFormula Expression Syntax. The result of this expression will be calculated and considered the value of this **ValueNode**, so no assigment operator is required.

Relation	Type	Description
Operand	Operand	A symbol and ValueSource that represents an operand of the expression

2.3.4.1.14 avm.Operand

An **Operand** represents a symbol and value to be used in a **ComplexFormula** expression.

Attribute	Type	Description
Symbol	String	The symbol for the operand. The **ComplexFormula** *Expression* will refer to this value by its symbol. The symbol must not start with a number or include any spaces or mathematical operators.

Relation	Type	Description
ValueSource	ValueNode	The node from which the value for this operand should be taken.

2.3.4.1.15 avm.Value

A **Value** container characterizes the nature of a value but not its exact expression. It may have one avm.ValueExpressionType object as a child.

Attribute	Type	Description
Unit	String	The unit of the Value, expressed as the "symbol" specified by the QUDT unit library.
DataType	DataTypeEnum	Indicates the computing data type of the value.
DimensionType	DimensionTypeEnum	Indicates the dimension type of the value.
Dimensions	String	If DimensionType is non-scalar, indicates the dimensions of the data structure.

```
<Value
  DataType="Real"
  DimensionType="Scalar"
  Dimensions="1"
  ID="nv.case_thermal_conductivity"
  Unit="W/(K*m)">
...
</Value>
```

2.3.4.1.16 avm.ValueExpressionType

Abstract baseclass covering many types of expressions of the value.

2.3.4.1.17 avm.FixedValue

With **FixedValue**, the value expression is captured as a string literal. Vectors and Matrices should be expressed using Python syntax.

Attribute	Type	Description
Value	String	A literal capturing the expressed value

```
<ValueExpression xsi:type="ns1:FixedValue">
   <Value>42.0</Value>
</ValueExpression>
```

2.3.4.1.18 avm.CalculatedValue

With **CalculatedValue**, the value expression is determined by executing a calculation or procedural code fragment (in Python). These expressions may use other available Value objects. The syntax for these expressions is not yet defined.

Attribute	Type	Description
Expression	String	(required) A declarative or procedural expression describing how the value should be determined
Type	CalculationTypeEnum	(required) Defines the type of the expression

2.3.4.1.19 avm.DerivedValue

With **DerivedValue**, the value expression can be described as the value of another value container.

Relation	Type	Description
ValueSource	ValueNode	(required) The value container from which the value for this should be taken

```
<ValueExpression
  ValueSource="nv.temp_inlet_nom_coolant"
  xsi:type="ns1:DerivedValue"/>
```

2.3.4.1.20 avm.ParametricValue

With **ParametricValue**, the value expression may be selected by a user of the component. For instance, a Component model may represent a driveshaft design with a parametrizable length, bounded by the limits of manufacturability. In this case, a user would select a driveshaft length when instantiating the component into their design.

Note that the attributes of this Value are themselves described with ValueExpressionType objects, making them flexible.

Relation	Type	Description
Default	ValueExpressionType	(required) The default value for this field. Used if no AssignedValue is provided.
AssignedValue	ValueExpressionType	The value that the user has assigned (overriding the default)
Minimum	ValueExpressionType	The minimum selectable value (if value must be a scalar number)
Maximum	ValueExpressionType	The maximum selectable value (if value must be a scalar number))

2.3.4.1.21 avm.ProbabilisticValue

(Abstract) The value is described by a distribution.

2.3.4.1.22 avm.NormalDistribution

The value is described by a normal distribution. Note that the parameters of the distribution are themselves described with ValueExpressionType objects, making them flexible.

Relation	Type	Description
Mean	ValueExpressionType	(required) The mean of the distribution
StandardDeviation	ValueExpressionType	(required) The standard deviation of the distribution

2.3.4.1.23 avm.UniformDistribution

The value is described by a uniform distribution.

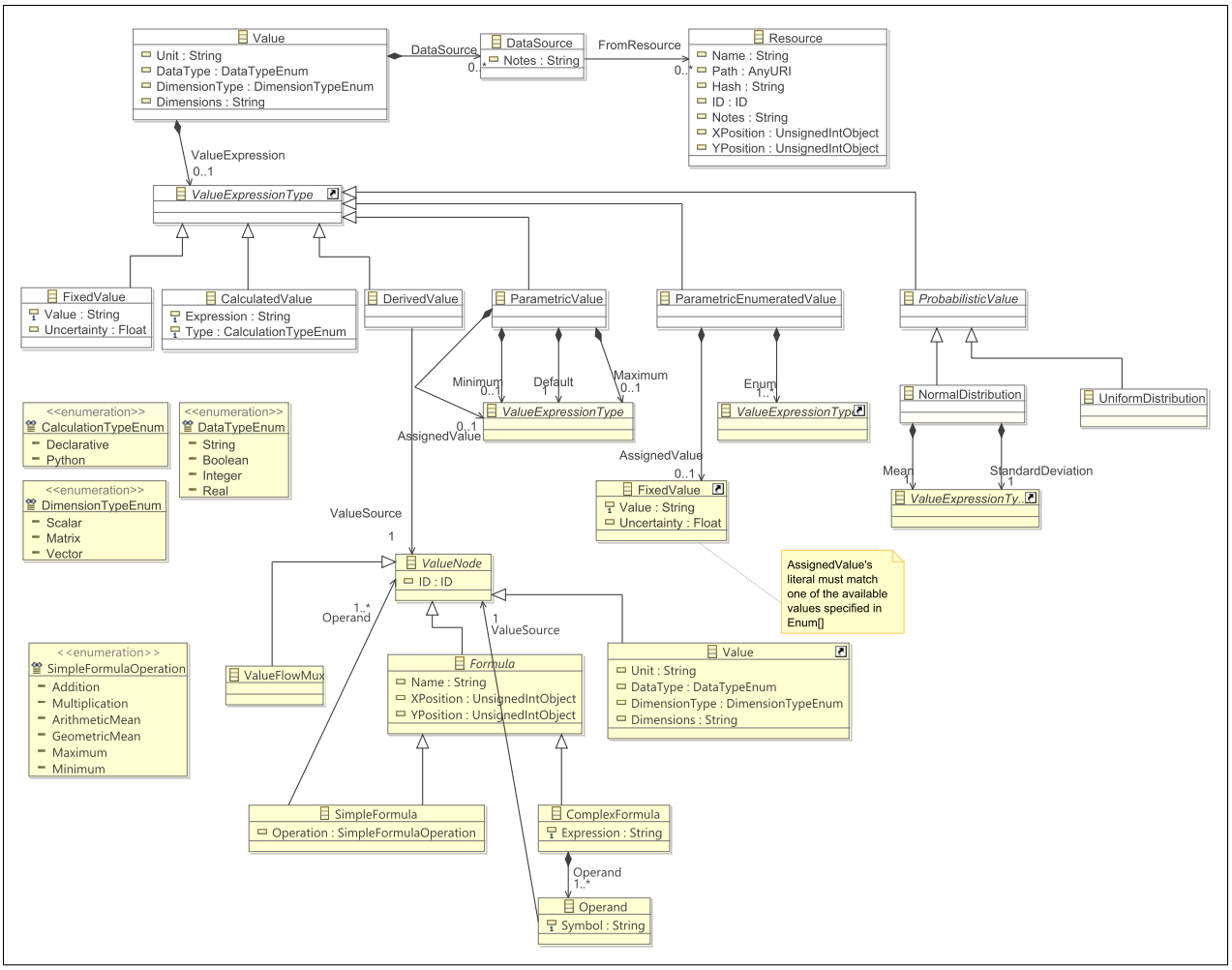

Figure 2.8: avm Namespace: Value diagram

2.3.4.1.24 avm.Connector

A Connector is an aggregation of DomainModelPort concepts that are intended to be composed at the same time. In a design context, the Connector is composed with another components's Connector, and the two Connectors must share a common definition. See 2.3.2.4 for further discussion on Connector semantics.

Attribute	Type	Description
Definition	AnyURI	If this connector is strongly typed and instantiated from a definition database, this is its definition URI
Name	String	The name of this connector instance
Notes	String	Any notes describing the connector
XPosition	Unsigned Int	The X Position of the element, if rendered graphically within its parent model
YPosition	Unsigned Int	The Y Position of the element, if rendered graphically within its parent model

Relation	Type	Description
Role	Port	Ports that are members of the Connector. These Ports must be mapped to DomainModelPorts contained within DomainModels of the Component.

```
<Connector
  ConnectorComposition="cml_zodb.cml_connectors_mechanical.
    CML_Connector_Mechanical_Shaft_Spline_Involute"
  ID="conn.eng_torque_out_to_transmission"
  Name="eng_torque_out_to_transmission">
  <Role Name="Z_Axis" PortMap="cad.EXT_TORQUE_OUT_axis_z"
    xsi:type="ns4:Axis"/>
  <Role Name="XY_Plane" PortMap="cad.EXT_TORQUE_OUT_plane_xy"
    xsi:type="ns4:Plane"/>
  <Role Name="YZ_Plane" PortMap="cad.EXT_TORQUE_OUT_plane_yz"
    xsi:type="ns4:Plane"/>
  <Role Class="Modelica.Mechanics.MultiBody.Interfaces.FlangeWithBearing"
    Name="brg_02" PortMap="mod.conn.brg_02" xsi:type="ns3:Connector"/>
</Connector>
```

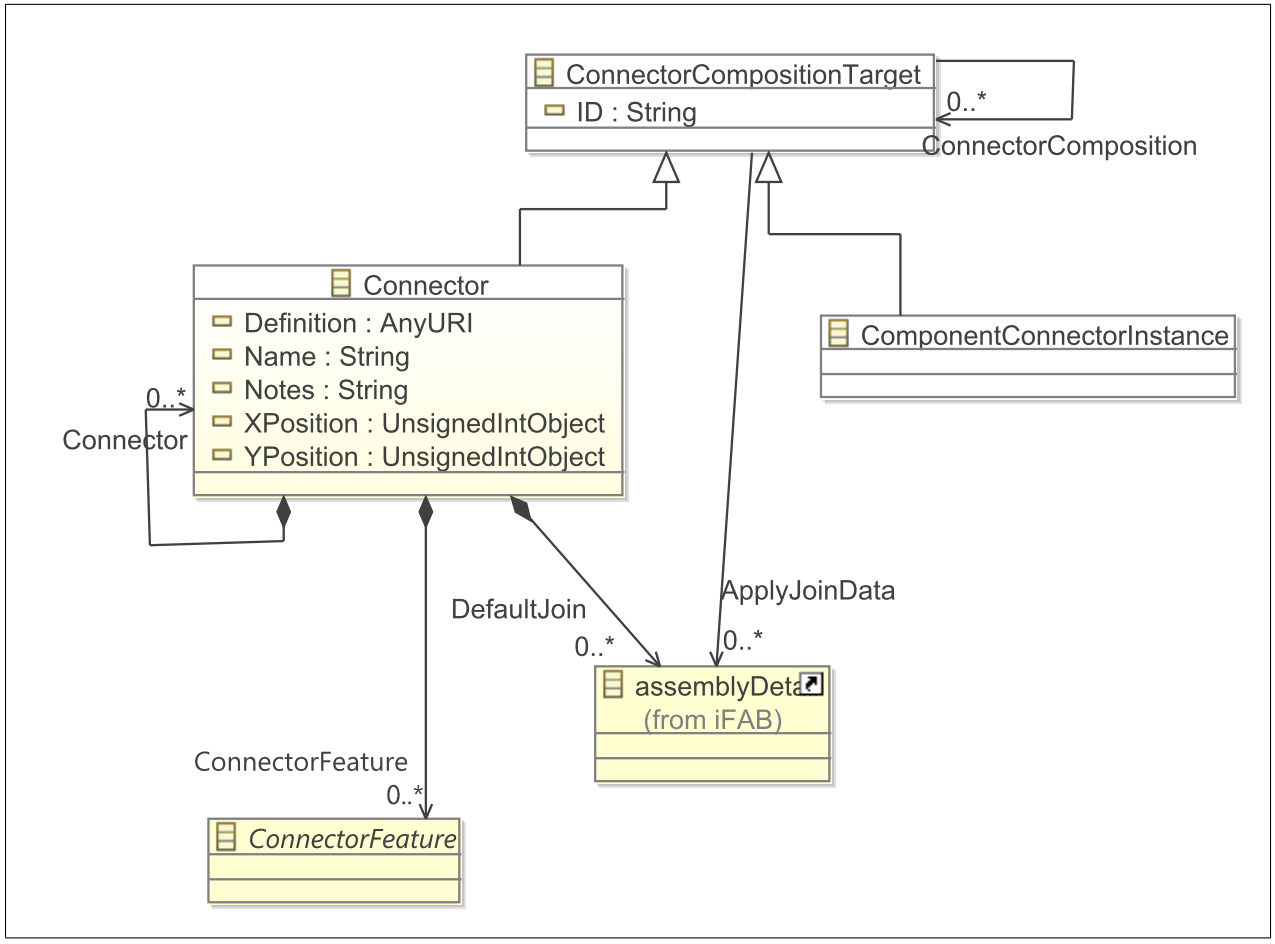

Figure 2.9: avm Namespace: Connector diagram

2.3.4.1.25 avm.DomainModel

(Abstract) Baseclass for descriptions of domain-specific models of the component.

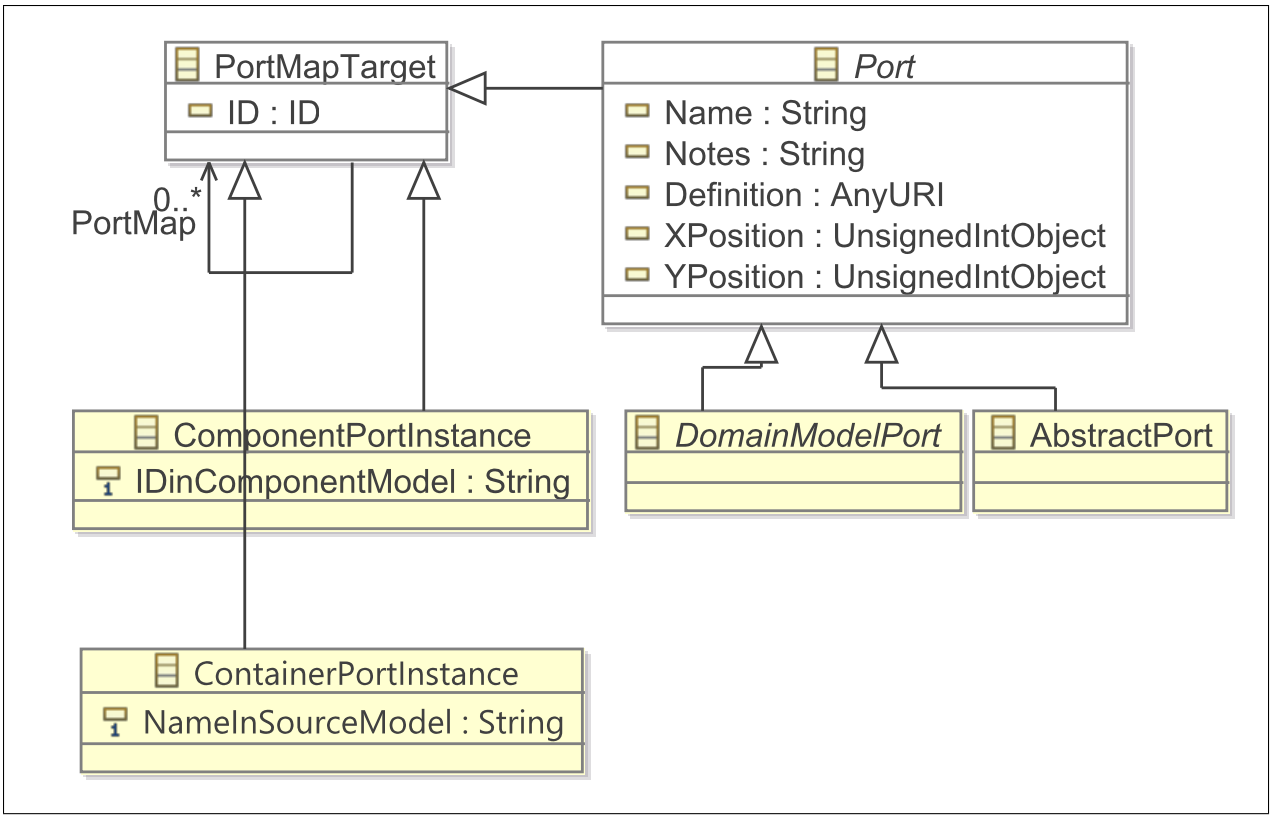

Figure 2.10: avm Namespace: Port diagram

Attribute	Type	Description
Author	String	The author of the model
Name	String	The name of the model
Notes	String	Any notes regarding the model
XPosition	Unsigned Int	The X Position of the element, if rendered graphically within its parent model
YPosition	Unsigned Int	The Y Position of the element, if rendered graphically within its parent model

Relation	Type	Description
UsesResource	Resource	Links the DomainModel description to the concrete artifacts implementing the model

2.3.4.1.26 avm.DomainModelPort

(Abstract) Baseclass for domain-specific ports contained within Domain Model objects. Since each domain has its own characteristic ports, this concept must be defined for each Domain Model type.

Attribute	Type	Description
Notes	String	Any notes regarding the model
XPosition	Unsigned Int	The X Position of the element, if rendered graphically within its parent model
YPosition	Unsigned Int	The Y Position of the element, if rendered graphically within its parent model

2.3.4.1.27 avm.DomainModelParameter

(Abstract) Baseclass for parameters of Domain Models. Since each domain may have its own rules about parameters, this concept must be defined for each Domain Model type.

Attribute	Type	Description
Notes	String	Any notes regarding the parameter

2.3.4.1.28 avm.DomainModelMetric

(Abstract) Baseclass for Metrics of Domain Models. A Metric is a value that can be extracted from the model after it has been used in an analysis. It may be a simulation variable that must be checked after a simulation, the moment-of-inertia of a physical object once it has been parametrically sized, etc.

Attribute	Type	Description
ID	ID	The ID of the metric, unique within the scope of the component model
Notes	String	Any notes regarding the metric
XPosition	Unsigned Int	The X Position of the element, if rendered graphically within its parent model
YPosition	Unsigned Int	The Y Position of the element, if rendered graphically within its parent model

Relation	Type	Description
Value	Value	A Value object capturing the last-calculated value of this metric

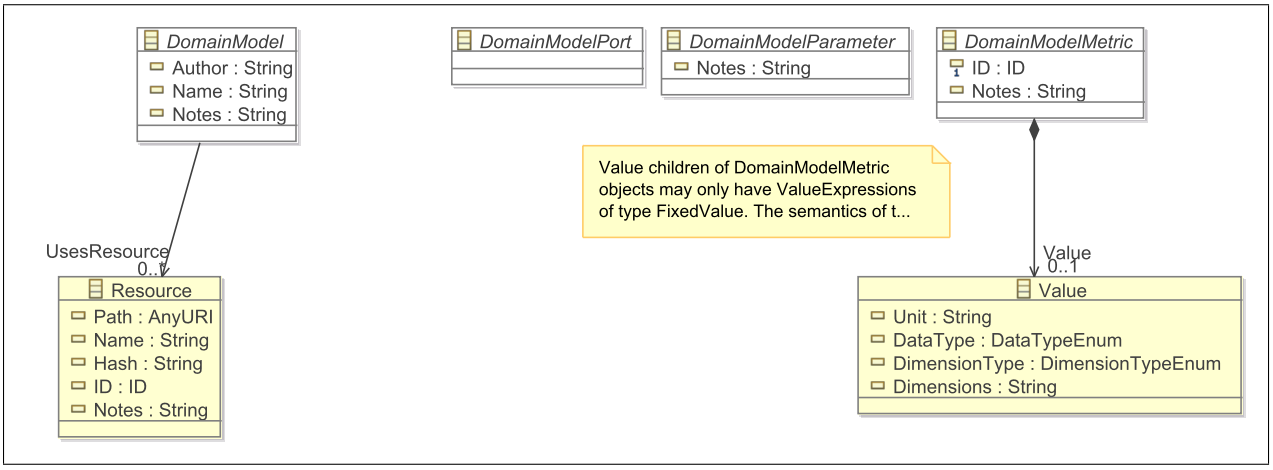

Figure 2.11: avm Namespace: DomainModel diagram

2.3.4.1.29 avm.Resource

Describes an artifact that must be included with the Component model, including models used for analysis, documentation, images, etc. Services that process Component models use these links to locate the artifacts with which it is associated.

Attribute	Type	Description
Name	AnyURI	A name for this linked Resource
Path	AnyURI	The path to the Resource. May either be an absolute path, or a path relative to the component ACM file itself.
Hash	String	MD5 hash of the artifact
Notes	String	Any notes describing the artifact
ID	ID	The ID of this resource, unique within the scope of the component model
XPosition	Unsigned Int	The X Position of the element, if rendered graphically within its parent model
YPosition	Unsigned Int	The Y Position of the element, if rendered graphically within its parent model

```
<ResourceDependency
  ID="cad_stp.path"
  Name="h170_cat_c9_600hp_prt.stp"
  Notes="STEP\H_Power_Package_Drivetrain\h170_cat_c9_600hp_prt.stp"
  Path="CAD\h170_cat_c9_600hp_prt.stp"/>
```

2.3.4.2 avm.modelica Namespace

2.3.4.2.1 avm.modelica.ModelicaModel

Subtype of avm.DomainModel
Describes a Modelica Model of the Component's behavior. Note that this does not *define* a ModelicaModel, it just describes the key features of a Modelica model that is defined in a separate .mo file.

Attribute	Type	Description
Class	String	(Required) The full path of the Modelica Model class

Relation	Type	Description
Connector	avm.modelica.Connector	A child Connector of the model
Parameter	avm.modelica.Parameter	A parameter of the model
Redeclare	avm.modelica.Redeclare	A redeclare statement of the model
Metric	avm.modelica.Metric	A metric of the model
Limit	avm.modelica.Limit	A limit of the model

```
<DomainModel
  Author="Ricardo"
  Class="C2M2L_Ext.C2M2L_Delivered_Component_Implementations.Prime_Movers.
         Reciprocating.Compression_Ignition.Engine_Basic.
         Example_Engine_Basic_Tstat_Degas"
  Notes="LanguageVersion:3.2 ModelingTool:Dymola ToolVersion:2013 (64-bit)
         FileFormat:mo"
  UsesResource="modelica.path"
  xsi:type="ns3:ModelicaModel">
  ...
</DomainModel>
```

2.3.4.2.2 avm.modelica.Parameter

A description of a parameter of a Modelica entity.

Attribute	Type	Description
Locator	String	(Required) The path to the parameter, relative to its parent entity

Relation	Type	Description
Value	Value	Describes the value of the parameter

```
<Parameter
  Locator="mass"
  Notes="cml_zodb.cml_basic_physicals.Mass">
  <Value ID="mod.Parameter.mass">
    <ValueExpression ValueSource="nv.common_info.mass"
       xsi:type="ns1:DerivedValue"/>
  </Value>
</Parameter>
```

2.3.4.2.3 avm.modelica.Metric

A metric of the Modelica Model.

Attribute	Type	Description
Locator	String	(required) The path to the varable to be checked, relative to the parent object.

```
<Metric ID="mod.metric.max_top_hose_temp.y"
  Locator="max_top_hose_temp.y"
  Notes="cml_zodb.cml_basic_physicals.Temperature"/>
```

2.3.4.2.4 avm.modelica.Limit

A limit imposed on a variable of the Modelica Model. The value of the variable (or signal) must be checked after a simulation has completed.

Attribute	Type	Description
Name	String	The name of the limit
VariableLocator	String	(required) The path to the varable to be checked, relative to the parent of the Limit object
BoundType	BoundTypeEnum	(required) Describes the type of bounded limitation imposed on the variable
ToleranceTimeWindow	Float	The maximum period of time for which it is acceptable for the variable to violate its bounds
Notes	String	Any notes describing the limit

Relation	Type	Description
TargetValue	Value	Describes the value against which the Limit's bounds must be tested

```
<Limit
  BoundType="MustNotExceed"
  Name="max_top_hose_temp"
  VariableLocator="mod.metric.max_top_hose_temp.y">
  <TargetValue>
    <ValueExpression ValueSource="nv.max_top_hose_temp"
       xsi:type="ns1:DerivedValue"/>
  </TargetValue>
</Limit>
```

2.3.4.2.5 avm.modelica.Connector

A description of a Modelica Connector. Note that this concept is distinct from the avm.Connector concept. Connector is a first-class concept within Modelica.

Attribute	Type	Description
Class	String	(Required) The full path of the Connector's class
Locator	String	The path to the instantiated Connector, relative to the ModelicaModel in which it is contained

Relation	Type	Description
Redeclare	avm.modelica.Redeclare	A redeclare statement of the Connector
Parameter	avm.modelica.Parameter	A parameter of the Connector

```
<Connector
  Class="Modelica.Fluid.Interfaces.FluidPort"
  ID="mod.conn.from_cac"
  Name="from_cac">
  <Redeclare Locator="Medium" Type="Package">
    <Value>
      <ValueExpression xsi:type="ns1:FixedValue">
        <Value>C2M2L_Ext.Media.Ideal_Gases.Simple_Air</Value>
      </ValueExpression>
    </Value>
  </Redeclare>
</Connector>
```

2.3.4.2.6 Class Diagrams

2.3.4.3 avm.cad Namespace

2.3.4.3.1 avm.cad.CADModel

Subtype of avm.DomainModel
Describes a CAD Model of the Component. Note that this does not *define* a CAD Model, it just describes the key features of a CAD model that is defined in a separate file.

Relation	Type	Description
Datum	avm.cad.Datum	Describes a Datum in the CAD Model.
Parameter	avm.cad.Parameter	Describes a Parameter of the CAD Model.
ModelMetric	avm.cad.Metric	Describes a Metric in the CAD Model.

```
<DomainModel UsesResource="cad.path" xsi:type="ns4:CADModel">
```

2.3.4.3.2 avm.cad.Parameter

Represents a parameter of the CAD Model file.

Attribute	Type	Description
Name	String	(required) The name of the parameter in the CAD Model file.

Relation	Type	Description
Value	Value	Describes the value of the parameter.

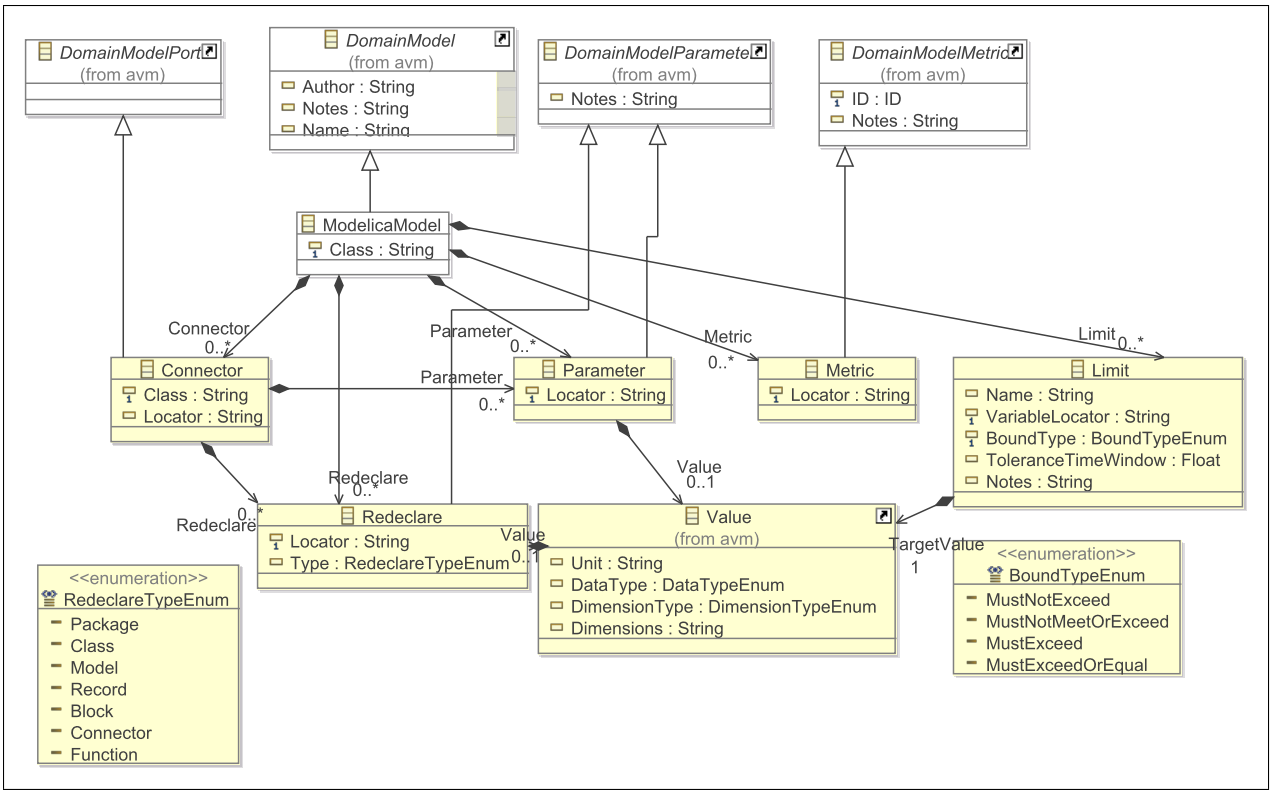

Figure 2.12: avm.modelica Namespace

2.3.4.3.3 avm.cad.Metric

A Metric is a property of the CAD Model or Datum that should be retrieved and set once the model has been used in a system composition. Common applications include checking the moment of inertia or mass, as calculated by the CAD tool, of a parametrically-variable model, where these values are not known until the parameter values are applied.

Attribute	Type	Description
Name	String	(required) The name of the property in the CAD Model file.

```
<ModelMetric ID="cad.TOTAL_MASS" Name="TOTAL_MASS">
   <Value DataType="Integer" DimensionType="Scalar" Dimensions="1"
     ID="cad.value.TOTAL_MASS" Unit="kg">
     <ValueExpression xsi:type="ns1:FixedValue">
       <Value>898</Value>
     </ValueExpression>
   </Value>
 </ModelMetric>
```

2.3.4.3.4 avm.cad.Datum

(Abstract) Describes a Datum that can be found in the CAD Model. Note that this does not *define* a Datum, it just describes one that can be found in the CAD Model.

Attribute	Type	Description
DatumName	String	The name of the datum in the CAD Model file.

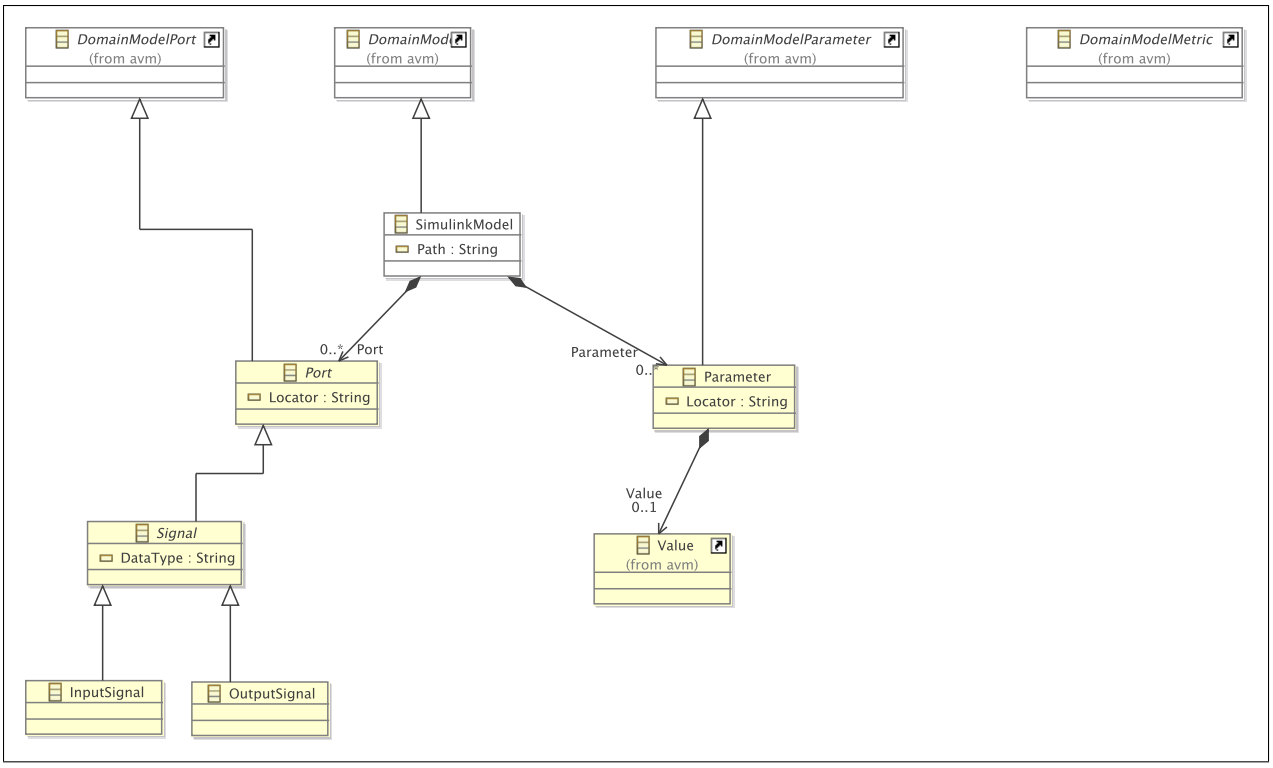

Figure 2.13: avm.simulink Namespace

Relation	Type	Description
DatumMetric	avm.cad.Metric	Describes a Metric of the Datum.

2.3.4.3.5 avm.cad.Point

Subtype of avm.cad.Datum
Represents a Point datum within the CAD Model file.

2.3.4.3.6 avm.cad.Axis

Subtype of avm.cad.Datum
Represents an Axis datum within the CAD Model file.

```
<Datum DatumName="EXT_MOUNT_1_axis_z"
  ID="cad.EXT_MOUNT_1_axis_z"
  xsi:type="ns4:Axis"/>
```

2.3.4.3.7 avm.cad.Plane

Subtype of avm.cad.Datum
Represents a Plane datum within the CAD Model file.

Relation	Type	Description
SurfaceReverseMap	avm.cad.Plane	Used for mapping a Plane datum to a corresponding Role in a Connector. An alternative to PortMap, this relation will reverse the Side A - Side B matching semantics from the usual case.

```
<Datum DatumName="EXT_MOUNT_1_plane_xy"
  ID="cad.EXT_MOUNT_1_plane_xy"
  xsi:type="ns4:Plane"/>
```

2.3.4.3.8 avm.cad.CoordinateSystem

Subtype of avm.cad.Datum
Represents a Coordinate System datum within the CAD Model file.

2.3.4.3.9 avm.cad.GuideDatum

Subtype of avm.ConnectorFeature
A **Guide Datum** is a marker that modifies the meaning of a datum. Whereas typically Datums from composed Connectors are constrained permanently in a CAD model, Guide Datums are constrained only while the model is being constructed, and are then unconstrained once the model is complete. If a composition is underconstrained, as in a joint, the Guide Datums may be used to set an "initial position" for the two parts.

Relation	Type	Description
Datum	avm.cad.Datum	Maps to the ID of the datum that should be considered a Guide Datum.

```
<Datum DatumName="EXT_TORQUE_OUT"
  Definition="CML_Mapping\cml_zodb.cml_extended_physicals.Coord_System_Definition"
  ID="cad.EXT_TORQUE_OUT"
  xsi:type="ns4:CoordinateSystem">
  <Metric ID="cad.EXT_TORQUE_OUT.position" Name="position">
    <Value DimensionType="Vector"
      ID="cad.EXT_TORQUE_OUT.position.vector" Unit="mm">
      <ValueExpression xsi:type="ns1:FixedValue">
        <Value>{0.0,2.35408792633e-18,-0.06825}</Value>
      </ValueExpression>
    </Value>
  </Metric>
  <Metric ID="cad.EXT_TORQUE_OUT.local_x" Name="local_x">
    <Value DimensionType="Vector"
      ID="cad.EXT_TORQUE_OUT.local_x.vector" Unit="mm">
      <ValueExpression xsi:type="ns1:FixedValue">
        <Value>{1.0,0.0,0.0}</Value>
      </ValueExpression>
    </Value>
  </Metric>
  <Metric ID="cad.EXT_TORQUE_OUT.local_y" Name="local_y">
    <Value DimensionType="Vector"
      ID="cad.EXT_TORQUE_OUT.local_y.vector" Unit="mm">
      <ValueExpression xsi:type="ns1:FixedValue">
        <Value>{0.0,1.0,0.0}</Value>
      </ValueExpression>
    </Value>
```

```
    </Metric>
  </Datum>
```

2.3.4.3.10 Class Diagrams

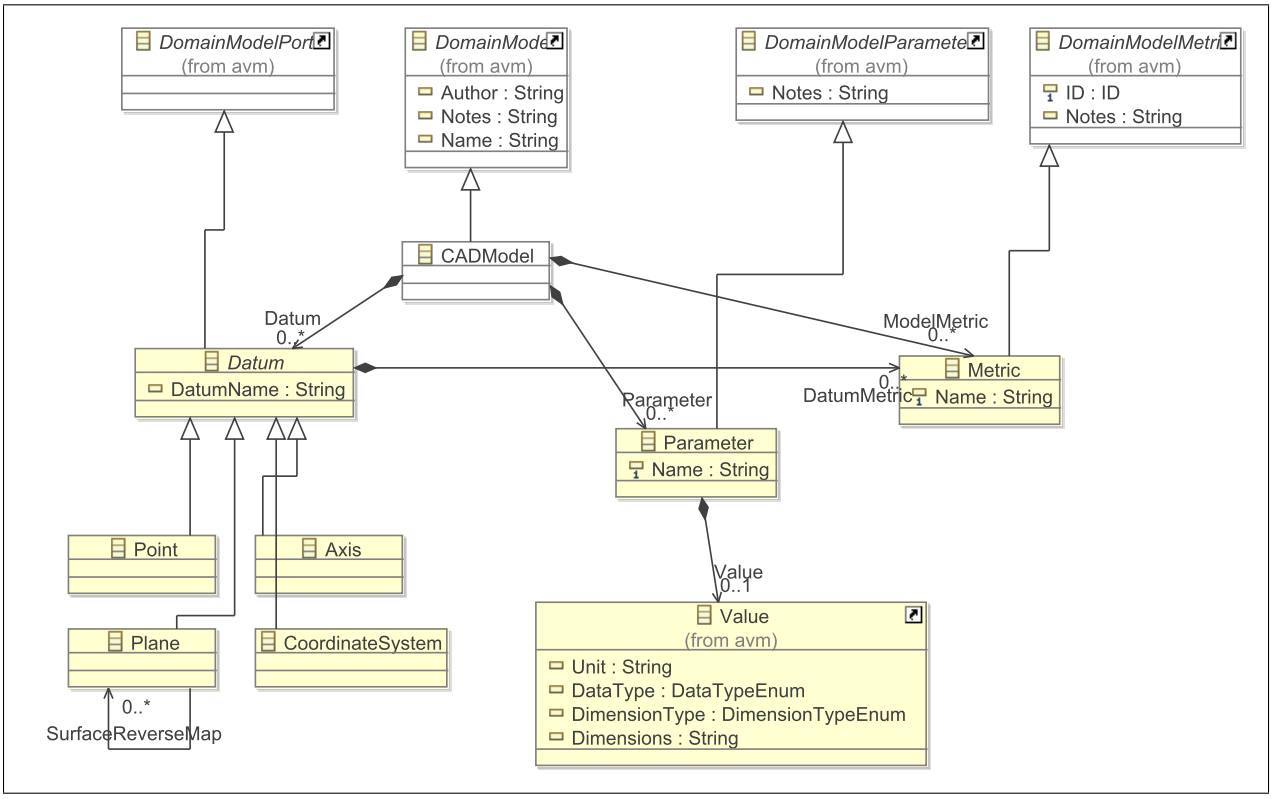

Figure 2.14: avm.cad Namespace: CADModel diagram

2.3.4.4 avm.manufacturing Namespace

2.3.4.4.1 avm.manufacturing.ManufacturingModel

Subtype of avm.DomainModel
Describes a Manufacturing Model of the Component. Note that this does not *define* a Manufacturing Model, it just describes the key features of a Manufacturing model that is defined in a separate file.

Relation	Type	Description
Parameter	avm.manufacturing.Parameter	Describes a parameter of the Manufacturing model.
Metric	avm.manufacturing.Metric	Describes a metric of the Manufacturing model.

```
<DomainModel UsesResource="manuf.path"
  xsi:type="ns2:ManufacturingModel">
```

2.3.4.4.2 avm.manufacturing.Parameter

Describes a parameter of the Manufacturing model.

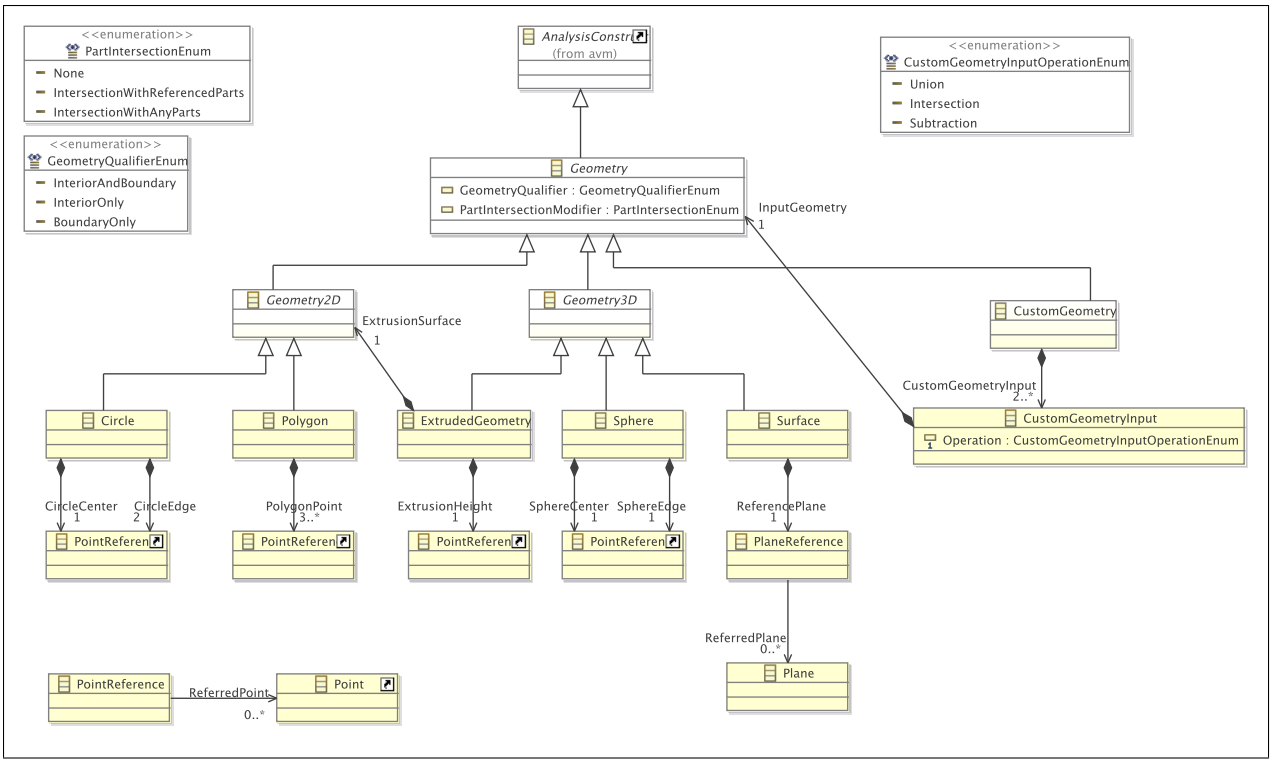

Figure 2.15: avm.cad Namespace: Geometry diagram

Attribute	Type	Description
Locator	String	The path to the corresponding field in the Manufacturing Model.
Name	String	A user-friendly name for this parameter.

Relation	Type	Description
Value	Value	Describes the value of the parameter.

```
<Parameter Name="procurement__make_or_buy">
  <Value DataType="String" DimensionType="Scalar" Dimensions="1"
    ID="procurement.make_or_buy">
    <ValueExpression xsi:type="ns1:FixedValue">
      <Value>Buy</Value>
    </ValueExpression>
  </Value>
</Parameter>
```

2.3.4.4.3 avm.manufacturing.Metric

Describes a Metric of the Manufacturing model. This is a value that must be calculated by an analysis tool, such as the per-unit cost for a fabricated component.

Attribute	Type	Description
Name	String	(required) The name of the property to be extracted.

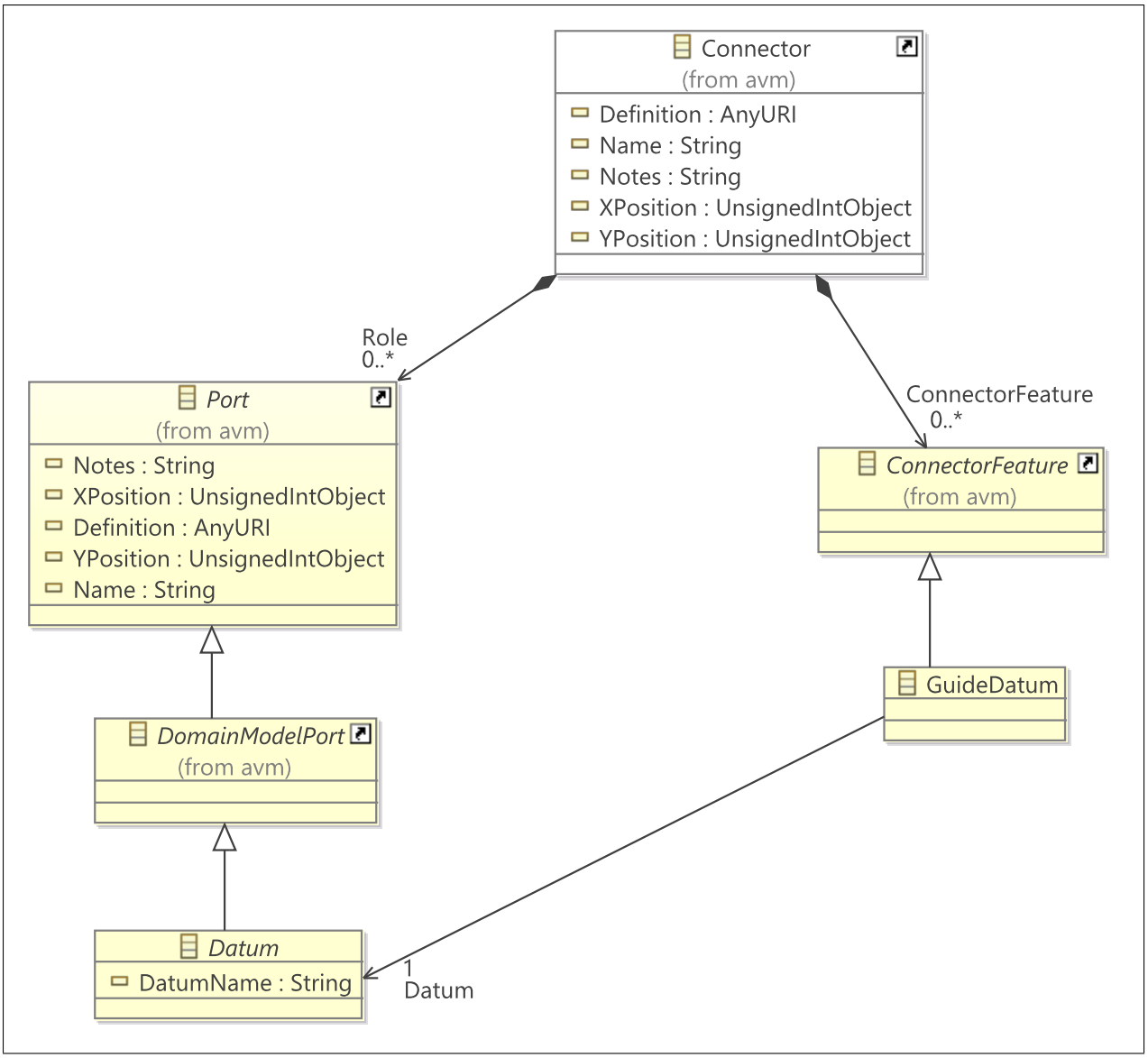

Figure 2.16: avm.cad Namespace: Guide Datum diagram

2.3.5 Restrictions

Although many restrictions on model structure are imposed by the XML Schema (XSD), not all are currently checked. This section lists some of the additional restrictions.

2.3.5.1 Property Names

Each Property must have a unique name within its level of hierarchy.

Property names must conform to the limitations of Modelica names, and not include the following characters (among others):

```
. / \
```

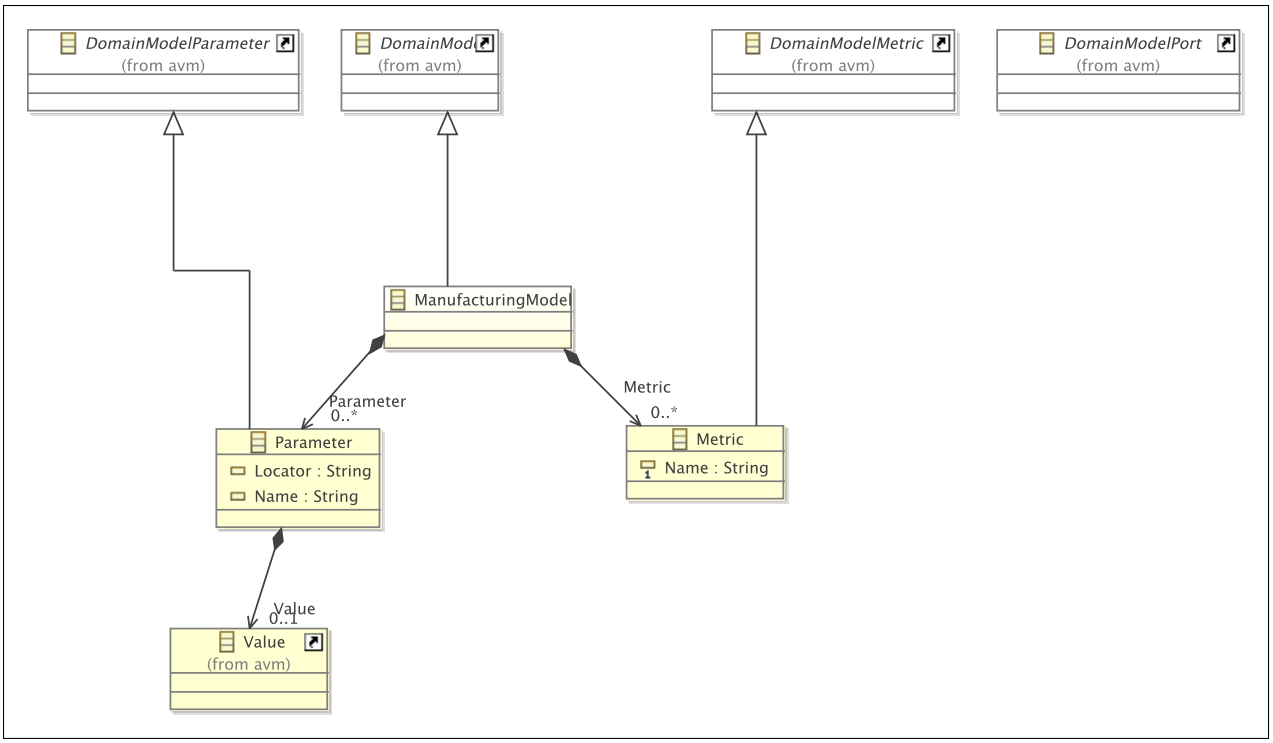

Figure 2.17: avm.manufacturing Namespace

2.3.5.2 Property Values

All Property objects must have a Value child.

2.3.6 ComplexFormula Expression Syntax

ComplexFormula objects contain *Expression*s which allow mathematical formulas to be defined. These expressions must use the syntax defined by the **MuParser** math library. The resulting value of such an *Expression* is considered to be the value of the **CustomFormula** node, so no assignment operator is required. Note that only a subset of MuParser syntax is supported. Supported functions and operators are listed here. The original documentation for MuParser's syntax can be found here: `http://muparser.beltoforion.de/mup_features.html#idDef2`

In **ComplexFormula** *Expression*s, model-defined **operands** can be referenced using their *Symbol*s.

2.3.6.1 Functions

Function
sin(x)
cos(x)
tan(x)
asin(x)
acos(x)
atan(x)
sinh(x)
cosh(x)
tanh(x)
asinh(x)
acosh(x)
atanh(x)
log2(x)
log10(x)
log(x)
ln(x)
exp(x)
sqrt(x)
sign(x)
rint(x)
abs(x)
min(x1, x2, x3, ...)
max(x1, x2, x3, ...)
sum(x1, x2, x3, ...)
avg(x1, x2, x3, ...)

2.3.6.2 Operators

Operators with higher priority will be evaluated first.

Operator	Meaning	Priority
&&	logical and	1
——	logical or	2
<=	less or equal	4
>=	greater or equal	4
!=	not equal	4
==	equal	4
>	greater than	4
<	less than	4
+	addition	5
-	subtraction	5
*	multiplication	6
/	division	6
^	raise x to the power of y	7

2.4 CAD Domain Model Specification

2.4.1 Overview

Computer Aided Design (CAD) models are needed in the Adaptive Vehicle Make (AVM) program to support 3D geometry composition and design analysis including structural, thermal, fluid, blast, and ballistic. Reasoning about the geometry of an AVM component requires a CAD model and additional information contained in the AVM Component Model (ACM). This appendix describes the conventions and guidelines for these CAD models as well as the related information in the AVM Component Model.

2.4.2 CAD Files

2.4.2.1 File Format

The CAD format supported by the META tools is PTC Inc. Creo® 2.0 M070 or above.

2.4.2.2 Units

All CAD files must have units specified in millimeters, kilograms, and seconds (mmKs).

2.4.2.3 View representations

In addition to the default representations, each Creo® model shall contain the following view representations:

View Representation	Description
Featured_Rep	Contains all of the geometry
Defeatured_Rep	Contains the basic shape without details such as fillets, chamfers, smaller holes

The Creo® model tree defines the sequential order in which features are created. The model tree starts with a primitive extrusion that contains no detail features such as holes, fillets, and chamfers. The detail features are added to the primitive feature such that the detail features are dependent only on the first primitive extrusion. Additionally, detail features do not depend upon other detail features. For example, a hole is positioned relative to the first primitive extrusion and not relative to a fillet. This approach accommodates building a simplified representation (view feature in Creo®) by simply removing the detail features from the model tree. Off-the-shelf part views are created in a similar fashion, however, for geometry imported rather than created natively in Creo®, the defeatured representation may be created through a shrink-wrap or other geometry simplification technique.

2.4.2.3.1 Properties

All .prt files have the material specified which is represented with a single material. Composite materials such as carbon fiber-reinforced resin and braid-reinforced rubber are considered a single material. Creo® has two approaches to specifying material properties. One approach is to specify the material (e.g. 6060-T6) and use the material properties along with the model geometry to compute the mass properties. If the model geometry accurately represents the part, then this approach is used. The other approach is to explicitly enter the mass properties into the CAD model and thus not compute the mass properties based on geometry. This approach is used when the model is incomplete such as when the geometry uses surfaces instead of solids, when the geometry does not accurately represent the part, and when the material properties are approximated. Composites (and other inseparable assemblies) with imbibed components and dissimilar materials such as metallic frame and backing plate, inserts, stand offs, and ceramic tiles are best managed as an assembly.

For the case where the part geometry accurately represents the part, the model is set to use Creo®'s "Geometry and Density" and the model material is specified. The material specification in the CAD model at a minimum contains the density which corresponds to the density found in the META Material Library.

For the case where the mass properties are explicitly entered, the model is set to compute mass properties via Creo®'s "Geometry and Parameters" or "Fully Assigned". Additionally, the following model parameters are set:

Description	**Creo® Parameter**	**Units**
Density	MP_DENSITY	kg/mm^3
Mass	PRO_MP_ALT_MASS	kg
Volume	PRO_MP_ALT_VOLUME	mm^3
Center of gravity, x-component	PRO_MP_ALT_COGX	mm
Center of gravity, y-component	PRO_MP_ALT_COGY	mm
Center of gravity, z-component	PRO_MP_ALT_COGZ	mm
Moment of inertia, xx-component	PRO_MP_ALT_IXX	$kg\ mm^2$
Moment of inertia, yy-component	PRO_MP_ALT_IYY	$kg\ mm^2$
Moment of inertia, zz-component	PRO_MP_ALT_IZZ	$kg\ mm^2$
Moment of inertia, xz-component	PRO_MP_ALT_IXZ	$kg\ mm^2$
Moment of inertia, yz-component	PRO_MP_ALT_IYZ	$kg\ mm^2$
Moment of inertia, xy-component	PRO_MP_ALT_IXY	$kg\ mm^2$

Material properties (e.g. density, yield strength, ultimate strength, etc.) are stored in the CAD file for user reference however the values are retrieved from the META Material Library when needed by a tool for analysis. The material name specified in the Creo® part file must be identical to the material name in the META Material Library. All .prt files shall have the material specified.

2.4.2.3.2 Datums

Datums are used to support composition of components within a 3D space and also support the application of boundary conditions on the component. Creo® default datums shall not be used as geometric datums to define interfaces or for geometric dimensioning and tolerancing.

2.4.2.3.2.1 Planes

Side A of datum planes point away from the part surface associated with the datum plane. Interface coordinate systems shall have the positive Z axes shall point away from the part surface associated with side A of the datum plane

2.4.2.3.2.2 Axes

An example of an axis is the centerline of a cylinder.

2.4.2.3.2.3 Points

In addition to supporting component composition, datum points are used to define polygons, circles, spheres, cylinders, and extrusions for identifying the geometric placement of loads and constraints. Datum points are consistent across parts within a component class.

2.4.2.3.2.4 Coordinate Systems

One or more coordinate systems in each component may be used to position the component in the assembly before the necessary structural support components are present in the design. The location of the coordinate systems is based on functional factors of the component. For example, a hatch may have the coordinate system centered on the opening. All components in a class have one coordinate system for positioning and that coordinate system's name, location, and orientation is consistent for all components in that class.

2.4.2.3.3 Connectors

The AVM Component Model Specification contains details for connectors. For CAD, connectors represent the physical connection between parts and assemblies. Datums are used to identify the connector geometry.

2.4.2.3.4 Joins

Joins such as welds, bolts, and rivets are specified according to the Design Data Package.

2.4.2.3.4.1 Welds

Welds are represented by 3D geometry.

2.4.2.3.4.2 Bolts

Bolts are represented using bolt parts.

2.4.2.3.4.3 Rivets

Rivets are represented using rivet parts.

2.4.2.3.5 Part Notes

Part notes shall be specified in accordance with ANSI Y14.41. These notes provide information needed for manufacturing. All notes shall include the corresponding parameter note names. Notes must be in a true 3D model orientation. Notes shall not be placed Flat to Screen.

2.4.2.3.6 Default Tolerances

Default tolerances are specified in the notes. The default tolerances for a metric (mm) dimensioning system are:

Value	Tolerance
X.X	± 1.0
X.XX	± .750
X.XXX	± 0.250
ANG	±5°

If these default tolerances are not appropriate for a particular part, the default tolerance parameter must be changed.

2.4.2.3.7 Surface Treatments

Default surface treatments are specified in the notes.

2.4.2.3.8 CAD Model Maturity

CAD models must include model maturity which is specified via the AVM component model parameter MODEL_MATURITY with the following possible values:

Value	Description
Conceptual	Part file is conceptual and lacks detailed geometry needed to fabricate the part
Fully_Detailed_Geometry	Part file contains fully detailed geometry
Fully_Detailed_Model	Part file contains fully-detailed geometry, model notes, and complete ANSI Y14.41 annotations.

2.4.2.3.9 CAD Model Release State

CAD models must include release state specified which is specified via the AVM component model parameter MODEL_RELEASE_STATE with the following possible values:

Value	Description
In_Work	CAD model is in-work
Checking	CAD model is being checked prior to release
Release	CAD model is released

2.4.3 Content in the AVM Component Model

Additional information required for geometric analysis must be provided in the *wrapper* AVM component model for a given component. The AVM Component Model is detailed in the AVM Component Model Specification. Sections 2.4.6 and 2.4.7 contain details for the different types of CAD files.

2.4.3.1 Manufacturer Load Ratings

Load ratings are specified for off-the-shelf components and are used in structural analysis.

2.4.4 Feature Requirements and Guidelines

Geometric features must adhere to the following requirements and guidelines.

Feature	Notes
Internal threads	Hole shall be tap drill size to the maximum drilled depth with cosmetic feature for the thread at nominal diameter to the full thread length
External threads	Solid body on the part shall be nominal size with cosmetic feature for the thread at tap drill size
Internal gears and splines	Extruded feature shall be modeled to the inside diameter with the cosmetic feature at the outside (root) diameter
External gears and splines	Extruded feature shall be modeled to the outside diameter with the cosmetic feature at the root diameter
Counterbores and countersinks	Provide clearance for the head of a screw or mating part
Small features	Standard edge breaks and other small features should not be added to the model

The following additional guidelines are related to creating a model that is readily re-usable for make parts.

Feature	Notes
Edges	There should be no edges shorter than .005 inch (.127 mm).
Cylindrical surfaces	There should be no small cylindrical surfaces smaller than .05 R (1.27 mm).
Surface gaps	There should be no surface gaps or overlaps in the model.
Mirrored features	When required, mirrored features shall be created by mirroring in conjunction with the copy command. Straight mirroring creates features that cannot be easily redefined and does not show the proper dimensioning on a drawing.
Copied features	When copying features make sure only the dimensions which should be dependent on the original are copied.

2.4.5 Mesh Files

The META tools support pre-meshed components which can be used for various analyses such as structural finite element analysis. Components are typically meshed automatically as needed for a particular analysis. Pre-meshing components supports situations where automatic meshing yields a less-than-desirable mesh. Mesh files are stored in the same directory as the component's CAD files. Mesh file formats are dictated by each analysis tool.

2.4.6 Make parts

Make parts are those parts that are custom and would be built as part of fabricating the design. This is in contrast to off-the-shelf parts that are available for purchase. Part geometry for a make part is represented through detailed CAD features. One make part is represented by one CAD part file. Models shall be full scale to the dimensional mean of the design tolerance zone unless otherwise specified in the model notes. The mean of dimensional tolerance limits is the average of the upper and lower limits of the tolerance zone. This should not be confused with the nominal condition, which is the explicitly stated base value of a dimension to which tolerance is applied.

See the following table for examples.

Example dimension/tolerance	**Mean**	**Nominal**
1.125 ± 0.125	1.125	1.125
1.00 +0.25/-0.00	1.125	1.00
1.25 +0.00/-0.25	1.125	1.25
1.00 to 1.25	1.125	No meaning
1.0625 +.1875/-0.0625	1.125	1.0625

The following details model simplification and other exceptions.

Item	**Notes**
Standard stock materials	Stock materials shall be modeled at nominal.
Threads, gears, and splines	May be cosmetic features if full feature definition is provided in the notes. Their geometry shall be nominal. For example, torsion bar splines commonly have one tooth missing to define positioning during assembly, therefore a fully detailed model is required.
Components with repetitive detail	Springs, chains, perforated plate, expanded metal sheet, and other components with repetitive detail shall be modeled with simplified geometry, the minimum required to convey design intent.
Minimum and maximum radii	The radius shall be modeled to mean value.
Maximum and minimum angles	Maximum and minimum angles, be modeled to the recommended values applicable to objective process such as castings and forging draft angles.

2.4.7 Off-the-shelf parts

Off-the-shelf parts, or buy parts, are parts that can be purchased and don't have to be built as part of fabricating the design. Part geometry for an off-the-shelf part is represented through CAD features. Off-the-shelf part vendors often supply lower fidelity models to obfuscate details of their design. In this case, the model may be sufficient for a lower fidelity representation of the part. Vendors sometimes supply fully detailed models that are not suitable for purposes such as meshing and design rendering. In this case, a lower fidelity representation is warranted to facilitate rendering and analysis. While a structural analysis of these parts typically would not involve finite element analysis, meshing the part is needed to transfer

loads to other parts. Representing the multiple fidelities is achieved with one part file using the Creo® views feature. Typically, off-the-shelf parts CAD model maturity would be "Conceptual" and in some cases "Fully_Detailed_Geometry" but rarely "Fully_Detailed_Model". For off-the-shelf parts, the default CAD model release state would be "Release".

2.4.7.1 Component Geometry Captured with a CAD Part File that Represents an Assembly

In some cases, assemblies are better represented without detailing all parts that represent the assembly. For example, the engine can be sufficiently defined with exterior geometry and aggregate mass properties without adding individual parts such as pistons, manifolds, bearings, and bolts. In this case, one CAD part file represents an assembly. Make parts, or in this case make assemblies, are not represented by a CAD part file that represents an assembly because insufficient information exists to manufacture the assembly. Therefore, only off-the-shelf assemblies are represented by these part files. Properties identified in Section 2.4.2.3.1 are still specified for part files, however the material name corresponds to the most common material in the assembly. The part file must convey that the part represents an assembly via setting the AVM component model parameter name PART_REPRESENTS_AN_ASSEMBLY to "Yes". Additionally, the AVM component model parameter ASSEMBLY_MOST_COMMON_MATERIAL must be set to a material name resident in the META Material Library. Each CAD part file that represents an assembly must be set to compute mass properties via Creo®'s "Geometry and Parameters" or "Fully Specified". The density of the most common material is retrieved from the META Material Library when needed by a tool for analysis or an effective density may be used by dividing the specified mass by the specified volume.

2.4.7.2 Component Geometry Represented with a CAD Assembly

A CAD assembly consisting of multiple CAD part files can be used describe a buy component. Examples include suspension assemblies, damper assemblies, and brake calipers. Each part file must have the material specified using a material name resident in the META Material Library.

2.5 Dynamics Domain Model Specification

2.5.1 Overview

Dynamics Domain Models represent the physical behavior of components, and are implemented as Modelica models. Modelica is a non-proprietary, object-oriented, equation-based language to conveniently model complex physical systems, e.g., mechanical, electrical, electronic, hydraulic, thermal, control, or process-oriented subcomponents. https://modelica.org

2.5.2 Modelica Language Specification

Follow the Modelica Language Specification 3.2 and make sure your models comply with it. AVM Components support only a subset of the Modelica language. This subset encompasses only those features which are essential for composition of the individual models/components, and excludes some of the more nuanced and advanced aspects of the Modelica language.

2.5.3 File Format

Modelica models and packages are stored as files with a '.mo' extension. These are textual files and they can be edited using text editors (e.g., Notepad or Notepad++) or with Modelica-specific tools, such as OMEdit or Dymola. Modelica packages and subpackages can be saved as one single file or as a directory structure one directory per package and one file per model.

2.5.4 Modelica Standard Library and User Defined Libraries

Modelica.org provides a non-proprietary Modelica Standard Library (MSL) and each Modelica tool includes at least one version of the MSL. Users can develop their own custom libraries, which often extend and/or use MSL models. When creating a new package for use in conjunction with OpenMETA, use at least MSL version 3.2 or newer. While a library is under development, it is best to save each model as a separate file, which makes the task of maintaining and updating the models much easier. When the package is finalized and prepared for use as a reference in AVM Component Models, it is better to save the entire package, i.e. Modelica Library, as one 'package.mo' file within a directory named with the library name and its version, like **UserDefinedLibrary 1.2.45**, where **UserDefinedLibrary** is the name of the library, **1** is the major version number, **2** is the minor version number, **45** is a revision number (like Subversion revision number).

2.5.4.1 Referencing a Modelica model

The AVM Component model defines a Class attribute, which is the fully qualified unique path of the Modelica model. For instance: Modelica.Electrical.Analog.Basic.Resistor is the Class for the MSL ideal resistor.

2.5.5 Modelica Tools

There are both open-source (OpenModelica, JModelica, etc.) and commercial tools (Dymola, SimulationX, MapleSim, SystemModeler, etc.) which are available to build/edit/simulate Modelica models. No matter which tool is utilized to develop user-defined Libraries, users must check if their library can be loaded with OpenModelica 1.9.0 RC1+ in order to ensure compatibility with the OpenMETA toolset. For example, the Component Authoring Tool uses the Open Modelica Compiler (OMC) to extract interface information from the Modelica model for use in the corresponding AVM Component.

2.5.6 Naming restrictions

Follow all Modelica naming conventions and restrictions based on the Modelica Language Specification. If you use a Modelica tool to edit the models, the tool will ensure that models conform to the rules. When a system model is being built using the OpenMETA tools, similar naming restrictions apply:

-do *not* use top-level Modelica library names: Modelica, OpenModelica, UserDefinedLibraryName

-do *not* use Modelica keywords: input, final, protected, extends, etc.

2.5.7 Parameter

Public parameters are externally visible parameters of the model. If the parameters are required to be changed for different instances or by the user they should be marked as public. If the parameters are fixed for the component class they should be marked as protected, and will not be visible inside the AVM Component Model.

2.5.7.1 Real

Real-valued parameters are supported as AVM Component Parameters/Properties. They are also supported at the Modelica model wrapper level without any type-checking feature.

2.5.7.2 Boolean and Integer

Boolean- and Integer-valued parameters are **not** supported as AVM Component Parameters/Properties. However, they **are** supported at the Modelica model wrapper level without any type-checking feature.

2.5.7.3 Units

Units are currently **not** supported. If the Property/Parameter has a unit specified it will not cause any problems, but the unit type will be overwritten when the model is instantiated.

2.5.7.4 Tables and Vectors

Tables and Vectors are **not** supported as AVM Component Parameters/Properties. They **are** supported at the Modelica model wrapper level without any type-checking feature. If a table needs to be parametrized from the AVM Component Parameters/Properties, use a Real parameter value to archive that.

2.5.7.5 Enumeration

Enumeration parameters are **not** supported as AVM Component Parameters/Properties. They **are** supported at the Modelica model wrapper level without any type-checking feature.

2.5.7.6 Replaceable

Replaceable models, packages, functions, records and blocks are **not** supported as AVM Component Parameters/Properties. They **are** supported at the Modelica model wrapper level and inside ModelicaConnectors, called ModelicaRedeclare, without any type-checking feature.

2.5.8 Connector

Connectors in Modelica represent interfaces between models. The composition between Modelica models is done through connectors using connect statements. Connectors can represent physical and logical (signal) connections. Physical connectors transport power and energy between models and signal connectors share variables and signals among models.

2.5.8.1 Connectors with parameters

Connectors can have parameters such as the FlangeWithBearing connector from the MSL MultiBody library. Some connectors can be parametrized, but it is better to keep the parameters fixed inside the connector and not refer to model-level parameters or other elements inside the model. If there is an unavoidable need to parametrize connectors, there are two options to accomplish this task:

Option 1: use fixed port parameters - preferred way, limited possibility of problems
Option 2: through environments -could cause problems
Option 3: through model parameters - could cause problems

2.5.8.2 Connectors with replaceable elements

There are connectors which have replaceable elements, like the fluid connectors. A good way to handle these cases is to create your own connectors, each with a fixed fluid type, and instantiate that type of connector. This will ensure that the user will connect only compatible ports together even in native Modelica tools. The various approaches to this situation are described below:

Option 1: utilize user-defined connectors (fluid type cannot be changed) - preferred way

Option 2: create your own fluid types like Oil1, Oil2, Oil3 and restrict your connector to the base class Oil. - elevated potential for problems

Option 3: use the fluid ports from the MSL - most likely will cause probleam.

2.5.8.3 Floating connectors

When a Modelica model is set up for simulation and it uses multiple Modelica models, i.e. a test for the models, it a good practice to avoid connectors at the top level which do not belong to any subsystem. Some Modelica Tools will initialize the variables/unknowns in the Modelica connectors if they are not specifically defined. It is better to create a model which has such a connector as an output and define the initial values of the variables inside that model.

2.5.8.4 Bus connectors and breakouts

2.5.9 Multi-fidelity

Each AVM Component can have multiple Dynamics Domain Model representations. This concept is used to link a single component to different models with different levels of abstraction, i.e. low fidelity, medium fidelity, and high fidelity models, which can provide increasing levels of simulation accuracy at the cost of longer simulation times. The connector interfaces must be exactly the same across all representations. The number of parameters can vary, but all parameters must be derived from a common set of parameters.

2.5.10 Simple and Custom Formula

Simple and Custom Formula can be used to capture relationships between AVM Component-level parameters. Simple Formula elements are translated into Modelica code, but Custom Formula elements are not; in the Custom Formula case, sometimes the parameter propagation relationship is lost in the translated model. If you would like to preserve the Custom Formula elements, they need to be implemented directly in the Modelica model.

2.5.11 Composition

Composition of Modelica models is done through connectors and parameters, which are interfaces of the AVM Component model. Composition between connectors represents interaction between components, whereas parameter connections ensure correct parametrization of components like engine and engine control unit (ECU). In this example, the ECU parameters depend on engine's parameters.

2.5.12 Equations and algorithms

Equations, initial equations, and algorithms can be defined inside the Modelica model, but that content has no representation in the AVM Component model.

2.6 Cyber Domain Model Specification

2.6.1 Overview

Cyber designs represent the details of controller logic and the (in the future) software and hardware used to implement those details. Cyber controller design models are usually created using Simulink, and then imported into the CyberComposition modeling language. Controller designs are simulated as part of Modelica dynamics models. In CyPhyML the controller interface is adapted to be able to execute the generated controller code within the Modelica dynamics simulation. This chapter describes the important details regarding the integration and simulation of controller components in the AVM modeling framework.

2.6.2 Cyber Model Files

The basic process flow for controllers designed in Simulink is as follows: A control designer imports particular discrete-time subsystems from an existing Simulink/Stateflow model to create new components in the CyberComposition language. The port data types, rates, and execution priorities are determined by Simulink and recorded in the CyberComposition model during the import process. The user then defines the interface mapping required to compose the controller with the rest of the Modelica model. Generated C code is created using function block templates, and then integrated into the overall Modelica simulation by compiling the controller code into a shared library which will be called by the Modelica external function interface during simulation.

2.6.2.1 Simulink

Simulink models may be specified in the MDL model format, but the new compressed XML model file format (.slx) has not been tested. We have used our tools with Matlab versions R2012A and R2013A. Any Simulink model can be imported, possibly including Stateflow charts. However, our code generation tools are limited to a specific set of Simulink blocks. The Stateflow and Embedded Matlab conversions also have limitations. These limitations only affect our ability to generate code.

2.6.2.2 CyberComposition

The GME cyber composition language captures the controller details, and defines the interface between the Simulink model and Modelica. Interface ports and parameters are typed in both Simulink and Modelica, and must be compatible. CyberComposition also contains another simple controller specification language called SignalFlow, which is more suited to manual model creation in the AVM tools environment.

2.6.2.3 Modelica

The Modelica interface to the cyber models is as defined in the Modelica Specification (v. 3.2) as described in the section on Dynamics. Each CyberComposition controller gets a wrapper that defines the required calls and data types for the Modelica External Function Interface (EFI).

2.7 Manufacturing Domain Model Specification

The iFAB Foundry Data Specification is a collection of XML schemas that describe the information required to analyze, procure, and manufacture a design. The information can be divided into two groups, component information and assembly information. Textual descriptions of the information are provided in the sections below and the xsd schemas are provided as appendices.

2.7.1 Overview

2.7.1.1 Manufacturing Related Component Information

In general, there are 3 types of components: Commercial Off the Shelf (COTS), Make to Order (MTO), and custom manufactured. Each of these component types require specific information that is used to analyze, source, procure, and manufacture. Each component type will be defined and a schema that can be used to fully define the component is provided in the following sections.

2.7.2 Core Concepts

2.7.2.1 Commercial Off the Shelf

Commercial Off the Shelf (COTS) components refer to components that can be purchased from a vendor, or a set of vendors, as a fully defined component with published/quoted purchasing, cost, lead time, and qualification information. An example of a COTS component would be an engine in a catalogue with known contact information for the manufacturer or supplier.

Included in the COTS component type are components that are already owned by the designer/government that are labeled as Government Off the Shelf (GOTS). Also known as Government Furnished Equipment (GFE), these components are similar to COTS components in that all of the purchasing information is known and can be readily provided to analyze, procure, and manufacture the component.

2.7.2.2 Make to Order

Make to order components are defined as components with an established manufacturing/vendor base that produce parameterized components. For example drive shafts with specified lengths and diameters. Where the drive shaft material and available lengths and diameters are given in a catalogue and can be purchased from the vendor or manufacturer.

Both COTS and MTO components are considered purchased components and follow the purchased component schema. There are several data elements required for purchased components. Commercial Off the Shelf and Make to Order Required Information

- Cost - for a specified order quantity
- Lead time - for a specified order quantity
- Packaging Pallet, box, crate, pieces
- Transportation Needs temperature controlled, hazardous material, or both
- Vendor/Manufacturer contact information CAGE code, catalogue number part description, shipping dimensions, shipping mass, shipped from address

2.7.3 Custom manufactured Components

Custom manufactured components are the components that are to be made in the iFAB Foundry distributed manufacturing environment and include things like brackets, sheet metal boxes, hoses, plate metal, etc. In order to properly analyze and construct a custom manufactured part, this type has been further subdivided into several classes. These part classes include machined, casting, forging, plate, pipe bar and tube, additive, polymer, and plastic.

Common Custom Manufactured Component Required Information

- Material Aluminum, alloy steel, carbon steel, stainless steel (Specific alloys are further specified)
- Inorganic Coating Black chrome, sulfuric acid anodized, hard coat anodized, manganese phosphate, black oxide, nickel, zinc phosphate
- Organic Coating (Painted) CARC, non-CARC
- General tolerance information

  Curved Surface Tolerances General (mm), Surface Roughness (um).

  Curved Wall Tolerances General (mm), Surface Roughness (um).

  Simple Hole Tolerances Diametrical (mm), Positional (mm), Surface Roughness (um)
- Holes threaded (yes/no)

Machined Part Class Required Information

- Covered by the common information

Plate/Sheet Part

- Common information

  Material, Coatings, Simple Hole Tolerances
- Tolerance information

  Planar Face Tolerances General (mm), Surface Roughness (um).

  Complex Hole Tolerances General (mm), Surface Roughness (um).

  Bend Angle Tolerance

Pipe/Bar/Tube Components

- Common information

  Material, Coatings, Simple Hole Tolerances
- Tolerance information

  Ends Tolerances General (mm), Surface Roughness (um).

  Complex Hole Tolerances General (mm), Surface Roughness (um).

  Bend Angle Tolerance

2.7.4 Manufacturing Related Assembly Information

Manufacturing assembly information describes how any two or more components are joined together in the context of an assembly. This information is used to analyze and manufacture a given design using the iFAB Foundry manufacturing capabilities.

Two components can be joined together in several different ways. This specification document describes a subset of the total number of joining mechanisms, which represent the most common mechanisms for military ground vehicle manufacturing. This subset also encompasses the joining mechanisms that can be automatically assessed using the iFAB Foundry tools.

The assembly joining operations include, mechanical, welded, and bonded/epoxy.

Mechanical Joins refer to connections that are fastened by bolts, fasteners, or compression fits.

Mechanical Join Required Information

- Fastening Method

  Bolted
- Fastener Quantity (integer)

- Required Torque (NM; real)

  Bolted Blind

- Fastener Quantity (integer)
- Required Torque (NM; real)

  Machined Screw

- Fastener Quantity (integer)
- Required Torque (NM; real)

  Press Fit

- Required Force (NM; real)

  Snap Fit

  Crimp/Clamp Fit

- Required Force (NM; real)

Welded Joins refer to connections between components/piece parts that are welded in place.
Welded Join Required Information

- Weld Type

  Seam

  Stitch

  Spot

- Weld Penetration

  Full

  Partial

- Weld Length (mm; real)
- Two-Sided Weld (enumeration; yes/no)
- Inspection Requirement (enumeration)

  Visual

  X-Ray

  Ultrasonic

- Joint Type (enumeration)

  Butt

  Corner

  Edge

  Lap

  Tree

- Part 1 Name (String)
- Part 1 Thickness (Real)
- Part 1 Material (String)
- Part 2 Name (String)

- Part 2 Thickness (Real)
- Part 2 Material (String)
-

Bonded/Epoxy Joins refer to connections between two or more components/piece parts that are joined by glue or epoxy.

Bonded/Epoxy Join Required Information

- Material (String)
- Volume (mm3; real)

2.7.5 XML Schemas

2.7.5.1 Manufacturing Details Component XML Schema

```
<?xml version="1.0" encoding="utf-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:annotation>
    <xs:appinfo>iFAB Foundry BOM Part Manufacturing Schema</xs:appinfo>
    <xs:documentation xml:lang="en">This schema defines an XML file for representing the
        manufacturing details for parts.</xs:documentation>
  </xs:annotation>

  <xs:include schemaLocation="common.xsd" />
  <xs:include schemaLocation="materials.xsd" />
  <xs:include schemaLocation="procurementDetails.xsd" />

  <xs:element name="manufacturingDetails">
    <xs:complexType>
      <xs:choice>
        <xs:element name="purchased" type="purchased" />
        <xs:element name="machined" type="machined" />
        <xs:element name="casting" type="casting" />
        <!-- <xs:element name="forging" type="forging" /> -->
        <xs:element name="plate" type="plate" />
        <xs:element name="pipeBarTube" type="pipeBarTube" />
        <!-- <xs:element name="additive" type="additive" /> -->
        <!-- <xs:element name="polymer" type="polymer" /> -->
        <!-- <xs:element name="plastic" type="plastic" /> -->
      </xs:choice>
    </xs:complexType>
  </xs:element>

  <xs:complexType name="baseClass" abstract="true">
    <xs:sequence>
    </xs:sequence>
  </xs:complexType>

  <xs:complexType name="purchased">
    <xs:complexContent>
      <xs:extension base="baseClass">
        <xs:sequence>
          <xs:element name="NSN" type="xs:string" minOccurs="0" />
          <!-- for backwards compatibility, too be removed -->
          <xs:choice minOccurs="1">
            <xs:element name="supplier" type="supplier" />
            <xs:element name="fabricationSupplier" type="fabricationSupplier" />
            <xs:element name="manufacturer" type="manufacturer" />

            <xs:element name="historicalSupplier" type="historicalSupplier" />
          </xs:choice>
        </xs:sequence>
      </xs:extension>
```

```
  </xs:complexContent>
</xs:complexType>

<xs:complexType name="machined">
  <xs:complexContent>
    <xs:extension base="baseClass">
      <xs:all>
        <xs:element name="material" type="machinedMaterials" />
        <xs:element name="planarFaces" type="generalTolerance" />
        <xs:element name="curvedSurfaces" type="generalTolerance" />
        <xs:element name="curvedWalls" type="generalTolerance" />
        <xs:element name="simpleHoles" type="simpleHoles" />
        <xs:element name="inorganicCoatings" type="inorganicCoatings" minOccurs="0" />
        <xs:element name="organicCoatings" type="organicCoatings" minOccurs="0" />
      </xs:all>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>

<xs:complexType name="casting">
  <xs:complexContent>
    <xs:extension base="baseClass">
      <xs:all>
        <xs:element name="material" type="castingMaterials" />
        <xs:element name="planarFaces" type="castingGeneralTolerance" />
        <xs:element name="curvedSurfaces" type="castingGeneralTolerance" />
        <xs:element name="curvedWalls" type="castingGeneralTolerance" />
        <xs:element name="simpleHoles" type="castingSimpleHoles" />
        <!-- <xs:element ref="heatTreatment" minOccurs="0" /> -->
        <xs:element name="inorganicCoatings" type="inorganicCoatings" minOccurs="0" />
        <xs:element name="organicCoatings" type="organicCoatings" minOccurs="0" />
      </xs:all>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>



<!--
<xs:complexType name="forging">
  <xs:complexContent>
    <xs:extension base="baseClass">
      <xs:all>
        <xs:element ref="baseMaterial" />
        <xs:element ref="material" />
        <xs:element ref="planarFaces" />
        <xs:element ref="curvedSurfaces" />
        <xs:element ref="curvedWalls" />
        <xs:element ref="simpleHoles" />
        <xs:element ref="heatTreatment" minOccurs="0" />
        <xs:element ref="inorganicCoatings" minOccurs="0" />
        <xs:element ref="organicCoatings" minOccurs="0" />
      </xs:all>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>
-->

<xs:complexType name="plate">
  <xs:complexContent>
    <xs:extension base="baseClass">
      <xs:all>
        <xs:element name="material" type="plateSheetMaterials" />
        <xs:element name="planarFaces" type="generalTolerance" />
        <xs:element name="bends" type="bends" />
        <xs:element name="simpleHoles" type="simpleHoles" />
        <xs:element name="complexHoles" type="generalTolerance" />
        <xs:element name="inorganicCoatings" type="inorganicCoatings" minOccurs="0" />
```

```
        <xs:element name="organicCoatings" type="organicCoatings" minOccurs="0" />
      </xs:all>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>

<xs:complexType name="pipeBarTube">
  <xs:complexContent>
    <xs:extension base="baseClass">
      <xs:all>
        <xs:element name="material" type="pipeBarTubeMaterials" />
        <xs:element name="ends" type="ends" />
        <xs:element name="bends" type="bends" />
        <xs:element name="simpleHoles" type="simpleHoles" />

        <xs:element name="complexHoles" type="generalTolerance" />
        <xs:element name="inorganicCoatings" type="inorganicCoatings" minOccurs="0" />
        <xs:element name="organicCoatings" type="organicCoatings" minOccurs="0" />
      </xs:all>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>

<!--
<xs:complexType name="additive">
  <xs:complexContent>
    <xs:extension base="baseClass">
      <xs:all>
        <xs:element ref="baseMaterial" />
        <xs:element ref="material" />
        <xs:element ref="planarFaces" />
        <xs:element ref="curvedSurfaces" />
        <xs:element ref="curvedWalls" />
        <xs:element ref="simpleHoles" />
        <xs:element ref="heatTreatment" minOccurs="0" />
        <xs:element ref="inorganicCoatings" minOccurs="0" />
        <xs:element ref="organicCoatings" minOccurs="0" />
      </xs:all>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>

<xs:complexType name="polymer">
  <xs:complexContent>
    <xs:extension base="baseClass">
      <xs:all>
        <xs:element ref="baseMaterial" />
        <xs:element ref="material" />
        <xs:element ref="planarFaces" />
        <xs:element ref="curvedSurfaces" />
        <xs:element ref="curvedWalls" />
        <xs:element ref="simpleHoles" />
        <xs:element ref="organicCoatings" minOccurs="0" />
      </xs:all>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>


<xs:complexType name="plastic">
  <xs:complexContent>
    <xs:extension base="baseClass">
      <xs:all>
        <xs:element ref="baseMaterial" />
        <xs:element ref="material" />
        <xs:element ref="process" />
        <xs:element ref="planarFaces" />
        <xs:element ref="curvedSurfaces" />
```

```
        <xs:element ref="curvedWalls" />
        <xs:element ref="simpleHoles" />
        <xs:element ref="organicCoatings" minOccurs="0" />
      </xs:all>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>
-->

<!-- <xs:element name="heatTreatment" type="heatTreatment" /> -->
<!-- <xs:element name="surfaceTreatment" type="surfaceTreatmentType" default="None" /> -->

<xs:complexType name="bends">
  <xs:all>
    <xs:element name="bendAngleTolerance" type="angle" default="0.1" />
  </xs:all>
</xs:complexType>

<xs:complexType name="castingGeneralTolerance">
  <xs:all>
    <xs:element name="generalTolerance" type="length" default="0.254" />
    <xs:element name="surfaceRoughness" type="length" default="0.006352" />
  </xs:all>
</xs:complexType>

<xs:complexType name="castingSimpleHoles">
  <xs:all>
    <xs:element name="diametricalTolerance" type="length" default="0.254" />
    <xs:element name="positionalTolerance" type="length" default="0.254" />
    <xs:element name="surfaceRoughness" type="length" default="0.006352" />
    <xs:element name="threaded" type="xs:boolean" default="false" />
  </xs:all>
</xs:complexType>


<xs:complexType name="ends">
  <xs:all>
    <xs:element name="generalTolerance" type="length" default="0.254" />
    <xs:element name="surfaceRoughness" type="length" default="0.003176" />
  </xs:all>
</xs:complexType>

<xs:complexType name="generalTolerance">
  <xs:all>
    <xs:element name="generalTolerance" type="length" default="0.127" />
    <xs:element name="surfaceRoughness" type="length" default="0.001588" />
  </xs:all>
</xs:complexType>

<xs:complexType name="simpleHoles">
  <xs:all>
    <xs:element name="diametricalTolerance" type="length" default="0.127" />
    <xs:element name="positionalTolerance" type="length" default="0.127" />
    <xs:element name="surfaceRoughness" type="length" default="0.001588" />
    <xs:element name="threaded" type="xs:boolean" default="false" />
  </xs:all>
</xs:complexType>

<!-- heat treatment option is currently disabled

<xs:complexType name="heatTreatment">
  <xs:all>
    <xs:element name="type" type="heatTreatmentType" />
    <xs:element name="hardness" type="hardness" />
  </xs:all>
</xs:complexType>

-->
```

```
  <xs:complexType name="inorganicCoatings">
    <xs:sequence>
      <xs:element name="inorganicCoating" type="inorganicCoatingType" minOccurs="0"
          maxOccurs="unbounded" />
    </xs:sequence>
  </xs:complexType>

  <xs:complexType name="organicCoatings">
    <xs:sequence>
      <xs:element name="organicCoating" type="organicCoatingType" minOccurs="0" maxOccurs="
          unbounded" />
    </xs:sequence>
  </xs:complexType>

  <!-- the heat treatment option is currently disabled (derived from properties)

  <xs:simpleType name="heatTreatmentType">
    <xs:restriction base="xs:token">
      <xs:enumeration value="None" />
      <xs:enumeration value="Annealing" />
      <xs:enumeration value="Hardening" />
      <xs:enumeration value="Tempering" />
      <xs:enumeration value="Surface␣Carburizing" />
      <xs:enumeration value="Carbon␣Restoration" />
      <xs:enumeration value="Induction␣Hardening" />
    </xs:restriction>
  </xs:simpleType>
-->

  <!-- the surface treatment option is currently disabled
  <xs:simpleType name="surfaceTreatmentType">
    <xs:restriction base="xs:token">
      <xs:enumeration value="None" />
      <xs:enumeration value="Shot␣Peening" />
      <xs:enumeration value="Blasting" />
    </xs:restriction>
  </xs:simpleType>

-->

  <xs:simpleType name="inorganicCoatingType">
    <xs:restriction base="xs:token">
      <xs:enumeration value="Black␣Chrome" />
      <xs:enumeration value="Cadmium" />
      <xs:enumeration value="Sulfuric␣Acid␣Annodize" />
      <xs:enumeration value="Hard␣Coat␣Annodize" />
      <xs:enumeration value="Manganese␣Phosphate" />
      <xs:enumeration value="Black␣Oxide" />
      <xs:enumeration value="Nickel" />
      <xs:enumeration value="Zinc␣Phosphate" />
    </xs:restriction>
  </xs:simpleType>

  <xs:simpleType name="organicCoatingType">
    <xs:restriction base="xs:token">
      <xs:enumeration value="CARC" />
      <xs:enumeration value="non-CARC" />
    </xs:restriction>
  </xs:simpleType>

  <xs:simpleType name="processType">
    <xs:restriction base="xs:token">
      <xs:enumeration value="Injection␣Molding" />
      <xs:enumeration value="Blow␣Molding" />
      <xs:enumeration value="Roto␣Molding" />
    </xs:restriction>
  </xs:simpleType>
```

```
<!-- Purchased part elements defined below -->

<xs:complexType name="CAGECode">
  <xs:sequence>
    <xs:element name="code" type="xs:string" />
  </xs:sequence>
</xs:complexType>

<xs:complexType name="CAGEInformation">
  <xs:complexContent>
    <xs:extension base="contactInfo">
      <xs:sequence>
        <xs:element name="companyName" type="xs:string" minOccurs="0" />
        <xs:element name="faxNumber" type="phoneNumber" minOccurs="0" />
      </xs:sequence>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>

<xs:complexType name="CAGE">
  <xs:sequence>
    <xs:element name="CAGEIdentifier">
      <xs:complexType>
        <xs:sequence>
          <xs:choice minOccurs="1" maxOccurs="unbounded">
            <xs:element name="CAGECode" type="CAGECode" />
            <xs:element name="CAGEInformation" type="CAGEInformation" />
          </xs:choice>
          <xs:element name="componentInfo" type="xs:anyURI" minOccurs="0" />
          <xs:element name="sourcingAndProcurement" minOccurs="0">
            <xs:complexType>
              <xs:choice minOccurs="1" maxOccurs="unbounded">
                <xs:element name="COTS" type="COTS" />
                <xs:element name="GFE" type="GFE" />
                <xs:element name="MTO" type="MTO" />
                <xs:element name="curation" type="curation" />
              </xs:choice>
            </xs:complexType>
          </xs:element>
        </xs:sequence>
      </xs:complexType>
    </xs:element>
  </xs:sequence>
</xs:complexType>

<xs:simpleType name="packaging">
  <xs:restriction base="xs:token">
    <xs:enumeration value="Pallet" />
    <xs:enumeration value="Box" />
    <xs:enumeration value="Crate" />
  </xs:restriction>
</xs:simpleType>

<xs:complexType name="transportationNeeds">
  <xs:sequence minOccurs="0" maxOccurs="unbounded">
    <xs:element name="transportationNeed" type="transportationNeed" />
  </xs:sequence>
</xs:complexType>

<xs:simpleType name="transportationNeed">
  <xs:restriction base="xs:token">
    <xs:enumeration value="TemperatureControl" />
    <xs:enumeration value="HazardousMaterial" />
  </xs:restriction>
</xs:simpleType>
```

```
<xs:complexType name="shipmentOptions">
  <xs:sequence minOccurs="1" maxOccurs="unbounded">
    <xs:element name="shipmentOption" type="shipmentOption" />
  </xs:sequence>
</xs:complexType>

<xs:complexType name="shipmentOption">
  <xs:sequence>
    <xs:element name="minimumQuantity" type="xs:nonNegativeInteger" minOccurs="0" default=
        "0" />
    <xs:element name="maximumQuantity" type="xs:nonNegativeInteger" minOccurs="0" default=
        "1000000000" />
    <xs:element name="leadTime" type="duration" />
    <xs:element name="leadTimeUncertainty" type="xs:double" minOccurs="0" default="0.0" />
    <xs:element name="deliveryRate" type="duration" minOccurs="0" />
    <xs:element name="price" type="price" />
    <xs:element name="shipmentLength" type="length" minOccurs="0" />
    <xs:element name="shipmentWidth" type="length" minOccurs="0" />
    <xs:element name="shipmentHeight" type="length" minOccurs="0" />
    <xs:element name="shipmentWeight" type="weight" minOccurs="0" />
    <xs:element name="packaging" type="packaging" minOccurs="0" default="Pallet" />
  </xs:sequence>
</xs:complexType>

<xs:complexType name="supplier">
  <xs:complexContent>
    <xs:extension base="CAGE">
      <xs:sequence>
        <xs:element name="catalogNumber" type="xs:string" minOccurs="0" />
        <xs:element name="partDescription" type="xs:string" minOccurs="0" />
        <xs:element name="FOB" type="FOB" minOccurs="0" default="destination" />
        <xs:element name="paymentTerms" type="xs:string" minOccurs="0" />
        <xs:element name="shipmentOptions" type="shipmentOptions" />
        <xs:element name="shippingAddress" type="companyInfo" />
        <xs:element name="transportationNeeds" type="transportationNeeds" minOccurs="0" />
        <xs:element name="unitOfIssue" type="xs:string" minOccurs="0" />
      </xs:sequence>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>

<xs:complexType name="fabricationSupplier">
  <xs:complexContent>
    <xs:extension base="CAGE">
      <xs:sequence>
        <xs:element name="catalogNumber" type="xs:string" minOccurs="0" />
        <xs:element name="partDescription" type="xs:string" minOccurs="0" />
        <xs:element name="FOB" type="FOB" minOccurs="0" default="destination" />
        <xs:element name="paymentTerms" type="xs:string" minOccurs="0" />
        <xs:element name="shipmentOptions" type="shipmentOptions" />
        <xs:element name="shippingAddress" type="companyInfo" />
        <xs:element name="transportationNeeds" type="transportationNeeds" minOccurs="0" />
        <xs:element name="unitOfIssue" type="xs:string" minOccurs="0" />
        <xs:element name="productionPartCost" type="price" />
        <xs:element name="productionPartCostUncertainty" type="xs:double" minOccurs="0"
            default="0.0" />
        <xs:element name="productionPartNRECost" type="price" />
        <xs:element name="productionPartToolingCost" type="price" />
        <xs:element name="productionQuantity" type="xs:nonNegativeInteger" minOccurs="0"
            default="50" />
        <xs:element name="prototypePartCost" type="price" />
        <xs:element name="prototypeCostUncertainty" type="xs:double" minOccurs="0" default
            ="0.0" />
        <xs:element name="prototypeNRECost" type="price" />
        <xs:element name="prototypePartToolingCost" type="price" />
        <xs:element name="prototypeQuantity" type="xs:nonNegativeInteger" minOccurs="0"
            default="1" />
      </xs:sequence>
```

```
      </xs:extension>
    </xs:complexContent>
  </xs:complexType>

  <!-- this should be avoided since no information is provided for automated analysis -->
  <xs:complexType name="manufacturer">
    <xs:complexContent>
      <xs:extension base="CAGE">
        <xs:sequence>
          <xs:element name="partDescription" type="xs:string" />
          <xs:element name="partNumber" type="xs:string" />
        </xs:sequence>
      </xs:extension>
    </xs:complexContent>
  </xs:complexType>

  <!-- this should be avoided since information is missing for automated analysis -->
  <xs:complexType name="historicalSupplier">
    <xs:complexContent>
      <xs:extension base="CAGE">
        <xs:sequence>
          <xs:element name="lastPurchased" type="xs:date" minOccurs="0" />
          <xs:element name="price" type="price" />
          <xs:element name="priceRangeMinimum" type="price" minOccurs="0" />
          <xs:element name="priceRangeMaximum" type="price" minOccurs="0" />
          <xs:element name="quantity" type="xs:nonNegativeInteger" minOccurs="0" default="1"
              />
          <xs:element name="unitOfIssue" type="xs:string" minOccurs="0" />
        </xs:sequence>
      </xs:extension>
    </xs:complexContent>
  </xs:complexType>
</xs:schema>
```

2.7.5.2 Assembly XML Schema

```
<?xml version="1.0" encoding="utf-8"?>

<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">

  <xs:annotation>
    <xs:appinfo>iFAB Foundry Assembly Details XML Schema</xs:appinfo>
    <xs:documentation xml:lang="en">This schema defines an XML file for representing
        assembly details.</xs:documentation>
  </xs:annotation>

  <xs:include schemaLocation="common.xsd" />

  <xs:element name="assemblyDetails">
    <xs:complexType>
      <xs:choice minOccurs="0" maxOccurs="unbounded">
        <xs:element name="mechanical" type="mechanical" />
        <xs:element name="welded" type="welded" />
        <xs:element name="soldered" type="soldered" />
        <xs:element name="brazed" type="brazed" />
        <xs:element name="glued" type="glued" />
      </xs:choice>
    </xs:complexType>
  </xs:element>

  <xs:complexType name="baseSeam" abstract="true">
    <xs:sequence>
      <xs:element name="part1" type="part" />
      <xs:element name="part2" type="part" />
      <xs:element name="description" type="xs:string" minOccurs="0" />
    </xs:sequence>
    <xs:attribute ref="id" use="required" />
  </xs:complexType>
```

```
<xs:complexType name="mechanical">
  <xs:complexContent>
    <xs:extension base="baseSeam">
      <xs:sequence>
        <xs:element name="fasteningMethod" type="fasteningMethod" />
        <xs:element name="fasteningQuantity" type="xs:positiveInteger" minOccurs="0"
            default="1" />
        <xs:element name="torque" type="torque" minOccurs="0" />
        <xs:element name="force" type="force" minOccurs="0" />
        <xs:element name="components" type="components" minOccurs="0" />
      </xs:sequence>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>

<xs:complexType name="welded">
  <xs:complexContent>
    <xs:extension base="baseSeam">
      <xs:sequence>
        <xs:element name="length" type="length" />
        <xs:element name="jointType" type="jointType" />
        <xs:element name="weldType" type="weldType" />
        <xs:element name="weldPenetration" type="weldPenetration" />
        <xs:element name="twoSided" type="xs:boolean" />
        <xs:element name="inspectionRequirement" type="inspectionRequirement" />

        <xs:element name="part1Thickness" type="length" />
        <xs:element name="part1Material" type="baseMaterial" />
        <xs:element name="part2Thickness" type="length" />
        <xs:element name="part2Material" type="baseMaterial" />
      </xs:sequence>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>

<xs:complexType name="soldered">
  <xs:complexContent>
    <xs:extension base="baseSeam">
      <xs:sequence>
        <xs:element name="length" type="length" />
        <xs:element name="fillerMaterial" type="fillerMaterial" />
        <xs:element name="fluxMaterial" type="fluxMaterial" />
      </xs:sequence>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>

<xs:complexType name="brazed">
  <xs:complexContent>
    <xs:extension base="baseSeam">
      <xs:sequence>
        <xs:element name="length" type="length" />
        <xs:element name="fillerMaterial" type="fillerMaterial" />
        <xs:element name="fluxMaterial" type="fluxMaterial" />
      </xs:sequence>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>

<xs:complexType name="glued">
  <xs:complexContent>
    <xs:extension base="baseSeam">
      <xs:sequence>
        <xs:element name="length" type="length" />
        <xs:element name="volume" type="volume" />
        <xs:element name="material" type="glue" />
      </xs:sequence>
```

```
    </xs:extension>
  </xs:complexContent>
</xs:complexType>

<xs:simpleType name="baseMaterial">
  <xs:restriction base="xs:token">
    <xs:enumeration value="Al" />
    <xs:enumeration value="Plain␣Carbon␣Steel" />
  </xs:restriction>
</xs:simpleType>

<xs:simpleType name="fillerMaterial">
  <xs:restriction base="xs:string" />
</xs:simpleType>

<xs:simpleType name="fluxMaterial">
  <xs:restriction base="xs:string" />
</xs:simpleType>

<xs:simpleType name="glue">
  <xs:restriction base="xs:string" />
</xs:simpleType>

<xs:complexType name="components">
  <xs:sequence minOccurs="0" maxOccurs="unbounded">
    <xs:element name="component" type="component" />
  </xs:sequence>
</xs:complexType>

<xs:complexType name="component">
  <xs:attribute ref="id" use="required" />
  <xs:attribute name="number" type="xs:string" use="optional" />
  <xs:attribute name="description" type="xs:string" use="optional" />
  <xs:attribute name="quantity" type="xs:positiveInteger" use="required" />
</xs:complexType>

<xs:simpleType name="fasteningMethod">
  <xs:restriction base="xs:token">
    <xs:enumeration value="Bolted" />
    <xs:enumeration value="Bolted␣(blind)" />
    <xs:enumeration value="Machined␣Screw" />
    <xs:enumeration value="Press␣Fit" />
    <xs:enumeration value="Snap␣Fit" />
    <xs:enumeration value="Crimp/Clamp␣Fit" />
  </xs:restriction>
</xs:simpleType>

<xs:simpleType name="jointType">
  <xs:restriction base="xs:token">
    <xs:enumeration value="Butt" />
    <xs:enumeration value="Corner" />
    <xs:enumeration value="Edge" />
    <xs:enumeration value="Lap" />
    <xs:enumeration value="Tee" />
  </xs:restriction>
</xs:simpleType>

<xs:simpleType name="weldType">
  <xs:restriction base="xs:token">
    <xs:enumeration value="Seam" />
    <xs:enumeration value="Stitch" />
    <xs:enumeration value="Spot" />
  </xs:restriction>
</xs:simpleType>

<xs:simpleType name="weldPenetration">
  <xs:restriction base="xs:token">
    <xs:enumeration value="Full" />
```

```
      <xs:enumeration value="Partial" />
    </xs:restriction>
  </xs:simpleType>

  <xs:simpleType name="inspectionRequirement">
    <xs:restriction base="xs:token">
      <xs:enumeration value="Visual" />
      <xs:enumeration value="X-Ray" />
    </xs:restriction>
  </xs:simpleType>


  <xs:complexType name="part">
    <xs:simpleContent>
      <xs:extension base="xs:string">
        <xs:attribute ref="id" use="required" />
      </xs:extension>
    </xs:simpleContent>
  </xs:complexType>

</xs:schema>
```

2.8 General Modeling Conventions

2.8.1 Domain Models at Multiple Levels of Fidelity

An AVM Component Model may come with multiple models within a given domain (CAD, Modelica, etc), each capturing component behavior or characteristics at differering levels of fidelity.

For details on how these "alternatives" are captured in the AVM Component Model, refer to the *Domain Model Alternatives* section of *Section 2.3.2.3: Domain Model Sublanguage*.

2.8.2 CAD

AVM Components may represent objects at varying levels of complexity. An AVM Component may represent a single part, such as a metal plate. It could represent a collection of objects that we could think of as an assembly in the real world, such as a diesel engine, but be captured by a single CAD part file. It could also represent the same assembly, but captured by a CAD assembly model with subordinate part files.

Depending on how the Component is modeled, certain conventions are required to ensure that all of the artifacts associated with a Component Model carry consistent identities. This is especially the case when analysis tools need to cross-reference elements in different domains, such as establishing references between CAD geometry models and detailed manufacturing models.

2.8.3 Component Represented by a Single Part Model

CAD representations of the component must capture it's geometry as a single part file. If multiple CAD models are provided, they are assumed to be of alternate levels of detail, or specialized for different purposes.

2.8.4 Component Represented by an Assembly of Parts

For some components, like brackets, the geometric representation may be provided as a CAD Assembly with a number of subordinate parts. Naming conventions are used to establish the unique identity of each instance of each part.

2.8.4.1 CAD

Each instance of each part in the assembly must have a unique part file. Even if two part instances are identical, they must be represented as unique part files with unique names.

Using an example of a bracket, featuring two identical plates welded together, we expect to see 3 CAD files: one for the assembly, and one for each plate.

2.8.4.2 Manufacturing

Each instance of each part must have a unique Manufacturing model. Each should be named to match the corresponding CAD part file. Any references to individual parts within these Manufacturing models must use this name also.

Using the same example of a bracket, featuring two identical plates welded together, we expect to see at least 3 Manufacturing models: one for each plate (identified by their respective unique names), and one representing the weld between them (referring to part1 and part2 using those unique part names).

2.8.4.3 Important Note

Since identities are established by file naming conventions, this means that only ONE of each component of this type may be included in a design. Using the bracket example, only one instance of that bracket may be instantiated. Otherwise, there will be name and identity conflicts.

This is a known limitation of this approach (VU META and iFAB are aware of this). The future plan is to find a reliable way to modify the part identities to include a notion of the component instance. This will probably take the form of an intelligent script that knows how to find the key identities and consistently prepend a "component instance" string to them.

Chapter 3

Component Authoring and Curation

3.1 Overview

A key consideration underlying the design of AVM Component model, was the ability to incorporate existing model asset base which often exists in mature engineering organizations. This core feature of AVM component model, while enabling reuse of existing IP, renders it challenging to author AVM component model as the multiple domain models, developed in multiple tools, need to be wrapped and packaged together. In order to aid the engineers and developers in authoring AVM components, a component authoring framework is being developed and provided along with the META tools.

In order to ensure quality and conformance of authored components, AVM defines a component curation process. The curation process verifies the delivered components, establishes in-class conformance of delivered components, validates the submitted component data using subject matter expertise, and hosts the submitted component on a Component Exchange infrastructure implemented by the Vehicle Forge project of AVM. The Component Exchange, similar to an online catalog, also implements services for discovery of components using a class taxonomy as well as based on component properties.

3.2 Component Authoring Toolkit

AVM Component Models incorporate metadata from their corresponding Domain Models, as well as some global properties of the component. Using this data, they describe the interfaces though which the Domain Models should be composed into a system model. They also provide structures to keep the parameters of each Domain Model synchronized.

To support the creation and maintenance of these structures within AVM Component Models, a Component Authoring Toolkit is provided with the OpenMETA CyPhy tools.

The framework addresses the following two primary use-cases:

1. Creating a New Component Class
2. Creating a New Instance of an Existing Component Class

Figure 3.1 depicts the workflow involved in authoring a new component class focusing on CAD domain model. The component authoring is initiated from the OpenMETA CyPhy tools. Subsequently, the author launches the CAD authoring tool (PTC Creo Parametric) using the META-Link capability of OpenMETA Tools. This enables the author to automatically construct domain model ports and CAD parameters in the OpenMETA CyPhy environment. Once the component ports are created, the author can invoke the Component Authoring tool from OpenMETA, and package additional artifacts required for the component and created the complete component package. The authoring tool also includes component validation tools (described later) that can be used to validate the authored components. Additionally, the authoring tool also provides capability to automatically submit the authored component for curation through the Vehicle Forge portal.

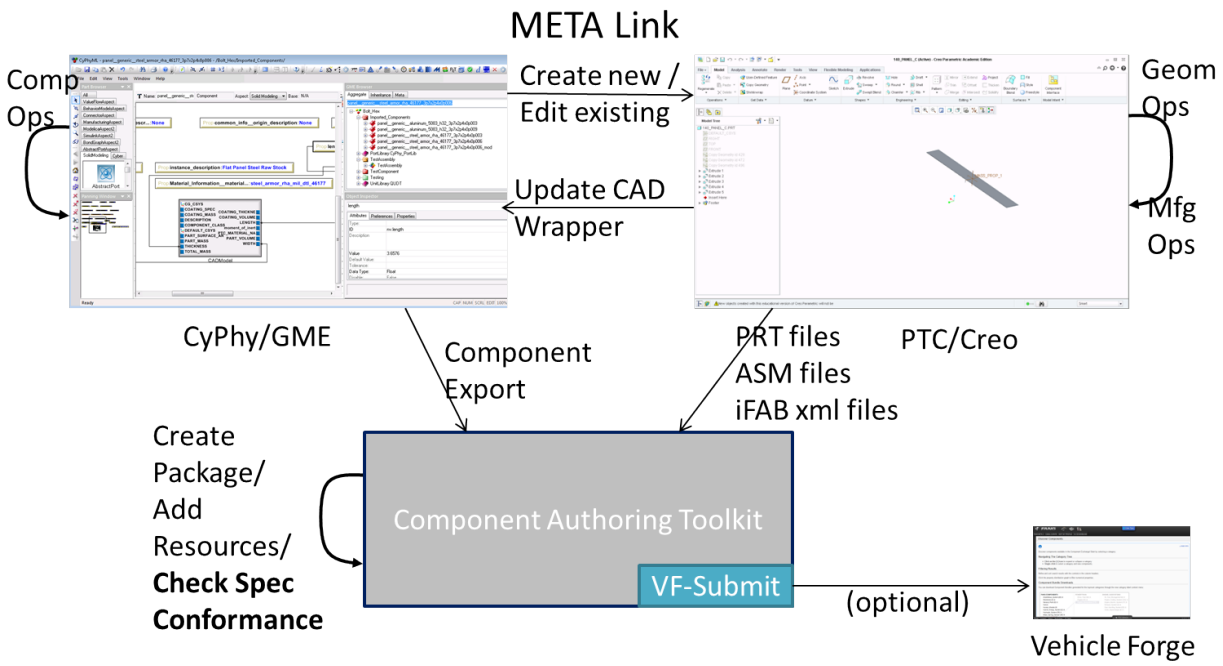

Figure 3.1: Component Authoring Workflow (New Component Class)

Figure 3.2 depicts a similar workflow for authoring a new instance of an existing component class. In this second use case parameterized templates and domain models already exist for the component class, and the component author needs to instantiate parameter values for their specific component instance. When parametric domain models are not available, as often the case with discrete CAD models, the author must create the domain model wrapper for the CAD model using the META-Link and CAD authoring tool (PTC/Creo parametric currently)

In support of these use cases, the Component Authoring Toolkit includes modules for common tasks, such as:

- Load a component model template and allow users to edit parameters
- Import metadata from a class in a Modelica library (parameters, connectors)
- Import metadata from a Creo part or assembly (parameters, datums)

Additional details on the Component Authoring Toolsuite, are included in the OpenMETA CyPhy documentation.

3.3 Curation Workflow

Component authors can submit components for curation using the Vehicle Forge portal. The curation workflow verifies the component and checks the in-class conformance of a given component. Optionally, the curation workflow can also validate the component with the involvement of subject matter experts. During submission the component author can define the distribution parameters and restriction of the submitted component, depending on which the component is repositoried in the global component exchange or a local component exchange.

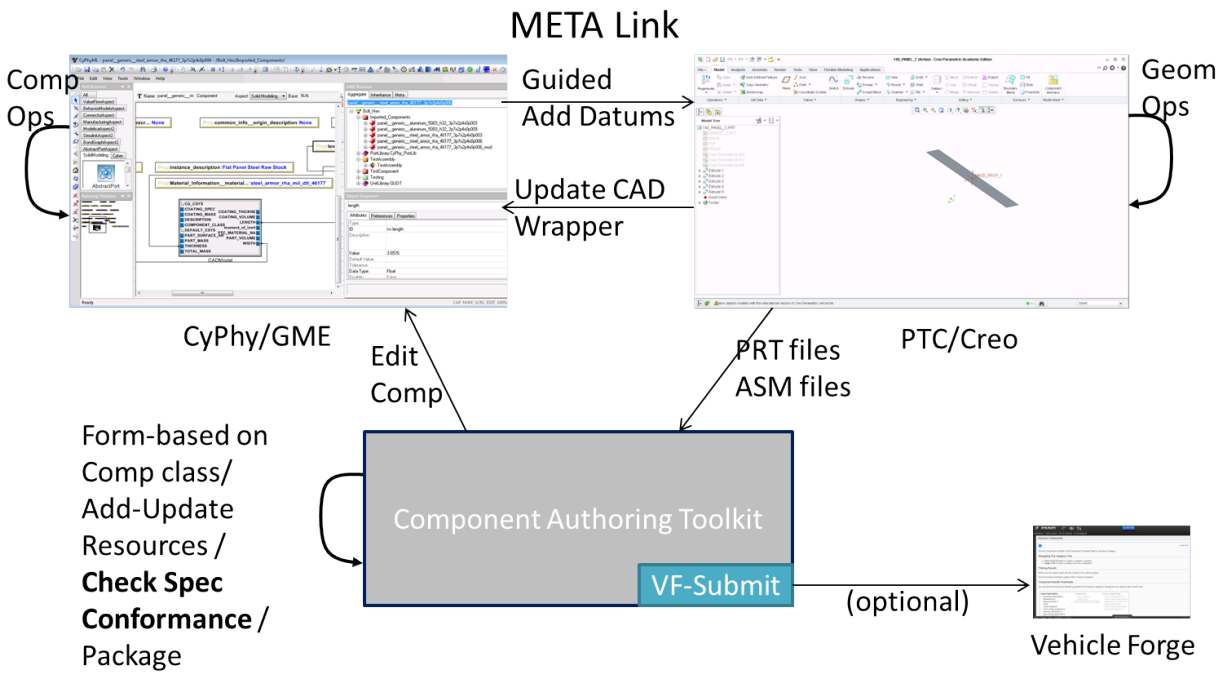

Figure 3.2: Component Authoring Workflow (New Component Instance)

3.4 Component Model APIs

See *Section 2.3.3.2: Software Library.*

3.5 Component Model Taxonomy

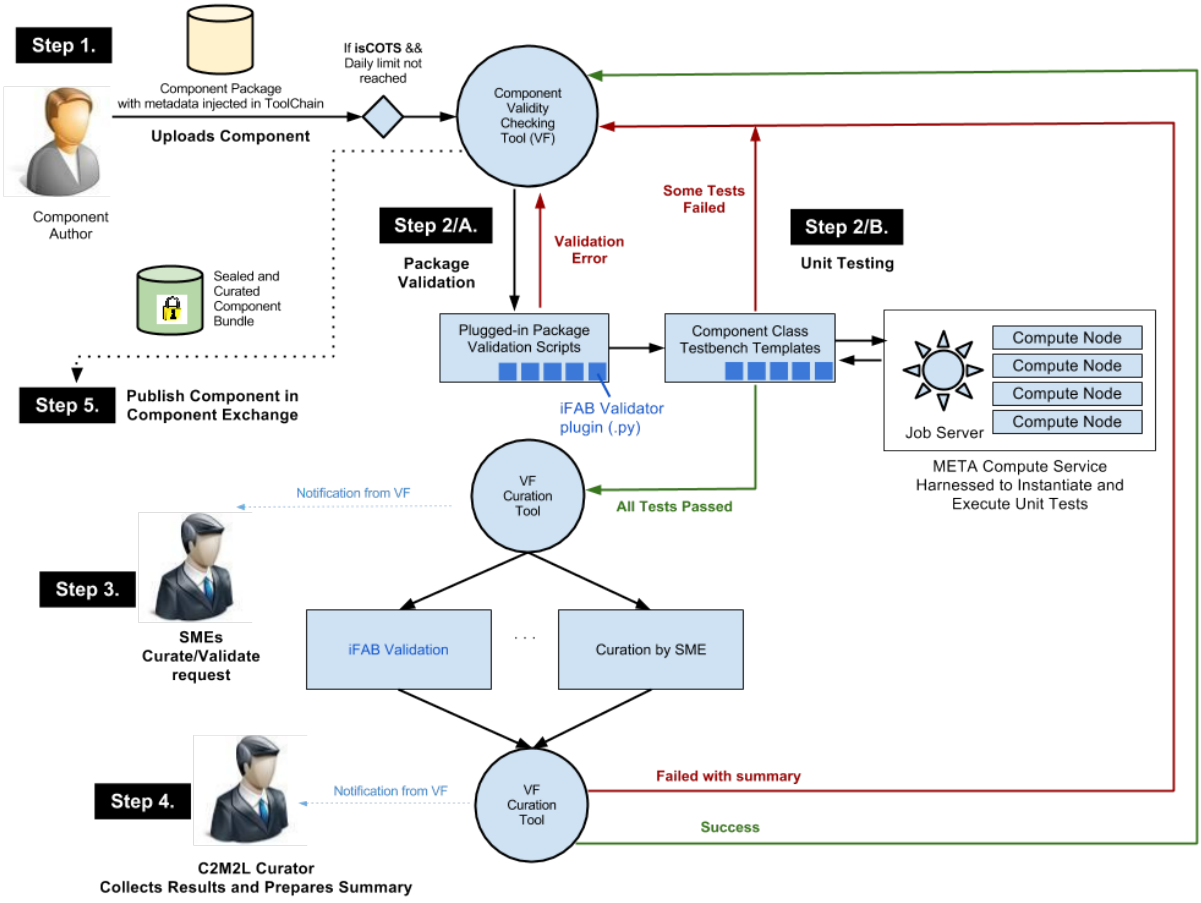

Figure 3.3: Component Curation Workflow

Chapter 4

Component Model Comformance Suite

A "conformance suite" is also provided for checking the validity of AVM Component Models. In addition to checking ACM files against the XML schema, the structures within are checked for semantic consistency and well-formedness.

4.1 Schema

An XML Validator included in the libxml package of Python is used to validate the submitted ACM (xml) file against AVM Component Model schema. See *Section 2.3.3.3: Schema*.

4.2 Python Validator Script

The Python validator script (drop_test.py) parses the AVM Component spec model and performs the following semantic checks on the component model.

- Check Resources
- Check Properties and Values
- Check Connectors
- Check Domain Models
  - Check CAD Domain Models
    - Check Datums
    - Check Parameters
  - Check Modelica Domain Models
    - Check Connector Definitions
    - Check Medium Specifications
    - Check Parameters
  - Check Manufacturing Domain Models
    - Check Parameters

The listing of the Python Validator Script is provided below:

```
from os import listdir
from os.path import isfile, join
from optparse import OptionParser
from fnmatch import fnmatch
from glob import glob
from lxml import etree
import sys
import tablib

XSI = "{http://www.w3.org/2001/XMLSchema-instance}"

# Stats dictionary structure
# comp_name, file_name, schema_valid, has_cad, has_modelica,
      has_manufacturing, valid_resources,
#              num_connectors, num_properties

# Category component counts
catCounts = dict()

def list_cxmls(mypath):  # return a list of xml files in the given directory
    """Return a list containing component xml files in a given directory."""
    xmls = [join(mypath,f) for f in listdir(mypath) if
        isfile(join(mypath,f)) and fnmatch(f, '*.acm')]
    return xmls


def schema_validate(xml, xsd):  # validates provided xml files against schema
    """Return true or false based on whether provided XML is valid against
       schema"""
    xmlschema_doc = etree.parse(xsd)
    xmlschema = etree.XMLSchema(xmlschema_doc)
    doc = etree.parse(xml)
    if xmlschema.validate(doc):
        return doc
    else:
        return None


def check_classification(xmldoc, statsData):  # checks that the component
                                                has a valid classification tag
    root = xmldoc.getroot()
    classif = root.findall('Classifications')
    hasClassif = True
    classifs = ''
    if classif is None or len(classif) == 0:
        hasClassif = False
    else:
        classifs = classif[0].text
    statsData.append(classifs)
    if classifs in catCounts:
        catCounts[classifs] += 1
    else:
        catCounts[classifs] = 1
```

```
    return hasClassif


def check_resources(xmldoc, mypath, statsData):  # checks that the resource
                                                   link in the XML file are valid
    """Return true or false based on whether the resource objects in the
       XMl point to valid files"""
    root = xmldoc.getroot()
    resources = root.findall("ResourceDependency")
    validResources = True
    for child in resources:
        res = child.get("Path")
        res_w = join(mypath, res) + '*'
        if len([n for n in glob(res_w) if isfile(n)]) == 0:
            validResources = False
    statsData.append(validResources)
    return True

def check_domain_resources(domain, root):
    uses = domain.get("UsesResource")
    resources = root.findall("ResourceDependency")
    hasValidResource = False
    for res in resources:
        rid = res.get("ID")
        if uses in rid:
            hasValidResource = True
            break
    if not hasValidResource:
        print ' '


    return True


def check_cad_model(cadroot, root):
    datums = cadroot.findall("Datum")
    hasCoordinateSystem = False
    hasPlanesAxis = False
    for d in datums:
        dtype = d.get(XSI + "type")
        if "CoordinateSystem" in dtype:
            hasCoordinateSystem = True
        elif "Point" in dtype or "Axis" in dtype or "Plane" in dtype:
            hasPlanesAxis = True

    if not hasPlanesAxis:
        print ' '

    return True


def check_modelica_model(moroot, root):
    return True
```

```
def check_manufacturing_model(maroot, root):
    return True


def check_domain_models(xmldoc, statsData):  # checks that the resource link in the XML fi
    """Return true or false based on whether the resource objects in the
       XMl point to valid files
    :param xmldoc: the XML element tree
    """
    root = xmldoc.getroot()
    domains = root.findall("DomainModel")
    hasCAD = False
    hasModelica = False
    hasManufacturing = False

    for child in domains:
        res = child.get(XSI + "type")
        check_domain_resources(child, root)
        if "CADModel" in res:
            hasCAD = check_cad_model(child, root)
        elif "ModelicaModel" in res:
            hasModelica = check_modelica_model(child, root)
        elif "ManufacturingModel" in res:
            hasManufacturing =  check_manufacturing_model(child, root)

    if not hasManufacturing:
        print 'ERROR: Component[{0}]: No Manufacturing Model'.format(root.get('Name'))
    statsData.extend([hasCAD, hasModelica, hasManufacturing])

    return True


def is_number(s):
    if s is not None:
        try:
            float(s)
            return True
        except ValueError:
            return False
    else: return False


def is_list(myinput):
    par=myinput.replace('[','')
    par=par.replace(']','')
    mat=par.split(',')
    p = True
    for x in mat:
        if not is_number(x):p=False
    if p: return True
    else: return False
```

```
def check_property(prop, proptype, root, propval, propnames2, errorcheck,
       warningcheck, fileoutput, concheck):  # checks a specific property
    if "CompoundProperty" in proptype:
        # check if compound is non empty and has unique children
        pprops = prop.findall("PrimitiveProperty")
        cprops = prop.findall("CompoundProperty")
        props = pprops + cprops
        if props is None or len(props) == 0:
            errorcheck += 1
            fileoutput.write('WARNING: Component[{0}]: Compound
              Property[{1}]: has no child property\n'.\
                format(root.get('Name'), prop.get('Name')))
        propnames = [p.get('Name') for p in props]
        if len(propnames) != len(set(propnames)):
            errorcheck += 1
            fileoutput.write('ERROR: Component[{0}]: Compound
              Property[{1}]: duplicate child properties\n'.\
                format(root.get('Name'), prop.get('Name')))
        for p in cprops:
            notes = check_property(p, 'CompoundProperty', root, propval,
              propnames2, errorcheck, warningcheck, fileoutput, concheck)
            warningcheck = notes[0];
            errorcheck = notes[1];
            propval = notes[2];
            propnames2 = notes[3];
            concheck = notes[4];
        for p in pprops:
            notes = check_property(p, 'PrimitiveProperty', root, propval,
              propnames2, errorcheck, warningcheck, fileoutput, concheck)
            warningcheck = notes[0];
            errorcheck = notes[1];
            propval = notes[2];
            propnames2 = notes[3];
            concheck = notes[4];
    elif "PrimitiveProperty" in proptype:
        # check if primitive has a unique value
        vals = prop.findall('Value')
        if vals is None or len(vals) != 1:
            errorcheck += 1
            fileoutput.write('ERROR: Component[{0}]: Primitive
             Property[{1}]: has illegal value\n'.\
                format(root.get('Name'), prop.get('Name')))

            ## print prop.get('Name')
        else:
            ##if len(propnames2) < 16:
                ValEx=vals[0].findall('ValueExpression')
                basicAtt1=vals[0].attrib.get('DataType')
                basicAtt2=vals[0].attrib.get('Dimensions')
                basicAtt3=vals[0].attrib.get('Unit')
                basicAtt4=ValEx[0].attrib.get('ValueSource')

##                print ValEx
                if (basicAtt4 != None):
```

```
##                     print basicAtt4
                    if ('CAD' in basicAtt4 or 'cad' in basicAtt4):
                        concheck += 1;
##                        print concheck
##                    root = basicAtt4.getroot()
##                    print root

                if ValEx != []:
                    aa=ValEx[0].findall('Value')
                    if aa != []:
                        myVal=aa[0].text
                        if myVal == [] or myVal == None:                        # check if
                            propval += 1;
                            errorcheck += 1;
                            fileoutput.write('ERROR: Component[{0}]:
                             Primitive Property[{1}]: has missing value\n'.\
                                format(root.get('Name'), prop.get('Name')))
                        elif basicAtt1=='Real':
                            if basicAtt3 == [] or basicAtt3 == None:
                                warningcheck += 1;
                                fileoutput.write('WARNING: Component[{0}]:
                                 Primitive Property[{1}]: No Units\n'.\
                                    format(root.get('Name'), prop.get('Name')))
                            if basicAtt2 ==
                             '1':                                  # check if
                                            scalars are numbers
                                if not is_number(myVal):
                                    propval += 1;
                                    errorcheck += 1;
                                    fileoutput.write('ERROR:
                                     Component[{0}]: Primitive
                                      Property[{1}]: has incorrect data
                                       type\n'.\
                                        format(root.get('Name'), prop.get('Name')))
                            else:
                                if is_number(basicAtt2):
                                    myLength = int(basicAtt2)
                                else:
                                    myDim=basicAtt2.split('x')
                                    myLength = int(myDim[0]) * int(myDim[1])
                                if not
                                 is_list(myVal):                               #
                                  check if matrix/vector is all numbers
                                    propval += 1;
                                    errorcheck += 1;
                                    fileoutput.write('ERROR:
                                     Component[{0}]: Primitive
                                      Property[{1}]: has incorrect data
                                       type\n'.\
                                        format(root.get('Name'), prop.get('Name')))
                                if not myVal.count(',') + 1 ==
                                 myLength:        # check if matrix/vector
                                  has right number of entries
                                    propval += 1;
```

```
                                        errorcheck += 1;
                                        fileoutput.write('ERROR:
                                         Component[{0}]: Primitive
                                          Property[{1}]: has dimension
                                           error\n'.\
                                            format(root.get('Name'),
                                             prop.get('Name')))
                                elif is_number(myVal) and basicAtt1=='String':
                                    propval += 1;
                                    warningcheck += 1;
                                    fileoutput.write('WARNING: Component[{0}]:
                                     Primitive Property[{1}]: has real data and
                                      listed as string'.\
                                        format(root.get('Name'), prop.get('Name')))

        propnames2.append(prop.get('Name'))
    return [warningcheck,errorcheck,propval,propnames2, concheck] ##propval



def check_properties(xmldoc, statsData, fileoutput):  # checks that the
     resource link in the XML file are valid
    """Return true or false based on whether the resource objects in the
       XMl point to valid files"""
    root = xmldoc.getroot()


    props = root.findall("Property")
    propnames = [p.get('Name') for p in props]
    if len(propnames) != len(set(propnames)):
        warningcheck += 1;
        print 'ERROR: Component[{0}]: duplicate property
         names'.format(root.get('Name'))

    propval = 0;
    warningcheck = 0;
    errorcheck = 0;
    concheck = 0;

    propnames2 = list()

    for child in props:
        notes = check_property(child, child.get(XSI + 'type'), root,
         propval, propnames2, errorcheck, warningcheck, fileoutput,
          concheck)
        propval = notes[2];
        errorcheck = notes[1];
        warningcheck = notes[0];
        propnames2 = notes[3];
        concheck = notes[4];


    statsData.append(len(props))                                    ##
```

```
     number of property errors in file
    print 'number of property errors =', propval
    statsData.append(propval)
    ##print True

    propval2 = len(propnames2) - len(set(propnames2))
    statsData.append(propval2)
    statsData.append(errorcheck)
    statsData.append(warningcheck)
    statsData.append(concheck)

    return True


def check_connectors(xmldoc, statsData):  # checks that the resource link
 in the XML file are valid
    """Return true or false based on whether the resource objects in the
       XMl point to valid files"""
    root = xmldoc.getroot()
    conns = root.findall("Connector")
    count = 0

    if (conns is None) or (len(conns) == 0):
        print 'WARNING: Component[{0}: No
         connectors'.format(root.get('Name'))
    else:
        names = [c.get('Name') for c in conns]
        if len(names) != len(set(names)):
            print 'ERROR: Component[{0}]: duplicate connector names'.\
                format(root.get('Name'))

        for child in conns:
            roles = child.findall('Role')
            if (roles is None) or (len(roles) == 0):
                print 'WARNING: Component[{0}]: Connector[{1}] has no
                 content'.format(root.get('Name'), child.get('Name'))
            for r in roles:
                source = r.attrib.get('PortMap')
                if 'cad' in source:
                    count += 1

    statsData.append(len(conns))
    statsData.append(count)
    return True


def parse_args():    # parses the command line arguments
    """Returns a list of program arguments"""
    parser = OptionParser()
    parser.add_option("-p", "--path", dest="path",
                  help="path to directory containing component XML's",
                   metavar="PATH")
    parser.add_option("-x", "--schema", dest="xsd",
                  help="path to the component XML schema", metavar="SCHEMA")
```

```
    parser.add_option("-q", "--quiet",
                  action="store_false", dest="verbose", default=True,
                  help="don't print status messages to stdout")

    (options, args) = parser.parse_args()
    if options.path is None:
        parser.error("missing required path argument")
        return None
    if options.xsd is None:
        parser.error("missing required schema argument")
        return None

    print 'XML Path: {0}'.format(options.path)
    print 'Schema File: {0}'.format(options.xsd)

    return options



def main(argv=sys.argv):
    options = parse_args()
    compxmls = list_cxmls(options.path)
    stats = list()
    filecount=0;
    fileoutput = open('Python_Log.txt','w')

    for x in compxmls:
        # parse the XML and validate
        statsData = list()
        xmldoc = schema_validate(x, options.xsd)
        if xmldoc is None:
            print 'ERROR: Component XML[{0}]: failed schema
             validation'.format(x)
            statsData.extend([x, '<a href="' + x + '">XML File</a>', False,
             None, False, False, False, False, 0, 0])  # Schema invalid
        else:
            xmldoc = etree.parse(x)
            statsData.extend([xmldoc.getroot().get('Name'), '<a href="' + x
             + '">XML File</a>', True])  # Schema valid
            check_classification(xmldoc, statsData)
            check_resources(xmldoc, options.path, statsData)
            check_domain_models(xmldoc, statsData)
            check_properties(xmldoc, statsData, fileoutput)
            ##check_parameters(xmldoc, statsData)
            check_connectors(xmldoc, statsData)
            #print len(statsData)
        stats.append(statsData)


    fileoutput.close()

    headers = ['Component', 'XML', 'Schema Valid', 'Classification',
     'Resources Valid', 'Has CAD',
```

```
                'Has Modelica', 'Has Manufacturing', 'Num Properties',
                 'Empty Properties','Duplicate Names' ,'Property
                  Errors','Property Warnings','CAD Derived Properties','Num
                   Connectors','CAD References in Connectors']
    tld = tablib.Dataset(*stats, headers=headers)
    tld2 = tablib.Dataset(headers=["Category", "Instances"])
    for k,v in catCounts.iteritems():
        tld2.append([k,v])

    stats_file = open("stats.html", "w")
    stats_file.write("<html>")
    stats_file.write('<!-- DataTables CSS --> \
<link rel="stylesheet" type="text/css"
 href="http://ajax.aspnetcdn.com/ajax/jquery.dataTables/1.9.4/css/jquery.dat
  aTables.css"> \
 \
<!-- jQuery --> \
<script type="text/javascript" charset="utf8"
 src="http://ajax.aspnetcdn.com/ajax/jQuery/jquery-1.8.2.min.js"></script> \
 \
<!-- DataTables --> \
<script type="text/javascript" charset="utf8"
 src="http://ajax.aspnetcdn.com/ajax/jquery.dataTables/1.9.4/jquery.dataTabl
  es.min.js"></script> \
\
<script type="text/javascript"> \
$(document).ready(function() { \
    $("table").dataTable(); \
} ); \
</script> \
')

    stats_file.write("<body>")
    stats_file.write('<p style="font-weight: bold;">Component Instance
     Report</p>')
    stats_file.write('<div id="component_table">')
    stats_file.write(tld.html)
    stats_file.write('</div>')
    stats_file.write('<div class="clear"/><br><br>')
    stats_file.write('<p style="font-weight: bold;">Category Count
     Report</p>')
    stats_file.write('<div id="category_table">')
    stats_file.write(tld2.html)
    stats_file.write('</div>')
    stats_file.write('<div class="clear"/>')
    stats_file.write("</body>")
    stats_file.write("</html>")
    stats_file.close()

    return 0

if __name__ == "__main__":
    sys.exit(main())
```

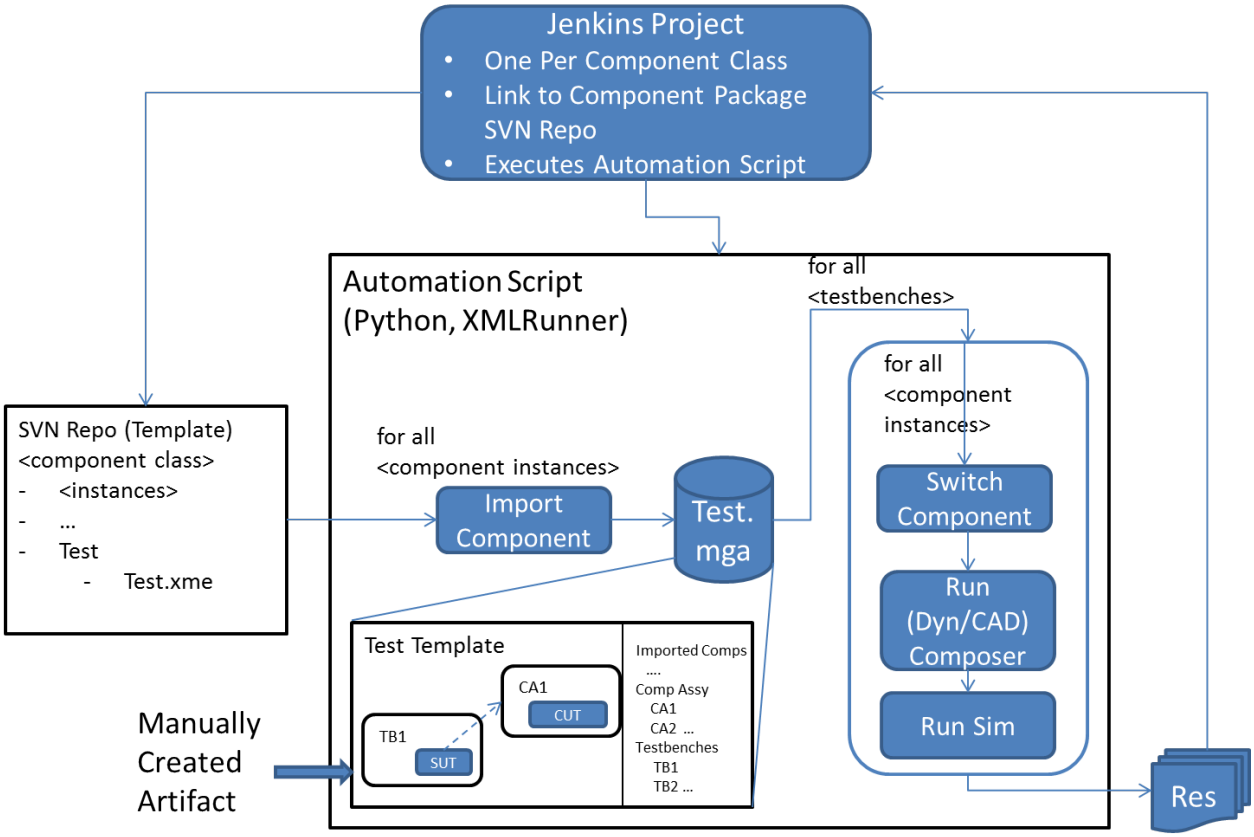

Figure 4.1: Unit Testing Framework

4.3 Component Class-based Unit Tests

An automated unit testing framework performs in-class conformance check on components. As depicted in figure 4.1 the unit-test framework automatically instantiates a component in a testbench model constructed apriori in META tools. The framework executes CyPhy model composer tools, and executes the generated artifacts (Modelica simulation, CAD Constraint engine, etc.). The results of the tests are post processed to determine if the component satisfies class-specific conformance requirements specified through the unit tests.

Chapter 5

Reference Model

5.1 Component Descriptor

5.2 Package Contents

Chapter 6

Formal Semantics of AVM Component Model

6.1 Domains

6.1.1 AVM Interchange Format Overview Semantics

File AVMInterchangeOverview.4ml
Uses AVMInterchange at "AVMInterchange.4ml"
Extends AVMInterchange

6.1.1.1 Constraints

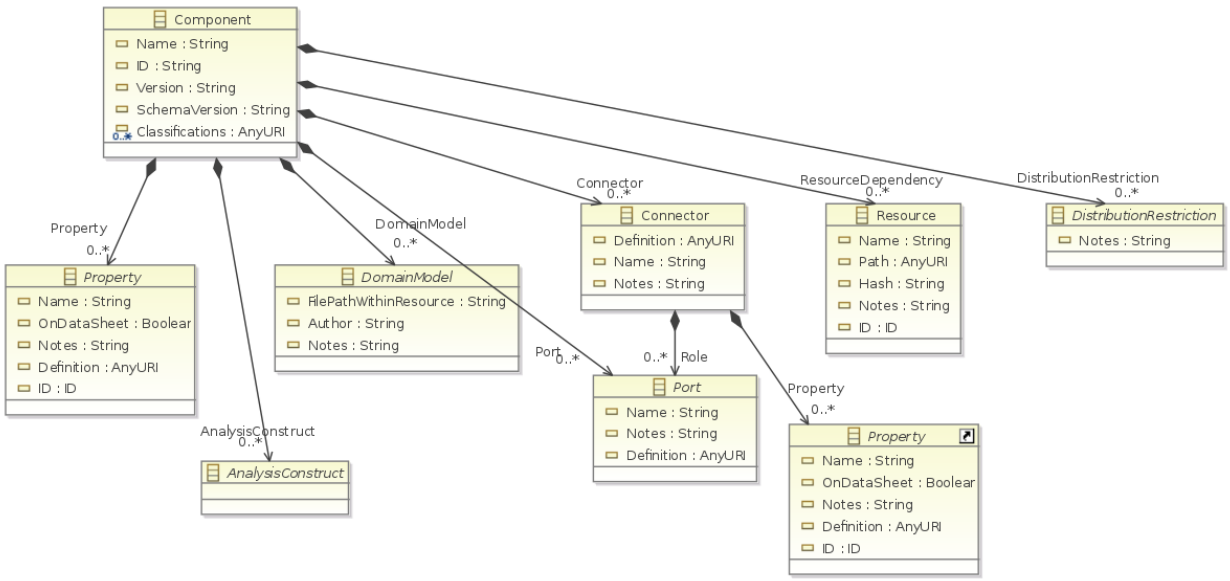

Figure 6.1: Component Class Diagram

Component Diagram Abstract Types

```
avm_Property ::=
```

```
    avm_CompoundProperty +
    avm_PrimitiveProperty.
  avm_AnalysisConstruct ::=
    avm_cad_Geometry +
    avm_cad_CustomGeometry +
    avm_cad_Geometry3D +
    avm_cad_Surface +
    avm_cad_Sphere +
    avm_cad_ExtrudedGeometry +
    avm_cad_Geometry2D +
    avm_cad_Polygon +
    avm_cad_Circle.
  avm_DomainModel ::=
    avm_simulink_SimulinkModel +
    avm_manufacturing_ManufacturingModel +
    avm_cad_CADModel +
    avm_modelica_ModelicaModel.
  avm_Port ::=
    avm_DomainModelPort +
    avm_simulink_Port1 +
    avm_simulink_Signal +
    avm_simulink_OutputSignal +
    avm_simulink_InputSignal +
    avm_AbstractPort +
    avm_cad_Datum +
    avm_cad_CoordinateSystem +
    avm_cad_Plane +
    avm_cad_Axis +
    avm_cad_Point +
    avm_modelica_Connector.
  avm_DistributionRestriction ::=
    avm_ITAR +
    avm_Proprietary +
    avm_SecurityClassification.
```

Component Diagram Concrete Types

```
avm_Component ::= new (
    aObjectId: Integer,
    ID: String,
    Name: String,
    SchemaVersion: String,
    Version: String
  ).
  avm_Connector ::= new (
    aObjectId: Integer,
    Definition: String,
    ID: String,
    Name: String,
    Notes: String
  ).
  avm_Resource ::= new (
    aObjectId: Integer,
    Hash: String,
    ID: String,
    Name: String,
    Notes: String,
    Path : String
  ).
```

Component Diagram Relationships

```
avm_Component2Property ::= new (
    Property: avm_Component,
    Dst: avm_Property
  ).
  avm_AnalysisConstruct2Component ::= new (
```

```
  Src: avm_AnalysisConstruct,
  AnalysisConstruct: avm_Component
).
avm_Component2DomainModel ::= new (
  DomainModel: avm_Component,
  Dst: avm_DomainModel).
avm_Component2Port ::= new (
  Port: avm_Component,
  Dst: avm_Port
).
avm_Component2Connector ::= new (
  Connector: avm_Component,
  Dst: avm_Connector
).
avm_Component2Resource ::= new (
  ResourceDependency: avm_Component,
  Dst: avm_Resource
).
avm_Component2DistributionRestriction ::= new (
  DistributionRestriction: avm_Component,
  Dst: avm_DistributionRestriction
).
avm_Connector2Port ::= new (
  Role: avm_Connector,
  Dst: avm_Port
).
avm_Connector2Property ::= new (
  Property: avm_Connector,
  Dst: avm_Property
).
```

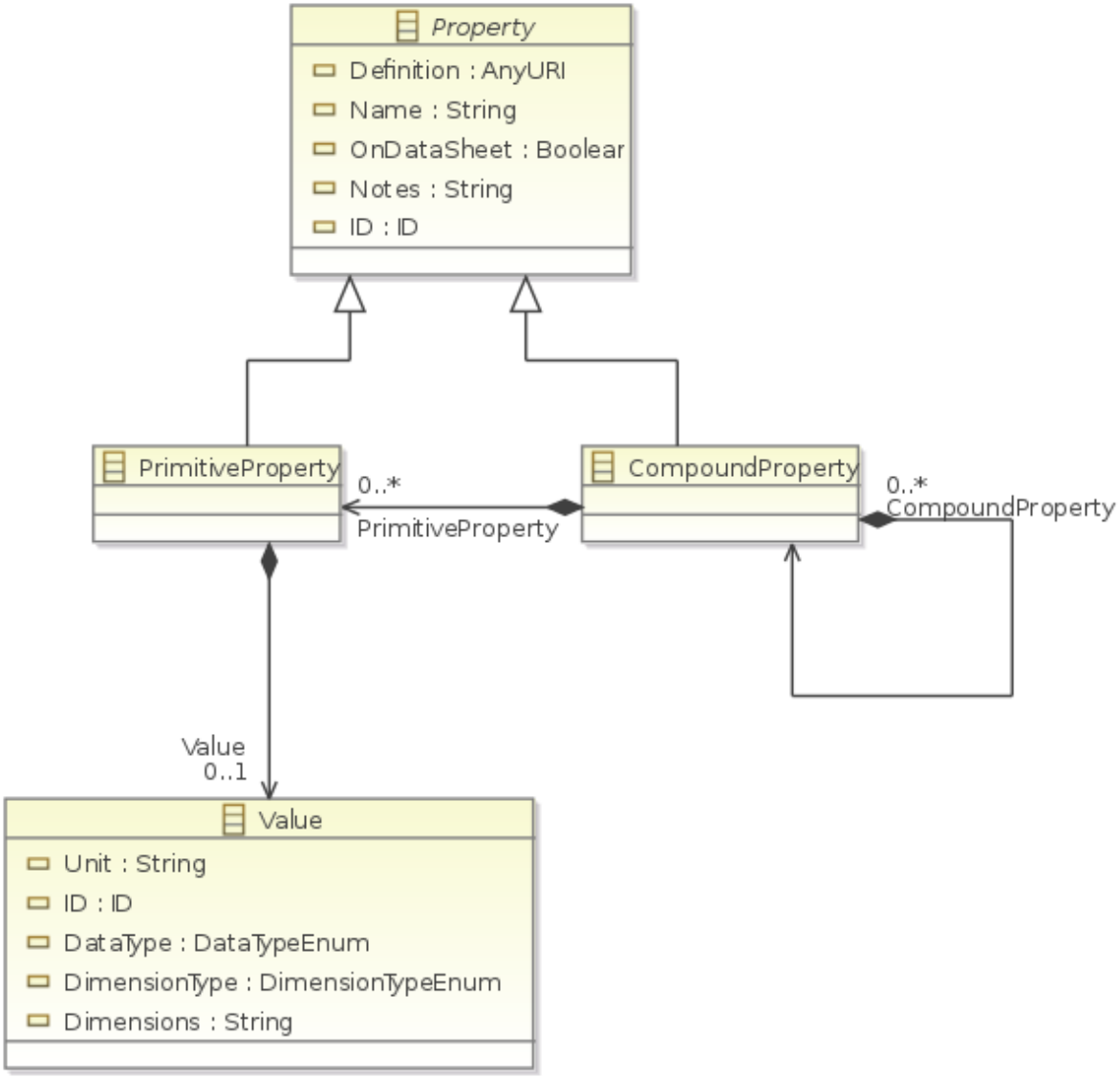

Figure 6.2: Property Class Diagram

Property Diagram Concrete Types

```
avm_CompoundProperty ::= new (
    aObjectId: Integer,
    Definition: String,
    ID: String,
    Name: String,
    Notes: String,
    OnDataSheet: Boolean,
    OnDataSheetSpecified: Boolean
  ).
  avm_PrimitiveProperty ::= new (
    aObjectId: Integer,
    Definition: String,
    ID: String,
    Name: String,
```

```
    Notes: String,
    OnDataSheet: Boolean,
    OnDataSheetSpecified: Boolean
  ).
  avm_Value ::= new (
    aObjectId: Integer,
    DataType: DataTypeEnum,
    DataTypeSpecified: Boolean,
    Dimensions: String,
    DimensionType: DimensionTypeEnum,
    DimensionTypeSpecified: Boolean,
    ID: String,
    Unit: String
  ).
```

Property Diagram Relationships

```
avm_PrimitiveProperty2Value ::= new (
    Value: avm_PrimitiveProperty,
    Dst: avm_Value
  ).
  avm_CompoundProperty2PrimitiveProperty ::= new (
    PrimitiveProperty: avm_CompoundProperty,
    Dst: avm_PrimitiveProperty
  ).
  avm_CompoundProperty2CompoundProperty ::= new (
    CompoundProperty1: avm_CompoundProperty,
    Dst: avm_CompoundProperty
  ).
```

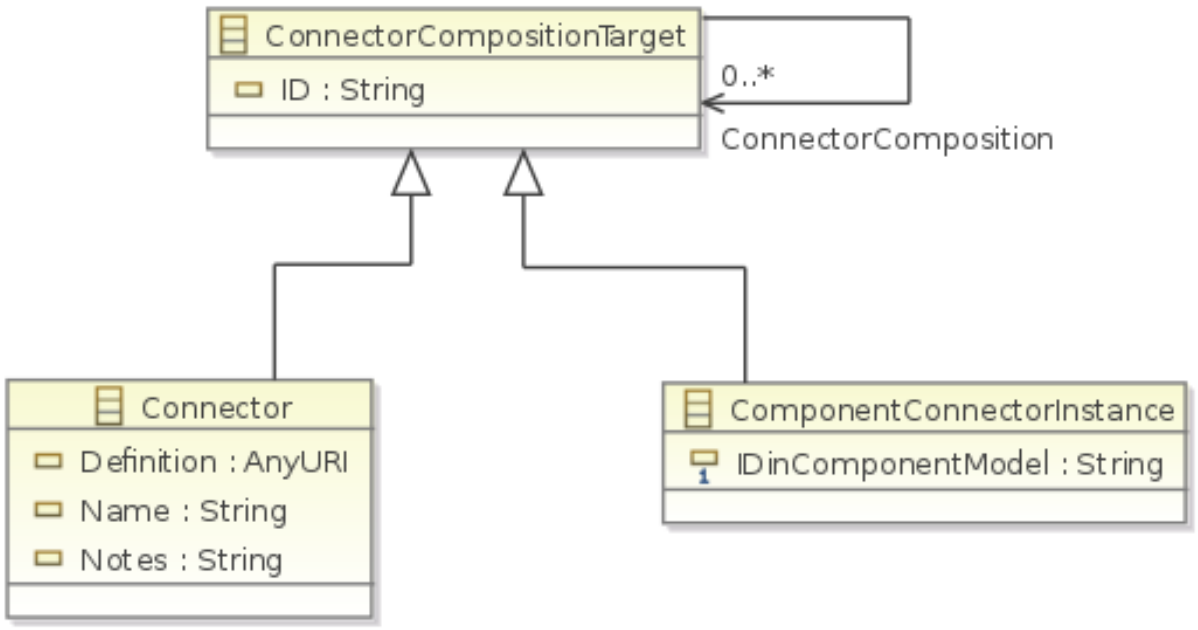

Figure 6.3: Connector Class Diagram

Connector Diagram Concrete Types

```
avm_ConnectorCompositionTarget ::= new (
```

```
    aObjectId: Integer,
    ID: String
  ).
  avm_ComponentConnectorInstance ::= new (
    aObjectId: Integer,
    ID: String,
    IDinComponentModel: String
  ).
```

Connector Diagram Relationships

```
avm_ConnectorCompositionTarget2StringSrc ::=
    avm_ConnectorCompositionTarget +
    avm_Connector +
    avm_ComponentConnectorInstance.
  avm_ConnectorCompositionTarget2String ::= new (
    ConnectorComposition: avm_ConnectorCompositionTarget2StringSrc,
    Dst: String //ID of a ConnectorCompositionTarget
  ).
```

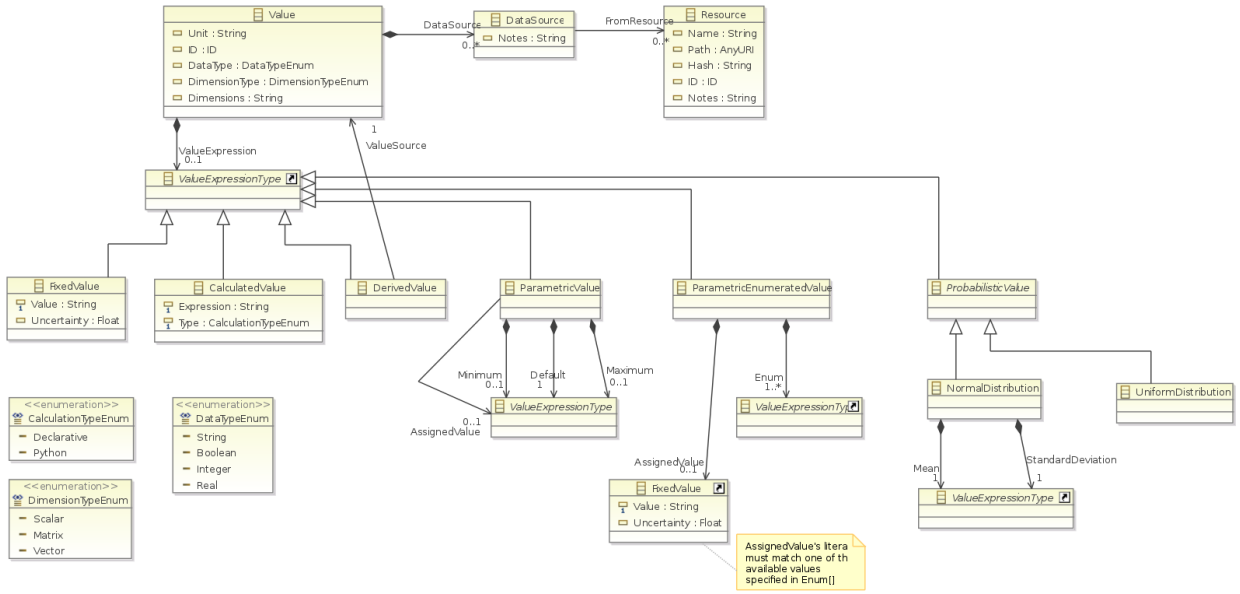

Figure 6.4: Value Interchange Class Diagram

Value Diagram Enumerated Types

```
CalculationTypeEnum ::= {
    "Declarative",
    "Python"
  }.
  DataTypeEnum ::= {
    "String",
    "Boolean",
    "Integer",
    "Real"
  }.
  DimensionTypeEnum ::= {
    "Matrix",
    "Vector",
    "Scalar"
  }.
```

Value Diagram Abstract Types

```
avm_ValueExpressionType ::=
    avm_ParametricEnumeratedValue +
    avm_FixedValue +
    avm_ProbabilisticValue +
    avm_UniformDistribution +
    avm_NormalDistribution +
    avm_ParametricValue +
    avm_DerivedValue +
    avm_CalculatedValue.
  avm_ProbabilisticValue ::=
    avm_UniformDistribution +
    avm_NormalDistribution.
```

Value Diagram Concrete Types

```
avm_DataSource ::= new (
    aObjectId: Integer,
    Notes: String
  ).
  avm_FixedValue ::= new (
    aObjectId: Integer,
    Uncertainty: Real,
    UncertaintySpecified: Boolean,
    Value: String
  ).
  avm_CalculatedValue ::= new (
    aObjectId: Integer,
    Expression: String,
    Type: CalculationTypeEnum
  ).
  avm_DerivedValue ::= new (
    aObjectId: Integer,
    ValueSource: String //Encodes relationship with Value, refers to ID attribute
  ).
  avm_ParametricValue ::= new (
    aObjectId: Integer
  ).
  avm_ParametricEnumeratedValue ::= new (
    aObjectId: Integer
  ).
  avm_UniformDistribution ::= new (
    aObjectId: Integer
  ).
  avm_NormalDistribution ::= new (
    aObjectId: Integer
  ).
```

Value Diagram Relationships

```
avm_Value2ValueExpressionType ::= new (
    ValueExpression: avm_Value,
    Dst: avm_ValueExpressionType
  ).
  avm_ParametricValue2ValueExpressionType ::= new (
    Default: avm_ParametricValue,
    Dst: avm_ValueExpressionType //Default
  ).
  avm_ParametricValue2ValueExpressionType1 ::= new (
    Maximum: avm_ParametricValue,
    Dst: avm_ValueExpressionType //Maximum
  ).
  avm_ParametricValue2ValueExpressionType2 ::= new (
    Minimum: avm_ParametricValue,
    Dst: avm_ValueExpressionType //Minimum
  ).
```

```
avm_ParametricValue2ValueExpressionType3 ::= new (
  AssignedValue: avm_ParametricValue,
  Dst: avm_ValueExpressionType //AssignedValue
).
avm_DataSource2Value ::= new (
  Src: avm_DataSource,
  DataSource: avm_Value
).
avm_DataSource2String ::= new (
  FromResource: avm_DataSource,
  Dst: String //ID of Resource
).
avm_FixedValue2ParametricEnumeratedValue ::= new (
  Src: avm_FixedValue,
  AssignedValue: avm_ParametricEnumeratedValue
).
avm_ParametricEnumeratedValue2ValueExpressionType ::= new (
  Enum: avm_ParametricEnumeratedValue,
  Dst: avm_ValueExpressionType
).
avm_NormalDistribution2ValueExpressionType ::= new (
  Mean: avm_NormalDistribution,
  Dst: avm_ValueExpressionType //Mean
).
avm_NormalDistribution2ValueExpressionType1 ::= new (
  StandardDeviation: avm_NormalDistribution,
  Dst: avm_ValueExpressionType //StandardDeviation
).
```

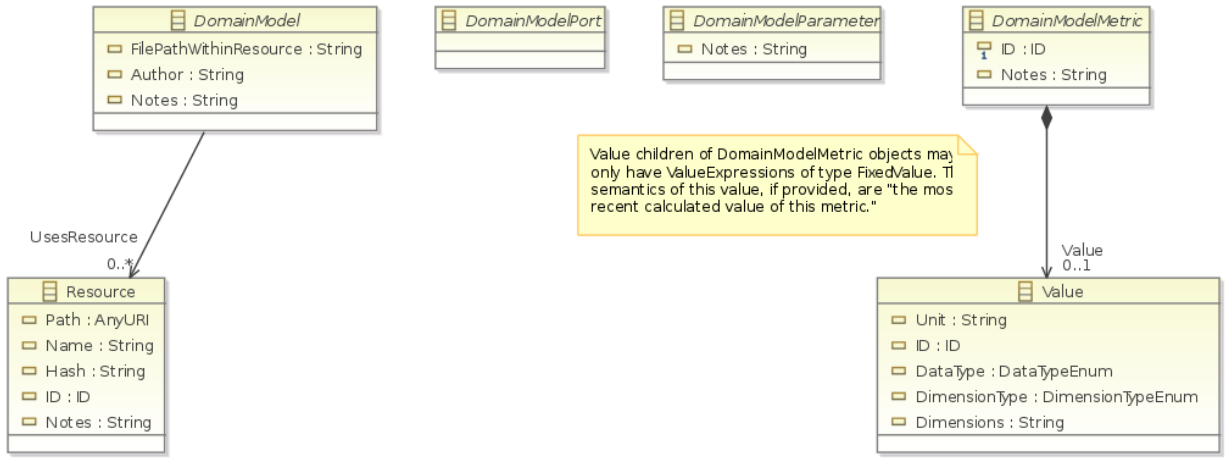

Figure 6.5: Domain Model Class Diagram

TODO: Add description of domain model diagram

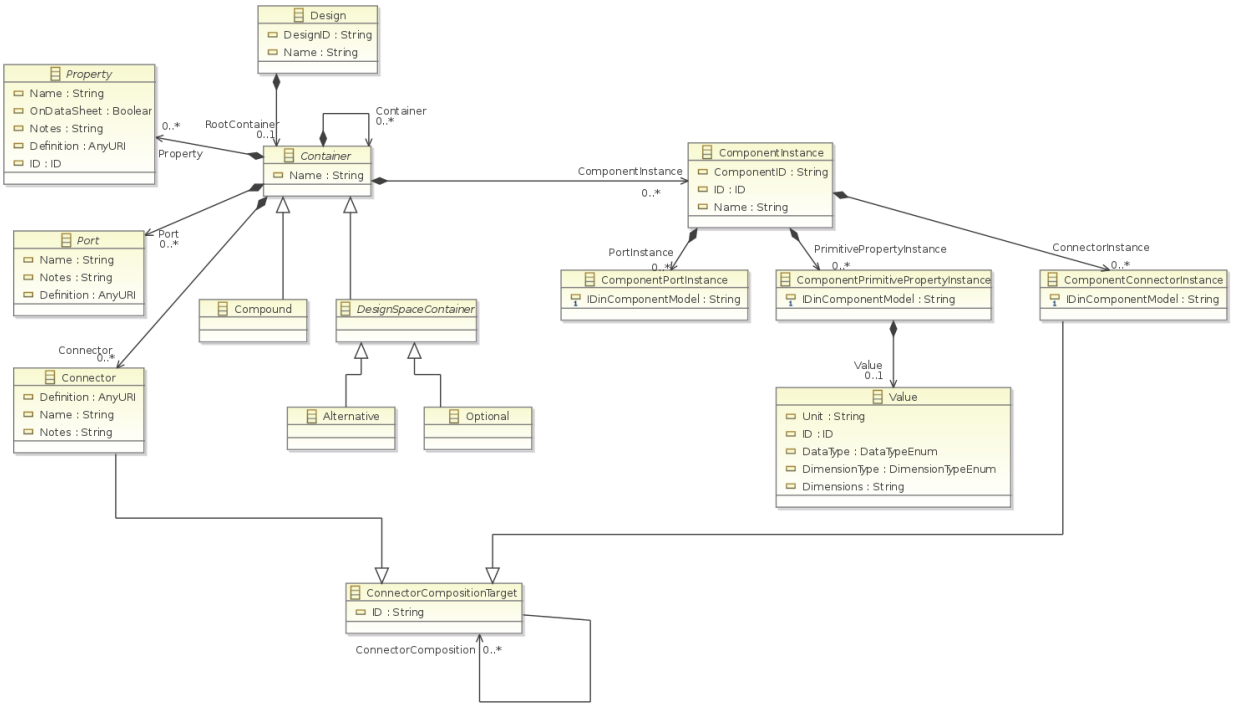

Figure 6.6: Design Interchange Class Diagram

Design Diagram Abstract Types

```
avm_Container ::=
    avm_DesignSpaceContainer +
    avm_Alternative +
    avm_Optional +
    avm_Compound.
  avm_DesignSpaceContainer ::=
    avm_Alternative +
    avm_Optional.
```

Design Diagram Concrete Types

```
avm_Design ::= new (
    aObjectId: Integer,
    DesignID: String,
    Name: String
  ).
  avm_Compound ::= new (
    aObjectId: Integer,
    Name: String
  ).
  avm_Alternative ::= new (
    aObjectId: Integer,
    Name: String
  ).
  avm_Optional ::= new (
    aObjectId: Integer,
    Name: String
  ).
  avm_ComponentInstance ::= new (
    aObjectId: Integer,
    ComponentID: String,
```

```
    ID: String,
    Name: String
  ).
  avm_ComponentPortInstance ::= new (
    aObjectId: Integer,
    ID: String,
    IDinComponentModel: String
  ).
  avm_ComponentPrimitivePropertyInstance ::= new (
    aObjectId: Integer,
    IDinComponentModel: String
  ).
```

Design Diagram Relationships

```
avm_Container2Design ::= new (
    Src: avm_Container,
    RootContainer: avm_Design
  ).
  avm_Container2Port ::= new (
    Port: avm_Container,
    Dst: avm_Port
  ).
  avm_Container2Property ::= new (
    Property: avm_Container,
    Dst: avm_Property
  ).
  avm_Connector2Container ::= new (
    Src: avm_Connector,
    Connector: avm_Container
  ).
  avm_Container2Container ::= new (
    Container1: avm_Container,
    Dst: avm_Container
  ).
  avm_ComponentInstance2Container ::= new (
    Src: avm_ComponentInstance,
    ComponentInstance: avm_Container
  ).
  avm_ComponentInstance2ComponentPortInstance ::= new (
    PortInstance: avm_ComponentInstance,
    Dst: avm_ComponentPortInstance
  ).
  avm_ComponentInstance2ComponentPrimitivePropertyInstance ::= new (
    PrimitivePropertyInstance: avm_ComponentInstance,
    Dst: avm_ComponentPrimitivePropertyInstance
  ).
  avm_ComponentConnectorInstance2ComponentInstance ::= new (
    Src: avm_ComponentConnectorInstance,
    ConnectorInstance: avm_ComponentInstance
  ).
```

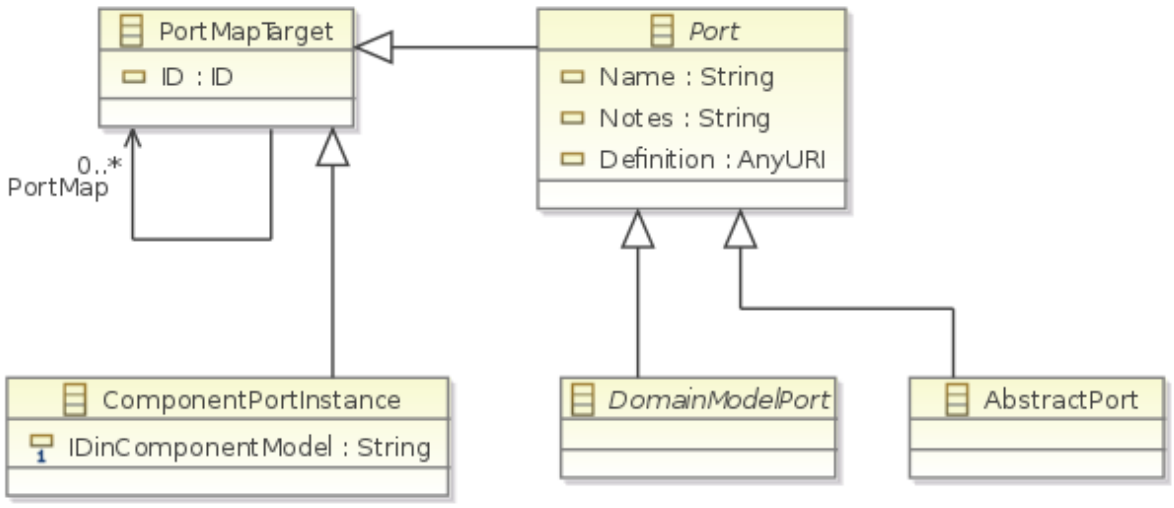

Figure 6.7: Port Class Diagram

Port Diagram Elements

```
avm_DomainModelPort ::=
    avm_simulink_Port1 +
    avm_simulink_Signal +
    avm_simulink_OutputSignal +
    avm_simulink_InputSignal +
    avm_cad_Datum +
    avm_cad_CoordinateSystem +
    avm_cad_Plane +
    avm_cad_Axis +
    avm_cad_Point +
    avm_modelica_Connector.
  avm_PortMapTarget ::= new (
    aObjectId: Integer,
    ID: String
  ).
  avm_AbstractPort ::= new (
    aObjectId: Integer,
    Definition: String,
    ID: String,
    Name: String,
    Notes: String
  ).
  avm_PortMapTarget2StringSrc ::=
    avm_PortMapTarget +
    avm_AbstractPort +
    avm_ComponentPortInstance +
    avm_cad_Point +
    avm_cad_Axis +
    avm_cad_Plane +
    avm_cad_CoordinateSystem +
    avm_modelica_Connector +
    avm_simulink_InputSignal +
    avm_simulink_OutputSignal.
  avm_PortMapTarget2String ::= new (
    PortMap: avm_PortMapTarget2StringSrc,
    Dst: String //ID of the other PortMapTarget
  ).
```

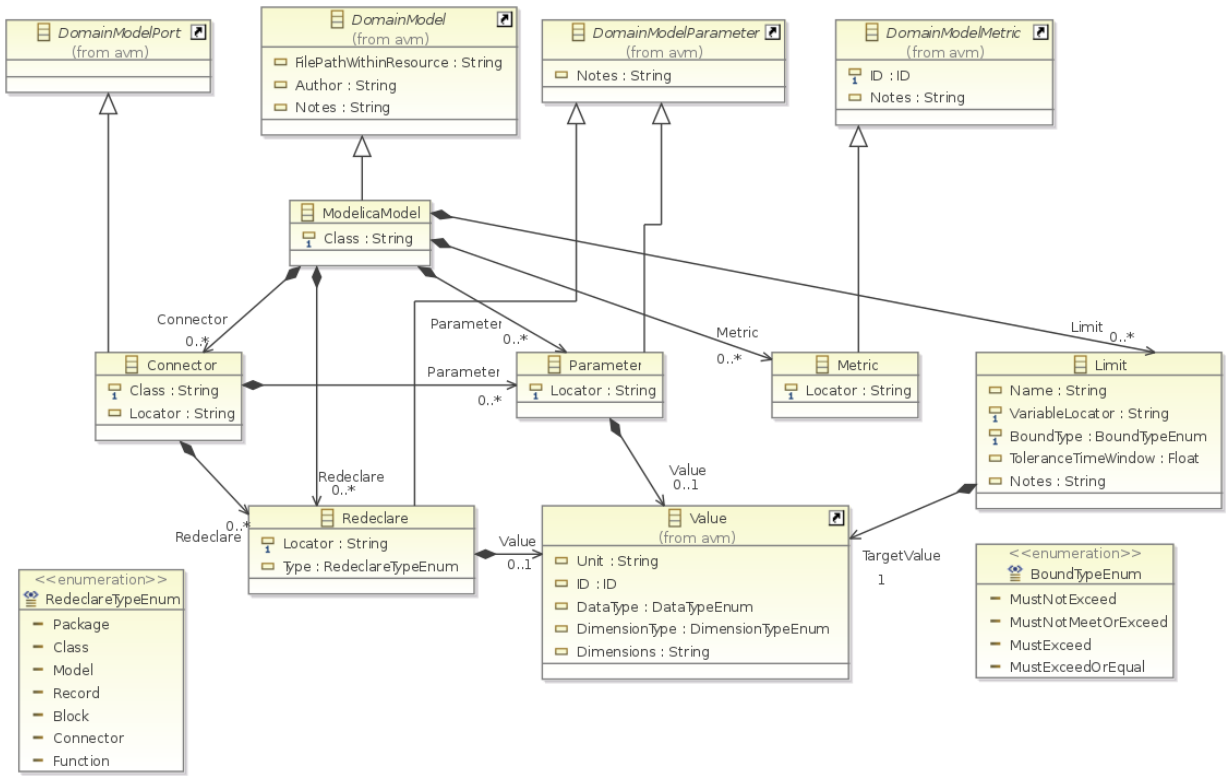

Figure 6.8: Modelica Domain Model Class Diagram

Modelica Diagram Elements

```
RedeclareTypeEnum ::= {
    "Package",
    "Class",
    "Model",
    "Record",
    "Block",
    "Connector",
    "Function"
  }.
  BoundTypeEnum ::= {
    "MustNotExceed",
    "MustNotMeetOrExceed",
    "MustExceed",
    "MustExceedOrEqual"
  }.
  avm_modelica_ModelicaModel ::= new (
    aObjectId: Integer,
    Author: String,
    Class: String,
    FilePathWithinResource: String,
    Notes: String,
    UsesResource: String).
  avm_modelica_Connector ::= new (
    aObjectId: Integer,
    Class: String,
    Definition: String,
    ID: String,
    Locator: String,
    Name: String,
```

```
  Notes: String
).
avm_modelica_Parameter ::= new (
  aObjectId: Integer,
  Locator: String,
  Notes: String
).
avm_modelica_Metric ::= new (
  aObjectId: Integer,
  ID: String,
  Locator: String,
  Notes: String
).
avm_modelica_Limit ::= new (
  aObjectId: Integer,
  BoundType: BoundTypeEnum,
  Name: String,
  Notes: String,
  ToleranceTimeWindow: Real,
  ToleranceTimeWindowSpecified: Boolean,
  VariableLocator: String
).
avm_modelica_Redeclare ::= new (
  aObjectId: Integer,
  Locator: String,
  Notes: String,
  Type: RedeclareTypeEnum,
  TypeSpecified: Boolean
).
avm_modelica_Connector2ModelicaModel ::= new (
  Src: avm_modelica_Connector,
  Connector: avm_modelica_ModelicaModel
).
avm_modelica_ModelicaModel2Redeclare ::= new (
  Redeclare: avm_modelica_ModelicaModel,
  Dst: avm_modelica_Redeclare
).
avm_modelica_ModelicaModel2Parameter ::= new (
  Parameter: avm_modelica_ModelicaModel,
  Dst: avm_modelica_Parameter
).
avm_modelica_Metric2ModelicaModel ::= new (
  Src: avm_modelica_Metric,
  Metric: avm_modelica_ModelicaModel
).
avm_modelica_Limit2ModelicaModel ::= new (
  Src: avm_modelica_Limit,
  Limit: avm_modelica_ModelicaModel
).
avm_modelica_Connector2Parameter ::= new (
  Parameter: avm_modelica_Connector,
  Dst: avm_modelica_Parameter
).
avm_modelica_Connector2Redeclare ::= new (
  Redeclare: avm_modelica_Connector,
  Dst: avm_modelica_Redeclare
).
avm_modelica_Redeclare2Value ::= new (
  Value: avm_modelica_Redeclare,
  Dst: avm_Value
).
avm_modelica_Parameter2Value ::= new (
  Value: avm_modelica_Parameter,
  Dst: avm_Value
).
avm_modelica_Limit2Value ::= new (
  TargetValue: avm_modelica_Limit,
  Dst: avm_Value
```

```
).
```

6.1.2 AVM Interchange Format Structural Semantics

File AVMInterchangeStructural.4ml
Uses AVMInterchange at "AVMInterchange.4ml"
Extends AVMInterchange

6.1.2.1 Constraints

Reference relationships in the interchange format are handled by relating the source object ot the ID of its target.These relations are invalid if there is no object of the appropriate type with the referenced ID.

```
InvalidIDRef :-
    avm_cad_PlaneReference2String(_, id), no avm_cad_Plane(_, _, _, id, _, _);
    avm_ConnectorCompositionTarget2String(_, id), no avm_ConnectorCompositionTarget(_,
        id), no avm_Connector(_, _, id, _, _), no avm_ComponentConnectorInstance(_, id,
        _);
    avm_PortMapTarget2String(_, id), id != "", no avm_AbstractPort(_, _, id, _, _), no
        avm_cad_Axis(_, _, _, id, _, _), no avm_cad_CoordinateSystem(_, _, _, id, _, _),
        no avm_cad_Plane(_, _, _, id, _, _), no avm_modelica_Connector(_, _, _, id, _,
        _, _), no avm_ComponentPortInstance(_, id, _);
    avm_DerivedValue(_, id), no avm_Value(_, _, _, _, _, _, id, _).
```

The ComponentID attribute of an avm.ComponentInstance must refer to an existing avm.Component.

```
ComponentInstanceInvalidComponentID :-
    avm_ComponentInstance(_, compID, _, _), no avm_Component(_, compID, _, _, _).
```

6.1.3 Helper Rules for Interchange Domain

File AVMInterchangeHelpful.4ml
Uses AVMInterchange at "AVMInterchange.4ml"
Extends AVMInterchange

6.1.3.1 Constraints

The AVMInterchangeHelpful domain extends the original generated AVMInterchange domain with severalrules for aggregation (e.g. of the notion of containment) and simplification. Relates and ID-possessing element from the interchange domain to its ID.

```
HasID ::= (AVMInterchange.Data, String).
  HasID(obj, id) :-
    obj is avm_PortMapTarget(_, id);
    obj is avm_simulink_OutputSignal(_, _, _, id, _, _, _);
    obj is avm_simulink_InputSignal(_, _, _, id, _, _, _);
    obj is avm_Component(_, id, _, _, _);
    obj is avm_ComponentPortInstance(_, id, _);
    obj is avm_AbstractPort(_, _, id, _, _);
    obj is avm_ConnectorCompositionTarget(_, id);
    obj is avm_ComponentConnectorInstance(_, id, _);
    obj is avm_Connector(_, _, id, _, _);
    obj is avm_CompoundProperty(_, _, id, _, _, _, _);
    obj is avm_PrimitiveProperty(_, _, id, _, _, _, _);
    obj is avm_Value(_, _, _, _, _, _, id, _);
    obj is avm_Resource(_, _, id, _, _, _);
    obj is avm_ComponentInstance(_, _, id, _);
    obj is avm_manufacturing_Metric(_, id, _, _);
    obj is avm_cad_Metric(_, id, _, _);
    obj is avm_cad_CoordinateSystem(_, _, _, id, _, _);
```

```
    obj is avm_cad_Plane(_, _, _, id, _, _);
    obj is avm_cad_Axis(_, _, _, id, _, _);
    obj is avm_cad_Point(_, _, _, id, _, _);
    obj is avm_modelica_Metric(_, id, _, _);
    obj is avm_modelica_Connector(_, _, _, id, _, _, _).
```

Serves as a polymorphic accessor for avm_Container concrete subtypes and converts their respective typesinto a string, which corresponds to the way it is represented in CyPhyML.

```
AVMDesignContainer ::= (container:avm_Container, aObjectID:Integer, Name:String,
    Type:String).
  AVMDesignContainer(cont, aID, name, type) :-
    cont is avm_Compound(aID, name), type = "Compound";
    cont is avm_Alternative(aID, name), type = "Alternative";
    cont is avm_Optional(aID, name), type = "Optional".
```

Serves as a polymorphic accessor for avm_Port concrete subtypes.

```
AVMPort ::= (port:avm_Port, aObjectID:Integer, ID:String, Name:String, Notes:String,
    Definition:String).
  AVMPort(port, aID, id, name, notes, def) :-
    port is avm_AbstractPort(aID, def, id, name, notes);
    port is avm_modelica_Connector(aID, _, def, id, _, name, notes);
    port is avm_cad_Axis(aID, _, def, id, name, notes);
    port is avm_cad_Point(aID, _, def, id, name, notes);
    port is avm_cad_Plane(aID, _, def, id, name, notes);
    port is avm_cad_CoordinateSystem(aID, _, def, id, name, notes);
    port is avm_simulink_InputSignal(aID, _, def, id, _, name, notes);
    port is avm_simulink_OutputSignal(aID, _, def, id, _, name, notes).
```

Aggregates all containment relationships from the Interchange domain into a single easy to use andunderstand 'Contains' relationship.

```
Contains ::= (AVMInterchange.Data, AVMInterchange.Data).
  Contains(a, b) :-
    avm_cad_ExtrudedGeometry2Geometry2D(a, b);
    avm_cad_CustomGeometryInput2Geometry(a, b);
    avm_cad_CustomGeometry2CustomGeometryInput(a, b);
    avm_cad_Circle2PointReference(a, b); //Circle -> CircleCenter
    avm_cad_Circle2PointReference1(a, b); //Circle -> CircleEdge
    avm_cad_PointReference2Polygon(b, a);
    avm_cad_ExtrudedGeometry2PointReference(a, b);
    avm_cad_PointReference2Sphere(b, a); //SphereCenter <- Sphere
    avm_cad_PointReference2Sphere1(b, a); //SphereEdge <- Sphere
    avm_cad_PlaneReference2Surface(b, a);
    avm_cad_CADModel2Metric(a, b);
    avm_cad_CADModel2Parameter(a, b);
    avm_cad_CADModel2Datum(a, b);
    avm_cad_Datum2Metric(a, b);
    avm_cad_Parameter2Value(a, b);
    avm_Component2Property(a, b);
    avm_AnalysisConstruct2Component(b, a);
    avm_Component2DomainModel(a, b);
    avm_Component2Connector(a, b);
    avm_Component2Port(a, b);
    avm_Component2Resource(a, b);
    avm_Component2DistributionRestriction(a, b);
    avm_Connector2Port(a, b);
    avm_Connector2Property(a, b);
    avm_Container2Container(a, b);
    avm_Container2Design(b, a);
    avm_Container2Property(a, b);
    avm_Container2Port(a, b);
    avm_Connector2Container(b, a);
    avm_ComponentInstance2ComponentPortInstance(a, b);
    avm_ComponentInstance2ComponentPrimitivePropertyInstance(a, b);
```

```
    avm_ComponentConnectorInstance2ComponentInstance(b, a);
    avm_ComponentInstance2Container(b, a);
    avm_ComponentPrimitivePropertyInstance2Value(a, b);
    avm_DomainModelMetric2Value(a, b);
    avm_manufacturing_ManufacturingModel2Metric(a, b);
    avm_manufacturing_ManufacturingModel2Parameter(a, b);
    avm_manufacturing_Parameter2Value(a, b);
    avm_modelica_Connector2ModelicaModel(b, a);
    avm_modelica_ModelicaModel2Redeclare(a, b);
    avm_modelica_ModelicaModel2Parameter(a, b);
    avm_modelica_Metric2ModelicaModel(b, a);
    avm_modelica_Limit2ModelicaModel(b, a);
    avm_modelica_Connector2Redeclare(a, b);
    avm_modelica_Connector2Parameter(a, b);
    avm_modelica_Parameter2Value(a, b);
    avm_modelica_Limit2Value(b, a);
    avm_CompoundProperty2CompoundProperty(b, a);
    avm_CompoundProperty2PrimitiveProperty(a, b);
    avm_PrimitiveProperty2Value(a, b);
    avm_simulink_Port12SimulinkModel(b, a);
    avm_simulink_Parameter2SimulinkModel(b, a);
    avm_simulink_Parameter2Value(a, b);
    avm_DataSource2Value(b, a);
    avm_Value2ValueExpressionType(a, b);
    avm_ParametricValue2ValueExpressionType(a, b); //ParametricValue -> Default
    avm_ParametricValue2ValueExpressionType1(a, b); //ParametricValue -> Maximum
    avm_ParametricValue2ValueExpressionType2(a, b); //ParametricValue -> Minimum
    avm_ParametricValue2ValueExpressionType3(a, b); //ParametricValue -> AssignedValue
    avm_FixedValue2ParametricEnumeratedValue(b, a);
    avm_ParametricEnumeratedValue2ValueExpressionType(a, b);
    avm_NormalDistribution2ValueExpressionType(a, b); //NormalDistribution -> Mean
    avm_NormalDistribution2ValueExpressionType1(a, b). //NormalDistribution ->
        StandardDeviation
```

Converts ID-based referencing for port map relationships into a simple (src,dst) relationship pair.

```
AVMPortMap ::= (avm_PortMapTarget2StringSrc, avm_PortMapTarget2StringSrc).
  AVMPortMap(src, dst) :-
    avm_PortMapTarget2String(src, dstID), HasID(dst, dstID), dst:
        avm_PortMapTarget2StringSrc.
```

Converts ID-based referencing for connector composition relationships into a simple (src,dst) relationship pair.

```
AVMConnectorComposition ::= (avm_ConnectorCompositionTarget2StringSrc,
    avm_ConnectorCompositionTarget2StringSrc).
  AVMConnectorComposition(src, dst) :-
    avm_ConnectorCompositionTarget2String(src, dstID), HasID(dst, dstID), dst:
        avm_ConnectorCompositionTarget2StringSrc.
```

Specifies that each avm.Property's flattened container should be the container of the top-levelavm.CompoundProperty.

```
FlattenedPropertyParent ::= (avm_Property, AVMInterchange.Data).
  FlattenedPropertyParent(prop, parent) :-
    Contains(parent, prop), parent: AVMInterchange.Data, prop: avm_Property,
        rflIsMember(parent, #avm_CompoundProperty) = FALSE;
    Contains(parent_prop, prop), prop: avm_Property, parent_prop: avm_CompoundProperty,
        FlattenedPropertyParent(parent_prop, parent).
```

The ValueContainer type specifies a union of all types that can contain an avm.Value.

```
ValueContainer ::=
    avm_ComponentPrimitivePropertyInstance +
    avm_DomainModelMetric +
    avm_PrimitiveProperty +
    avm_cad_Parameter +
```

```
    avm_modelica_Redeclare +
    avm_modelica_Parameter +
    avm_modelica_Limit +
    avm_simulink_Parameter +
    avm_manufacturing_Parameter.
```

Helper rule for deriving the DataType attribute of the avm.Value contained by a ValueContainer.

```
ValueContainerDataType ::= (ValueContainer, String).
  ValueContainerDataType(vc, dataType) :-
    Contains(vc, avm_Value(_, dataType, _, _, _, _, _, _)), vc: ValueContainer.
```

Helper rule for deriving the ID attribute of the avm.Value contained by a ValueContainer.

```
ValueContainerID ::= (ValueContainer, String).
  ValueContainerID(vc, id) :-
    Contains(vc, avm_Value(_, _, _, _, _, _, id, _)), vc: ValueContainer.
```

Helper rule for deriving the Unit attribute of the avm.Value contained by a ValueContainer.

```
ValueContainerUnit ::= (ValueContainer, String).
  ValueContainerUnit(vc, unit) :-
    Contains(vc, avm_Value(_, _, _, _, _, _, _, unit)), vc: ValueContainer.
```

Helper rule for deriving the Dimensions attribute of the avm.Value contained by a ValueContainer.

```
ValueContainerDimensions ::= (ValueContainer, String).
  ValueContainerDimensions(vc, dims) :-
    Contains(vc, avm_Value(_, _, _, dims, _, _, _, _)), vc: ValueContainer.
```

Helper rule for deriving the Value of an avm.Value object.

```
ValueContainerValue ::= (ValueContainer, String).
  ValueContainerValue(vc, value) :-
    Contains(vc, vet), vc: ValueContainer, ValueExpressionTypeValue(vet, value);
    vc is ValueContainer, no { vet | Contains(vc, vet), vet: avm_ValueExpressionType },
        value = "".
```

Helper rule for deriving the Value of an avm.ValueExpressionType.

```
ValueExpressionTypeValue ::= (avm_ValueExpressionType, String).
  ValueExpressionTypeValue(vet, value) :-
    vet is avm_FixedValue(_, _, _, value);
    vet is avm_DerivedValue(_, otherValueID), Contains(avm_Value(_, _, _, _, _, _,
        otherValueID, _), otherVET), ValueExpressionTypeValue(otherVET, value);
    avm_ParametricValue2ValueExpressionType3(vet, assignedVET),
        ValueExpressionTypeValue(assignedVET, value);
    vet is avm_ParametricValue, no avm_ParametricValue2ValueExpressionType3(vet, _),
        value = "";
    vet is avm_ValueExpressionType, rflIsMember(vet, #avm_FixedValue) = FALSE,
        rflIsMember(vet, #avm_DerivedValue) = FALSE, rflIsMember(vet,
        #avm_ParametricValue) = FALSE, value = "".
```

Helper rule for deriving the minimum value for the Range of an avm.ParametricValue.

```
ParametricValueRangeMin ::= (avm_ParametricValue, String).
  ParametricValueRangeMin(pv, rangeMin) :-
    avm_ParametricValue2ValueExpressionType2(pv, minVET),
        ValueExpressionTypeValue(minVET, rangeMin).
  ParametricValueRangeMin(pv, "-inf") :-
    pv is avm_ParametricValue, no avm_ParametricValue2ValueExpressionType2(pv, _).
```

Helper rule for deriving the maximum value for the Range of an avm.ParametricValue.

```
ParametricValueRangeMax ::= (avm_ParametricValue, String).
  ParametricValueRangeMax(pv, rangeMax) :-
    avm_ParametricValue2ValueExpressionType1(pv, maxVET),
        ValueExpressionTypeValue(maxVET, rangeMax).
  ParametricValueRangeMax(pv, "inf") :-
    pv is avm_ParametricValue, no avm_ParametricValue2ValueExpressionType1(pv, _).
```

Helper rule for deriving the Range of an avm.ParametricValue.

```
ParametricValueRange ::= (avm_ParametricValue, String).
  ParametricValueRange(pv, range) :-
    ParametricValueRangeMin(pv, minValue), ParametricValueRangeMax(pv, maxValue),
    range = strJoin(strJoin(minValue, ".."), maxValue).
```

Helper rule for deriving the DefaultValue of an avm.ParametricValue.

```
ParametricValueDefault ::= (avm_ParametricValue, String).
  ParametricValueDefault(pv, default) :-
    avm_ParametricValue2ValueExpressionType(pv, defVET),
        ValueExpressionTypeValue(defVET, default).
```

Constructs a newline-separated list of Component classifications. Since FORMULA requires stratification,which implies an ordering mechanism, we use the toNatural hash of the classification string. In realitythe ordering is unimportant.

```
ComponentClassificationsBuilder ::= (avm_Component, String, String).
  ComponentClassificationsBuilder(comp, class, cons) :-
    ComponentClassificationNext(comp, class, _), no ComponentClassificationNext(comp, _,
        class), cons = class;
    ComponentClassificationsBuilder(comp, prevClass, prevCons),
        ComponentClassificationNext(comp, prevClass, class), cons =
        strJoin(strJoin(prevCons, "\n"), class).

  ComponentClassificationNext ::= (avm_Component, String, String).
  ComponentClassificationNext(comp, class1, class2) :-
    avm_Component2String(comp, class1), hash1 = toNatural(class1),
    avm_Component2String(comp, class2), hash2 = toNatural(class2),
    hash1 < hash2, no { hash | avm_Component2String(comp, class), hash =
        toNatural(class), hash > hash1, hash < hash2 }.
```

Selects the completed construction from ComponentClassificationsBuilder for each Component.Defaults to an empty string if there is no collection for the Component.

```
ComponentClassifications ::= (avm_Component, String).
  ComponentClassifications(comp, classifications) :-
    ComponentClassificationsBuilder(comp, class, classifications), no
        ComponentClassificationNext(comp, class, _);
    comp is avm_Component, classifications = "", no
        ComponentClassificationsBuilder(comp, _, _).
```

6.2 Transformations

6.2.1 AVM Interchange Mapping to CyPhyML

File AVM2CyPhyML.4ml
Uses AVMInterchangeHelpful at "AVMInterchangeHelpful.4ml"
CyPhyML at "CyPhyML.4ml"
ErrorMsg at "ErrorMsg.4ml"

6.2.1.1 General

This transform maps models representing interchange exports into their CyPhyML representation. In doing so, it has two main purposes. First, it provides a precise specification for how these interchange representations should be imported. Second, it provides an indirect semantic specification for the interchange format since the semantic mapping of CyPhyML composed with this mapping is a semantic mapping for this transform's source domain.Some helper rules that relate specifically to this transformation are included within it. The more general ones can be found in AVMInterchangeHelpful.4ml.The majority of rules in this transform take the form of inferring AVM2CyPhyML maps for each input element that map to the CyPhyML output. This rules are written in an extremely verbose style which should make clear what elements and element attributes in the interchange model correspond to in the CyPhyML domain. Keeps a temporary (lifetime is bound to the transform) record of mapped elements. This is useful for relations which need to generate corresponding relations in the output model on the mapped versions of the related elements in the input model.

```
AVM2CyPhyML ::= (in1.Data, out1.Data).
```

6.2.1.2 Component Mapping

avm.CompoundProperty has no direct mapping in CyPhyML. Instead, compound properties are flattened and the names of the mapped properties are based on a path convention. For example, if the PrimitiveProperty named C is contained in a CompoundProperty named B which is contained in a CompoundProperty named A, then the PrimitiveProperty is mapped to a Property or Parameter in CyPhyML that has name A__B__C. This rule encodes the computation of these compound property paths.

```
CompoundPropertyPath ::= (in1.avm_Property, String).
  CompoundPropertyPath(prop, path) :-
    prop is in1.avm_CompoundProperty(_, _, _, path, _, _, _), no { parent |
        in1.Contains(parent, prop), parent: in1.avm_CompoundProperty };
    prop is in1.avm_PrimitiveProperty(_, _, _, path, _, _, _), no { parent |
        in1.Contains(parent, prop), parent: in1.avm_CompoundProperty };
    prop is in1.avm_CompoundProperty(_, _, _, prop_name, _, _, _), in1.Contains(parent,
        prop), CompoundPropertyPath(parent, parent_path), path = strJoin(parent_path,
        strJoin("__", prop_name));
    prop is in1.avm_PrimitiveProperty(_, _, _, prop_name, _, _, _), in1.Contains(parent,
        prop), CompoundPropertyPath(parent, parent_path), path = strJoin(parent_path,
        strJoin("__", prop_name)).
```

Specifies the mapping from the AVM DataTypeEnum to the CyPhyML Parameter/Property DataType enumeration.

```
DataTypePropertyParameterEnumMap ::= (in1.DataTypeEnum,
    out1.Enum_Parameter_Property_DataType).
  DataTypePropertyParameterEnumMap("Integer", "Integer").
  DataTypePropertyParameterEnumMap("Boolean", "Boolean").
  DataTypePropertyParameterEnumMap("String",  "String").
  DataTypePropertyParameterEnumMap("Real",    "Float").
```

Specifies the mapping from the AVM DataTypeEnum to the CyPhyML CADParameterType enumeration.

```
DataTypeCADParameterEnumMap ::= (in1.DataTypeEnum,
    out1.Enum_CADParameter_CADParameterType).
  DataTypeCADParameterEnumMap("Integer", "Integer").
  DataTypeCADParameterEnumMap("Boolean", "Boolean").
  DataTypeCADParameterEnumMap("String",  "String").
  DataTypeCADParameterEnumMap("Real",    "Float").
```

Specifies the mapping from the AVM Modelica RedeclareTypeEnum to the CyPhyML ModelicaRedeclare ModelicaRedeclareType enumeration. All mappings go to the CyPhyML metamodel default of Package if the TypeSpecified boolean is false.

```
ModelicaRedeclareTypeEnumMap ::= (Boolean, in1.RedeclareTypeEnum,
    out1.Enum_ModelicaRedeclare_ModelicaRedeclareType).
  ModelicaRedeclareTypeEnumMap(TRUE,  "Block",     "Block").
  ModelicaRedeclareTypeEnumMap(TRUE,  "Class",     "Class").
  ModelicaRedeclareTypeEnumMap(TRUE,  "Connector", "Connector").
  ModelicaRedeclareTypeEnumMap(TRUE,  "Function",  "Function").
  ModelicaRedeclareTypeEnumMap(TRUE,  "Model",     "Model").
  ModelicaRedeclareTypeEnumMap(TRUE,  "Package",   "Package").
  ModelicaRedeclareTypeEnumMap(TRUE,  "Record",    "Record").
  ModelicaRedeclareTypeEnumMap(FALSE, "Block",     "Package").
  ModelicaRedeclareTypeEnumMap(FALSE, "Class",     "Package").
  ModelicaRedeclareTypeEnumMap(FALSE, "Connector", "Package").
  ModelicaRedeclareTypeEnumMap(FALSE, "Function",  "Package").
  ModelicaRedeclareTypeEnumMap(FALSE, "Model",     "Package").
  ModelicaRedeclareTypeEnumMap(FALSE, "Package",   "Package").
  ModelicaRedeclareTypeEnumMap(FALSE, "Record",    "Package").
```

Specifies the mapping from the AVM ITARRestrictionLevelEnum to the CyPhyML ITAR RestrictionLevel enumeration.

```
ITARRestrictionLevelEnumMap ::= (in1.ITARRestrictionLevelEnum,
    out1.Enum_ITAR_RestrictionLevel).
  ITARRestrictionLevelEnumMap("ITAR",              "ITAR").
  ITARRestrictionLevelEnumMap("ITARDistributionD", "ITARDistributionD").
  ITARRestrictionLevelEnumMap("NotITAR",           "NotITAR").
```

The avm.DerivedValue ValueExpressionType results in a ValueFlow between the mapped CyPhyML elements of the Value containers.

```
out1.ValueFlow("", outSrc, outDst, "", "", 0) :-
    inSrc is ValueContainer, inDst is ValueContainer,
    in1.Contains(inSrc, srcValue), in1.Contains(srcValue, in1.avm_DerivedValue(_,
        dstValueID)),
    in1.Contains(inDst, in1.avm_Value(_, _, _, _, _, _, dstValueID, _)),
    AVM2CyPhyML(inSrc, outSrc), outSrc: out1.ValueFlow_Source,
    AVM2CyPhyML(inDst, outDst), outDst: out1.ValueFlow_Destination.
```

PortMap references on avm.PortMapTarget types result in a PortComposition between the mapped CyPhyML elements of the avm.PortMapTargets. (See AVMPortMap in AVMInterchangeHelpful.4ml)

```
out1.PortComposition("", outSrc, outDst, "", 0) :-
    in1.AVMPortMap(inSrc, inDst),
    AVM2CyPhyML(inSrc, outSrc), outSrc: out1.PortComposition_Source,
    AVM2CyPhyML(inDst, outDst), outDst: out1.PortComposition_Destination.
```

ConnectorComposition references on avm.ConnectorCompositionTarget types result in a ConnectorComposition between the mapped CyPhyML elements of the avm.ConnectorCompositionTargets. (See AVMConnectorComposition in AVMInterchangeHelpful.4ml)

```
out1.ConnectorComposition("", outSrc, outDst, "", 0) :-
    in1.AVMConnectorComposition(inSrc, inDst),
    AVM2CyPhyML(inSrc, outSrc), outSrc: out1.Connector,
    AVM2CyPhyML(inDst, outDst), outDst: out1.Connector.
```

An avm.DomainModel with a UsesResource reference ID to a Resource maps to a UsesResource relationship between the mapped elements in CyPhyML.

```
out1.UsesResource("", outSrc, outDst, "", 0) :-
    inSrc is in1.avm_modelica_ModelicaModel(_, _, _, _, _, resourceID),
    inDst is in1.avm_Resource(_, _, resourceID, _, _, _),
    AVM2CyPhyML(inSrc, outSrc), outSrc: out1.UsesResource_Source,
    AVM2CyPhyML(inDst, outDst), outDst: out1.UsesResource_Destination.
```

```
  out1.UsesResource("", outSrc, outDst, "", 0) :-
    inSrc is in1.avm_cad_CADModel(_, _, _, _, resourceID),
    inDst is in1.avm_Resource(_, _, resourceID, _, _, _),
    AVM2CyPhyML(inSrc, outSrc), outSrc: out1.UsesResource_Source,
    AVM2CyPhyML(inDst, outDst), outDst: out1.UsesResource_Destination.

  out1.UsesResource("", outSrc, outDst, "", 0) :-
    inSrc is in1.avm_manufacturing_ManufacturingModel(_, _, _, _, resourceID),
    inDst is in1.avm_Resource(_, _, resourceID, _, _, _),
    AVM2CyPhyML(inSrc, outSrc), outSrc: out1.UsesResource_Source,
    AVM2CyPhyML(inDst, outDst), outDst: out1.UsesResource_Destination.

  out1.UsesResource("", outSrc, outDst, "", 0) :-
    inSrc is in1.avm_simulink_SimulinkModel(_, _, _, _, _, resourceID),
    inDst is in1.avm_Resource(_, _, resourceID, _, _, _),
    AVM2CyPhyML(inSrc, outSrc), outSrc: out1.UsesResource_Source,
    AVM2CyPhyML(inDst, outDst), outDst: out1.UsesResource_Destination.
```

Specifies mapping of avm.Component to CyPhyML Component.

```
AVM2CyPhyML(
    avmComponent,
    out1.Component(
      name,             //name
      id,               //AVMID
      "",               //Cardinality
      classifications,  //Classifications
      "",               //Description
      0,                //ID
      "",               //InstanceGUID
      "",               //ManagedGUID
      "",               //NumAssociatedConfigs
      "",               //Revision
      version,          //Version
      "",               //objectID
      aID               //iObjectID
    )
  ) :-
    avmComponent is in1.avm_Component(
      aID,     //aObjectID
      id,      //ID
      name,    //Name
      _,       //SchemaVersion
      version  //Version
    ),
    in1.ComponentClassifications(avmComponent, classifications)
  .

  out1.Component(a,b,c,d,e,f,g,h,i,j,k,l,m) :- AVM2CyPhyML(_,
      out1.Component(a,b,c,d,e,f,g,h,i,j,k,l,m)).
```

Specifies mapping of avm.AbstractPort to CyPhyML AbstractPort.

```
AVM2CyPhyML(
    avmAbstractPort,
    out1.AbstractPort(
      name,        //name
      definition,  //Definition
      "",          //DefinitionNotes
      id,          //ID
      notes,       //InstanceNotes
      "",          //objectID
      aID          //iObjectID
    )
  ) :-
    avmAbstractPort is in1.avm_AbstractPort(
      aID,         //aObjectID
```

```
    definition, //Definition
    id,         //ID
    name,       //Name
    notes       //Notes
  )
.

out1.DesignElementToPortContainment(outComp, outPort) :-
  in1.Contains(inComp, inPort),
  AVM2CyPhyML(inComp, outComp), outComp: out1.DesignElementToPortContainment_Container,
  AVM2CyPhyML(inPort, outPort), outPort: out1.AbstractPort.

out1.ConnectorToPortContainment(outConn, outPort) :-
  in1.Contains(inConn, inPort),
  AVM2CyPhyML(inConn, outConn), outConn: out1.ConnectorToPortContainment_Container,
  AVM2CyPhyML(inPort, outPort), outPort: out1.AbstractPort.

out1.DesignContainerToPortContainment(outCont, outPort) :-
  in1.Contains(inCont, inPort),
  AVM2CyPhyML(inCont, outCont), outCont:
      out1.DesignContainerToPortContainment_Container,
  AVM2CyPhyML(inPort, outPort), outPort: out1.AbstractPort.

out1.AbstractPort(a,b,c,d,e,f,g) :- AVM2CyPhyML(_, out1.AbstractPort(a,b,c,d,e,f,g)).
```

Specifies mapping of avm.DistributionRestriction subtypes to their CyPhyML equivalents.

```
AVM2CyPhyML(
    avmSecClass,
    out1.SecurityClassification(
      "",     //name
      level, //Level
      notes, //Notes
      "",     //objectID
      aID     //iObjectID
    )
  ) :-
    avmSecClass is in1.avm_SecurityClassification(
      aID,    //aObjectID
      level, //Level
      notes   //Notes
    )
  .

  out1.SecurityClassification(a,b,c,d,e) :- AVM2CyPhyML(_,
      out1.SecurityClassification(a,b,c,d,e)).

  AVM2CyPhyML(
    avmProprietary,
    out1.Proprietary(
      "",     //name
      notes, //Notes
      org,    //Organization
      "",     //objectID
      aID     //iObjectID
    )
  ) :-
    avmProprietary is in1.avm_Proprietary(
      aID,    //aObjectID
      notes, //Notes
      org     //Organization
    )
  .

  out1.Proprietary(a,b,c,d,e) :- AVM2CyPhyML(_, out1.Proprietary(a,b,c,d,e)).

  AVM2CyPhyML(
    avmITAR,
```

```
    out1.ITAR(
      "",          //name
      notes,       //Notes
      cyPhyLevel,  //RestrictionLevel
      "",          //objectID
      aID          //iObjectID
    )
  ) :-
    avmITAR is in1.avm_ITAR(
      aID,       //aObjectID
      avmLevel,  //Level
      notes      //Notes
    ),
    ITARRestrictionLevelEnumMap(avmLevel, cyPhyLevel)
  .

  out1.ITAR(a,b,c,d,e) :- AVM2CyPhyML(_, out1.ITAR(a,b,c,d,e)).

  out1.ComponentToDistributionRestrictionContainment(outComp, outDR) :-
    in1.Contains(inComp, inDR),
    AVM2CyPhyML(inComp, outComp), outComp:
        out1.ComponentToDistributionRestrictionContainment_Container,
    AVM2CyPhyML(inDR, outDR), outDR:
        out1.ComponentToDistributionRestrictionContainment_Contained.
```

Specifies mapping of avm.Connector to CyPhyML Connector.

```
AVM2CyPhyML(
    avmConnector,
    out1.Connector(
      name,        //name
      definition,  //Definition
      "",          //DefinitionNotes
      id,          //ID
      notes,       //InstanceNotes
      "",          //objectID
      aID          //iObjectID
    )
  ) :-
    avmConnector is in1.avm_Connector(
      aID,         //aObjectID
      definition,  //Definition
      id,          //ID
      name,        //Name
      notes        //Notes
    )
  .

  out1.DesignElementToConnectorContainment(outComp, outConn) :-
    in1.Contains(inComp, inConn),
    AVM2CyPhyML(inComp, outComp), outComp:
        out1.DesignElementToConnectorContainment_Container,
    AVM2CyPhyML(inConn, outConn), outConn: out1.Connector.

  out1.DesignContainerToConnectorContainment(outCont, outConn) :-
    in1.Contains(inCont, inConn),
    AVM2CyPhyML(inCont, outCont), outCont:
        out1.DesignContainerToConnectorContainment_Container,
    AVM2CyPhyML(inConn, outConn), outConn: out1.Connector.

  out1.Connector(a,b,c,d,e,f,g) :- AVM2CyPhyML(_, out1.Connector(a,b,c,d,e,f,g)).
```

Specifies mapping of avm.PrimitiveProperty to CyPhyML Property. The mapping requires that the PrimitiveProperty's Value's ValueExpressionType is not avm.ParametricValue, else it maps to a CyPhyML Parameter.

```
AVM2CyPhyML(
```

```
    avmPrimProp,
    out1.Property(
      compName,         //name
      CyPhyMLNullRef,   //target
      "Addition",       //ComputationType
      cyphyDataType,    //DataType
      "",               //DefaultValue
      "",               //Description
      dims,             //Dimension
      FALSE,            //Disable
      id,               //ID
      TRUE,             //IsProminent
      "",               //ManagedGUID
      "",               //Tolerance
      "",               //Type
      value,            //Value
      "",               //objectID
      aID               //iObjectID
    )
  ) :-
    avmPrimProp is in1.avm_PrimitiveProperty,
    CompoundPropertyPath(avmPrimProp, compName),
    in1.Contains(avmPrimProp, avmValue),
    avmValue = in1.avm_Value(
      aID,          //aObjectID
      avmDataType,  //DataType
      _,            //DataTypeSpecified
      dims,         //Dimensions
      _,            //DimensionType
      _,            //DimensionTypeSpecified
      id,           //ID
      _             //Unit
    ),
    DataTypePropertyParameterEnumMap(avmDataType, cyphyDataType),
    in1.Contains(avmValue, avmValueExpressionType), rflIsMember(avmValueExpressionType,
        in1.#avm_ParametricValue) = FALSE,
    in1.ValueContainerValue(avmPrimProp, value)
  .

  out1.DesignElementToPropertyContainment(outComp, outProp) :-
    in1.FlattenedPropertyParent(inProp, inComp),
    AVM2CyPhyML(inComp, outComp), outComp:
        out1.DesignElementToPropertyContainment_Container,
    AVM2CyPhyML(inProp, outProp), outProp: out1.Property.

  out1.ConnectorToPropertyContainment(outConn, outProp) :-
    in1.FlattenedPropertyParent(inProp, inConn),
    AVM2CyPhyML(inConn, outConn), outConn: out1.ConnectorToPropertyContainment_Container,
    AVM2CyPhyML(inProp, outProp), outProp: out1.Property.

  out1.DesignContainerToPropertyContainment(outCont, outProp) :-
    in1.FlattenedPropertyParent(inProp, inCont),
    AVM2CyPhyML(inCont, outCont), outCont:
        out1.DesignContainerToPropertyContainment_Container,
    AVM2CyPhyML(inProp, outProp), outProp: out1.Property.

  out1.Property(a,b,c,d,e,f,g,h,i,j,k,l,m,n,o,p) :- AVM2CyPhyML(_,
      out1.Property(a,b,c,d,e,f,g,h,i,j,k,l,m,n,o,p)).
```

Specifies the mapping of avm.PrimitiveProperty to CyPhyML Parameter. The mapping occurs if the PrimitiveProperty's Value's ValueExpressionType is an avm.ParametricValue, otherwise it maps to a CyPhyML Property.

```
AVM2CyPhyML(
    avmPrimProp,
    out1.Parameter(
      compName,         //name
```

```
      CyPhyMLNullRef, //target
      "Addition",     //ComputationType
      cyphyDataType,  //DataType
      defValue,       //DefaultValue
      "",             //Description
      dims,           //Dimension
      FALSE,          //Disable
      id,             //ID
      "",             //ManagedGUID
      range,          //Range
      "",             //Type
      value,          //Value
      "",             //objectID
      aID             //iObjectID
    )
  ) :-
    avmPrimProp is in1.avm_PrimitiveProperty(
      aID,  //aObjectID
      _,    //Definition
      _,    //ID
      _,    //Name
      _,    //Notes
      _,    //OnDataSheet
      _     //OnDataSheetSpecified
    ),
    CompoundPropertyPath(avmPrimProp, compName),
    in1.Contains(avmPrimProp, avmValue), avmValue: in1.avm_Value,
    in1.Contains(avmValue, avmParametricValue), avmParametricValue:
        in1.avm_ParametricValue,
    in1.ValueContainerID(avmPrimProp, id),
    in1.ValueContainerDimensions(avmPrimProp, dims),
    in1.ValueContainerDataType(avmPrimProp, avmDataType),
    DataTypePropertyParameterEnumMap(avmDataType, cyphyDataType),
    in1.ValueExpressionTypeValue(avmParametricValue, value),
    in1.ParametricValueRange(avmParametricValue, range),
    in1.ParametricValueDefault(avmParametricValue, defValue)
  .

  out1.DesignElementToParameterContainment(outComp, parm) :-
    in1.FlattenedPropertyParent(primProp, inComp),
    AVM2CyPhyML(inComp, outComp), outComp:
        out1.DesignElementToParameterContainment_Container,
    AVM2CyPhyML(primProp, parm), parm: out1.Parameter.

  out1.ConnectorToParameterContainment(outConn, parm) :-
    in1.FlattenedPropertyParent(primProp, inConn),
    AVM2CyPhyML(inConn, outConn), outConn: out1.Connector,
    AVM2CyPhyML(primProp, parm), parm: out1.Parameter.

  out1.DesignContainerToParameterContainment(outCont, parm) :-
    in1.FlattenedPropertyParent(primProp, inCont),
    AVM2CyPhyML(inCont, outCont), outCont:
        out1.DesignContainerToParameterContainment_Container,
    AVM2CyPhyML(primProp, parm), parm: out1.Parameter.

  out1.Parameter(a,b,c,d,e,f,g,h,i,j,k,l,m,n,o) :- AVM2CyPhyML(_,
      out1.Parameter(a,b,c,d,e,f,g,h,i,j,k,l,m,n,o)).
```

Specifies mapping of avm.Resource to CyPhyML Resource.

```
AVM2CyPhyML(
    avmResource,
    out1.Resource(
      name,  //name
      hash,  //Hash
      id,    //ID
      notes, //Notes
      path,  //Path
```

```
      "",     //objectID
      aID     //iObjectID
    )
  ) :-
    avmResource is in1.avm_Resource(
      aID,    //aObjectID
      hash,   //Hash
      id,     //ID
      name,   //Name
      notes,  //Notes
      path    //Path
    )
  .

  out1.ComponentToResourceContainment(outComp, outResource) :-
    in1.Contains(inComp, inResource),
    AVM2CyPhyML(inComp, outComp), outComp: out1.ComponentToResourceContainment_Container,
    AVM2CyPhyML(inResource, outResource), outResource: out1.Resource.

  out1.Resource(a,b,c,d,e,f,g) :- AVM2CyPhyML(_, out1.Resource(a,b,c,d,e,f,g)).
```

Specifies mapping of avm.cad.CADModel to CyPhyML CADModel.

```
AVM2CyPhyML(
    avmCADModel,
    out1.CADModel(
      "CADModel", //name
      author,     //Author
      "Creo",     //FileFormat
      file_path,  //FilePathWithinResource
      "Part",     //FileType
      notes,      //Notes
      "",         //objectID
      aID         //iObjectID
    )
  ) :-
    avmCADModel is in1.avm_cad_CADModel(
      aID,        //aObjectID
      author,     //Author
      file_path,  //FilePathWithinResource
      notes,      //Notes
      _           //UsesResource
    )
  .

  out1.ComponentTypeToCADModelContainment(outComp, outModel) :-
    Contains(inComp, inModel),
    AVM2CyPhyML(inComp, outComp), outComp:
        out1.ComponentTypeToCADModelContainment_Container,
    AVM2CyPhyML(inModel, outModel), outModel: out1.CADModel.

  out1.CADModel(a,b,c,d,e,f,g,h) :- AVM2CyPhyML(_, out1.CADModel(a,b,c,d,e,f,g,h)).
```

Specifies mapping of avm.cad.Axis to CyPhyML Axis.

```
AVM2CyPhyML(
    avmAxis,
    out1.Axis(
      name,       //name
      datum_name, //DatumName
      definition, //Definition
      notes,      //DefinitionNotes
      id,         //ID
      notes,      //InstanceNotes
      "",         //objectID
      aID         //iObjectID
    )
```

```
  ) :-
    avmAxis is in1.avm_cad_Axis(
      aID,         //aObjectID
      datum_name,  //DatumName
      definition,  //Definition
      id,          //ID
      name,        //Name
      notes        //Notes
    )
  .

  out1.DesignElementToPortContainment(outComp, outAxis) :-
    in1.Contains(inComp, inAxis),
    AVM2CyPhyML(inComp, outComp), outComp: out1.DesignElementToPortContainment_Container,
    AVM2CyPhyML(inAxis, outAxis), outAxis: out1.Axis.

  out1.CADModelToCADDatumContainment(outModel, outAxis) :-
    in1.Contains(inModel, inAxis),
    AVM2CyPhyML(inModel, outModel), outModel:
        out1.CADModelToCADDatumContainment_Container,
    AVM2CyPhyML(inAxis, outAxis), outAxis: out1.Axis.

  out1.ConnectorToPortContainment(outConn, outAxis) :-
    in1.Contains(inConn, inAxis),
    AVM2CyPhyML(inConn, outConn), outConn: out1.ConnectorToPortContainment_Container,
    AVM2CyPhyML(inAxis, outAxis), outAxis: out1.Axis.

  out1.DesignContainerToPortContainment(outCont, outPort) :-
    in1.Contains(inCont, inPort),
    AVM2CyPhyML(inCont, outCont), outCont:
        out1.DesignContainerToPortContainment_Container,
    AVM2CyPhyML(inPort, outPort), outPort: out1.Axis.

  out1.Axis(a,b,c,d,e,f,g,h) :- AVM2CyPhyML(_, out1.Axis(a,b,c,d,e,f,g,h)).
```

Specifies mapping of avm.cad.Point to CyPhyML Point.

```
AVM2CyPhyML(
    avmPoint,
    out1.Point(
      name,        //name
      datum_name,  //DatumName
      definition,  //Definition
      notes,       //DefinitionNotes
      id,          //ID
      notes,       //InstanceNotes
      "",          //objectID
      aID          //iObjectID
    )
  ) :-
    avmPoint is in1.avm_cad_Point(
      aID,         //aObjectID
      datum_name,  //DatumName
      definition,  //Definition
      id,          //ID
      name,        //Name
      notes        //Notes
    )
  .

  out1.DesignElementToPortContainment(outComp, outPoint) :-
    in1.Contains(inComp, inPoint),
    AVM2CyPhyML(inComp, outComp), outComp: out1.DesignElementToPortContainment_Container,
    AVM2CyPhyML(inPoint, outPoint), outPoint: out1.Point.

  out1.CADModelToCADDatumContainment(outModel, outPoint) :-
    in1.Contains(inModel, inPoint),
    AVM2CyPhyML(inModel, outModel), outModel:
```

```
        out1.CADModelToCADDatumContainment_Container,
    AVM2CyPhyML(inPoint, outPoint), outPoint: out1.Point.

  out1.ConnectorToPortContainment(outConn, outPoint) :-
    in1.Contains(inConn, inPoint),
    AVM2CyPhyML(inConn, outConn), outConn: out1.ConnectorToPortContainment_Container,
    AVM2CyPhyML(inPoint, outPoint), outPoint: out1.Point.

  out1.DesignContainerToPortContainment(outCont, outPort) :-
    in1.Contains(inCont, inPort),
    AVM2CyPhyML(inCont, outCont), outCont:
        out1.DesignContainerToPortContainment_Container,
    AVM2CyPhyML(inPort, outPort), outPort: out1.Point.

  out1.Point(a,b,c,d,e,f,g,h) :- AVM2CyPhyML(_, out1.Point(a,b,c,d,e,f,g,h)).
```

Specifies mapping of avm.cad.CoordinateSystem to CyPhyML CoordinateSystem.

```
AVM2CyPhyML(
    avmCoordinateSystem,
    out1.CoordinateSystem(
      name,        //name
      datum_name,  //DatumName
      definition,  //Definition
      notes,       //DefinitionNotes
      id,          //ID
      notes,       //InstanceNotes
      "",          //objectID
      aID          //iObjectID
    )
  ) :-
    avmCoordinateSystem is in1.avm_cad_CoordinateSystem(
      aID,         //aObjectID
      datum_name,  //DatumName
      definition,  //Definition
      id,          //ID
      name,        //Name
      notes        //Notes
    )
  .

  out1.DesignElementToPortContainment(outComp, outCoordSys) :-
    in1.Contains(inComp, inCoordSys),
    AVM2CyPhyML(inComp, outComp), outComp: out1.DesignElementToPortContainment_Container,
    AVM2CyPhyML(inCoordSys, outCoordSys), outCoordSys: out1.CoordinateSystem.

  out1.CADModelToCADDatumContainment(outModel, outCoordSys) :-
    in1.Contains(inModel, inCoordSys),
    AVM2CyPhyML(inModel, outModel), outModel:
        out1.CADModelToCADDatumContainment_Container,
    AVM2CyPhyML(inCoordSys, outCoordSys), outCoordSys: out1.CoordinateSystem.

  out1.ConnectorToPortContainment(outConn, outCoordSys) :-
    in1.Contains(inConn, inCoordSys),
    AVM2CyPhyML(inConn, outConn), outConn: out1.ConnectorToPortContainment_Container,
    AVM2CyPhyML(inCoordSys, outCoordSys), outCoordSys: out1.CoordinateSystem.

  out1.DesignContainerToPortContainment(outCont, outPort) :-
    in1.Contains(inCont, inPort),
    AVM2CyPhyML(inCont, outCont), outCont:
        out1.DesignContainerToPortContainment_Container,
    AVM2CyPhyML(inPort, outPort), outPort: out1.CoordinateSystem.

  out1.CoordinateSystem(a,b,c,d,e,f,g,h) :- AVM2CyPhyML(_,
      out1.CoordinateSystem(a,b,c,d,e,f,g,h)).
```

Specifies mapping of avm.cad.Plane to CyPhyML Surface.

```
AVM2CyPhyML(
    avmPlane,
    out1.Surface(
      name,        //name
      "MATE",      //Alignment (Not set in C# importer, default in GME metamodel is MATE)
      datum_name,  //DatumName
      definition,  //Definition
      notes,       //DefinitionNotes
      id,          //ID
      notes,       //InstanceNotes
      "",          //objectID
      aID          //iObjectID
    )
  ) :-
    avmPlane is in1.avm_cad_Plane(
      aID,         //aObjectID
      datum_name,  //DatumName
      definition,  //Definition
      id,          //ID
      name,        //Name
      notes        //Notes
    )
  .

  out1.DesignElementToPortContainment(outComp, surf) :-
    in1.Contains(inComp, avmPlane),
    AVM2CyPhyML(inComp, outComp), outComp: out1.DesignElementToPortContainment_Container,
    AVM2CyPhyML(avmPlane, surf), surf: out1.Surface.

  out1.CADModelToCADDatumContainment(outModel, surf) :-
    in1.Contains(inModel, avmPlane),
    AVM2CyPhyML(inModel, outModel), outModel:
        out1.CADModelToCADDatumContainment_Container,
    AVM2CyPhyML(avmPlane, surf), surf: out1.Surface.

  out1.ConnectorToPortContainment(outConn, surf) :-
    in1.Contains(inConn, avmPlane),
    AVM2CyPhyML(inConn, outConn), outConn: out1.ConnectorToPortContainment_Container,
    AVM2CyPhyML(avmPlane, surf), surf: out1.Surface.

  out1.DesignContainerToPortContainment(outCont, outPort) :-
    in1.Contains(inCont, inPort),
    AVM2CyPhyML(inCont, outCont), outCont:
        out1.DesignContainerToPortContainment_Container,
    AVM2CyPhyML(inPort, outPort), outPort: out1.Surface.

  out1.Surface(a,b,c,d,e,f,g,h,i) :- AVM2CyPhyML(_, out1.Surface(a,b,c,d,e,f,g,h,i)).
```

Specifies mapping of avm.cad.Parameter to CyPhyML CADParameter.

```
AVM2CyPhyML(
    avmParm,
    out1.CADParameter(
      name,           //name
      cyPhyDataType,  //CADParameterType
      "",             //DefaultValue
      id,             //ID
      "",             //ParameterName
      "-inf..inf",    //Range
      units,          //Unit
      value,          //Value
      "",             //objectID
      aID             //iObjectID
    )
  ) :-
    avmParm = in1.avm_cad_Parameter(
      aID,  //aObjectID
      name, //Name
```

```
      notes //Notes
    ),
    in1.ValueContainerDataType(avmParm, avmDataType),
    in1.ValueContainerID(avmParm, id),
    in1.ValueContainerUnit(avmParm, units),
    in1.ValueContainerValue(avmParm, value),
    DataTypeCADParameterEnumMap(avmDataType, cyPhyDataType)
  .

  out1.CADModelToCADParameterContainment(outModel, outParm) :-
    in1.Contains(inModel, inParm),
    AVM2CyPhyML(inModel, outModel), outModel:
        out1.CADModelToCADParameterContainment_Container,
    AVM2CyPhyML(inParm, outParm), outParm: out1.CADParameter.

  out1.CADParameter(a,b,c,d,e,f,g,h,i,j) :- AVM2CyPhyML(_,
      out1.CADParameter(a,b,c,d,e,f,g,h,i,j)).
```

Specifies mapping of avm.cad.Metric to CyPhyML CADMetric.

```
AVM2CyPhyML(
    avmMetric,
    out1.CADMetric(
      name,  //name
      TRUE,  //IsCurrent
      name,  //ParameterName
      units,  //UnitOfMeasurement
      value, //Value
      "",    //objectID
      aID    //iObjectID
    )
  ) :-
    avmMetric = in1.avm_cad_Metric(
      aID,  //aObjectID
      _,    //ID
      name, //Name
      _     //Notes
    ),
    in1.ValueContainerUnit(avmMetric, units),
    in1.ValueContainerValue(avmMetric, value)
  .

  out1.CADModelToCADMetricContainment(outModel, outMetric) :-
    in1.Contains(inModel, inMetric),
    AVM2CyPhyML(inModel, outModel), outModel: out1.CADModel,
    AVM2CyPhyML(inMetric, outMetric), outMetric: out1.CADMetric.

  //NOTE: Interchange Diagrams allow containment of avm.cad.Metric by avm.cad.Datum, but
      no such
  //      relationship appears to exist in the CyPhyML metamodel.

  out1.CADMetric(a,b,c,d,e,f,g) :- AVM2CyPhyML(_, out1.CADMetric(a,b,c,d,e,f,g)).
```

Specifies mapping of avm.modelica.ModelicaModel to CyPhyML ModelicaModel

```
AVM2CyPhyML(
    avmModelicaModel,
    out1.ModelicaModel(
      "",     //name
      author, //Author
      class,  //Class
      path,   //FilePathWithinResource
      notes,  //Notes
      "",     //objectID
      aID     //iObjectID
    )
  ) :-
```

```
    avmModelicaModel is in1.avm_modelica_ModelicaModel(
      aID,     //aObjectID
      author,  //Author
      class,   //Class
      path,    //FilePathWithinResource
      notes,   //Notes
      _        //UsesResource
    )
  .

  out1.ComponentTypeToModelicaModelContainment(outComp, outModel) :-
    in1.Contains(inComp, inModel),
    AVM2CyPhyML(inComp, outComp), outComp:
        out1.ComponentTypeToModelicaModelContainment_Container,
    AVM2CyPhyML(inModel, outModel), outModel: out1.ModelicaModel.

  out1.ModelicaModel(a,b,c,d,e,f,g) :- AVM2CyPhyML(_, out1.ModelicaModel(a,b,c,d,e,f,g)).
```

Specifies mapping of avm.modelica.Connector to CyPhyML ModelicaConnector.

```
AVM2CyPhyML(
    avmModelicaConnector,
    out1.ModelicaConnector(
      name,        //name
      class,       //Class
      definition,  //Definition
      "",          //DefinitionNotes
      id,          //ID
      notes,       //InstanceNotes
      locator,     //Locator
      "",          //objectID
      aID          //iObjectID
    )
  ) :-
    avmModelicaConnector is in1.avm_modelica_Connector(
      aID,         //aObjectID
      class,       //Class
      definition,  //Definition
      id,          //ID
      locator,     //Locator
      name,        //Name
      notes        //Notes
    )
  .

  out1.DesignElementToPortContainment(outComp, outPort) :-
    in1.Contains(inComp, inPort),
    AVM2CyPhyML(inComp, outComp), outComp: out1.DesignElementToPortContainment_Container,
    AVM2CyPhyML(inPort, outPort), outPort: out1.ModelicaConnector.

  out1.ModelicaModelTypeToModelicaConnectorContainment(outModel, outConn) :-
    in1.Contains(inModel, inConn),
    AVM2CyPhyML(inModel, outModel), outModel:
        out1.ModelicaModelTypeToModelicaConnectorContainment_Container,
    AVM2CyPhyML(inConn, outConn), outConn: out1.ModelicaConnector.

  out1.ConnectorToPortContainment(outConn, outPort) :-
    in1.Contains(inConn, inPort),
    AVM2CyPhyML(inConn, outConn), outConn: out1.ConnectorToPortContainment_Container,
    AVM2CyPhyML(inPort, outPort), outPort: out1.ModelicaConnector.

  out1.DesignContainerToPortContainment(outCont, outPort) :-
    in1.Contains(inCont, inPort),
    AVM2CyPhyML(inCont, outCont), outCont:
        out1.DesignContainerToPortContainment_Container,
    AVM2CyPhyML(inPort, outPort), outPort: out1.ModelicaConnector.

  out1.ModelicaConnector(a,b,c,d,e,f,g,h,i) :- AVM2CyPhyML(_,
```

```
      out1.ModelicaConnector(a,b,c,d,e,f,g,h,i)).
```

Specifies mapping of avm.modelica.Redeclare to CyPhyML ModelicaRedeclare

```
AVM2CyPhyML(
    avmRedeclare,
    out1.ModelicaRedeclare(
      "",         //name
      "",         //DefaultValue
      cyPhyType, //ModelicaRedeclareType
      notes,      //Notes
      value,      //Value
      "",         //objectID
      aID         //iObjectID
    )
  ) :-
    avmRedeclare is in1.avm_modelica_Redeclare(
      aID,      //aObjectID
      _,        //Locator
      notes,    //Notes
      avmType, //Type
      typeSpec //TypeSpecified
    ),
    ModelicaRedeclareTypeEnumMap(typeSpec, avmType, cyPhyType),
    in1.ValueContainerValue(avmRedeclare, value)
  .

  out1.ModelicaModelTypeToModelicaRedeclareContainment(outModel, outRedec) :-
    in1.Contains(inModel, inRedec),
    AVM2CyPhyML(inModel, outModel), outModel:
        out1.ModelicaModelTypeToModelicaRedeclareContainment_Container,
    AVM2CyPhyML(inRedec, outRedec), outRedec: out1.ModelicaRedeclare.

  out1.ModelicaConnectorToModelicaRedeclareContainment(outConn, outRedec) :-
    in1.Contains(inConn, inRedec),
    AVM2CyPhyML(inConn, outConn), outConn: out1.ModelicaConnector,
    AVM2CyPhyML(inRedec, outRedec), outRedec: out1.ModelicaRedeclare.

  out1.ModelicaRedeclare(a,b,c,d,e,f,g) :- AVM2CyPhyML(_,
      out1.ModelicaRedeclare(a,b,c,d,e,f,g)).
```

Specifies mapping of avm.modelica.Parameter to CyPhyML ModelicaParameter

```
AVM2CyPhyML(
    avmParm,
    out1.ModelicaParameter(
      name,  //name
      "",    //DefaultValue
      id,    //ID
      value, //Value
      "",    //objectID
      aID    //iObjectID
    )
  ) :-
    avmParm is in1.avm_modelica_Parameter(
      aID, //aObjectID
      name, //Locator
      _     //Notes
    ),
    in1.ValueContainerID(avmParm, id),
    in1.ValueContainerValue(avmParm, value)
  .

  out1.ModelicaConnectorToModelicaParameterContainment(outConn, outParm) :-
    in1.Contains(inConn, inParm),
    AVM2CyPhyML(inConn, outConn), outConn:
        out1.ModelicaConnectorToModelicaParameterContainment_Container,
```

```
    AVM2CyPhyML(inParm, outParm), outParm: out1.ModelicaParameter.

  //NOTE: Interchange diagrams allow containment of avm.modelica.Parameter by
      avm.modelica.ModelicaModel
  //      but the CyPhyML metamodel does not appear to feature a corresponding
      relationship.

  out1.ModelicaParameter(a,b,c,d,e,f) :- AVM2CyPhyML(_,
      out1.ModelicaParameter(a,b,c,d,e,f)).
```

Specifies mapping of avm.manufacturing.ManufacturingModel to CyPhyML ManufacturingModel.

```
AVM2CyPhyML(
    avmManModel,
    out1.ManufacturingModel(
      "",         //name
      author,     //Author
      file_path,  //FilePathWithinResource
      notes,      //Notes
      "",         //objectID
      aID         //iObjectID
    )
  ) :-
    avmManModel is in1.avm_manufacturing_ManufacturingModel(
      aID,        //aObjectID
      author,     //Author
      file_path,  //FilePathWithinResource
      notes,      //Notes
      _           //UsesResource
    )
  .

  out1.ComponentTypeToManufacturingModelContainment(outComp, outManModel) :-
    in1.Contains(inComp, inManModel),
    AVM2CyPhyML(inComp, outComp), outComp:
        out1.ComponentTypeToManufacturingModelContainment_Container,
    AVM2CyPhyML(inManModel, outManModel), outManModel: out1.ManufacturingModel.

  out1.ManufacturingModel(a,b,c,d,e,f) :- AVM2CyPhyML(_,
      out1.ManufacturingModel(a,b,c,d,e,f)).
```

Specifies mapping of avm.manufacturing.Parameter to CyPhyML ManufacturingModelParameter. NOTE: The Notes variable is not mapped currently in the AVM2CyPhyML importer, but it seems that it should be, so we have done so here.

```
AVM2CyPhyML(
    avmManParameter,
    out1.ManufacturingModelParameter(
      name,           //name
      CyPhyMLNullRef, //target
      "",             //DefaultValue
      notes,          //Notes
      value,          //Value
      "",             //objectID
      aID             //iObjectID
    )
  ) :-
    avmManParameter = in1.avm_manufacturing_Parameter(
      aID, //aObjectID
      _,    //Locator
      name, //Name
      notes //Notes
    ),
    in1.ValueContainerValue(avmManParameter, value)
  .
```

```
  out1.ManufacturingModelToManufacturingModelParameterContainment(outManModel,
      outManParam) :-
    in1.Contains(inManModel, inManParam),
    AVM2CyPhyML(inManModel, outManModel), outManModel: out1.ManufacturingModel,
    AVM2CyPhyML(inManParam, outManParam), outManParam: out1.ManufacturingModelParameter.

  out1.ManufacturingModelParameter(a,b,c,d,e,f,g) :- AVM2CyPhyML(_,
      out1.ManufacturingModelParameter(a,b,c,d,e,f,g)).
```

Specifies mapping of avm.manufacturing.Metric to CyPhyML ManufacturingModelMetric. NOTE: Not in current AVM2CyPhyML importer.

```
AVM2CyPhyML(
    avmManMetric,
    out1.ManufacturingModelMetric(
      name,  //name
      id,    //ID
      notes, //Notes
      "",    //objectID
      aID    //iObjectID
    )
  ) :-
    avmManMetric is in1.avm_manufacturing_Metric(
      aID,  //aObjectID
      id,   //ID
      name, //Name
      notes //Notes
    )
  .

  out1.ManufacturingModelToManufacturingModelMetricContainment(outManModel,
      outManMetric) :-
    in1.Contains(inManModel, inManParam),
    AVM2CyPhyML(inManModel, outManModel), outManModel: out1.ManufacturingModel,
    AVM2CyPhyML(inManMetric, outManMetric), outManMetric: out1.ManufacturingModelMetric.

  out1.ManufacturingModelMetric(a,b,c,d,e) :- AVM2CyPhyML(_,
      out1.ManufacturingModelMetric(a,b,c,d,e)).
```

Specifies mapping of avm.simulink.SimulinkModel to CyPhyML SignalFlowModel.

```
AVM2CyPhyML(
    avmSimulinkModel,
    out1.SignalFlowModel(
      "",         //name
      author,     //Author
      "",         //Class
      filePath,   //FilePathWithinResource
      notes,      //Notes
      "",         //objectID
      aID         //iObjectID
    )
  ) :-
    avmSimulinkModel is in1.avm_simulink_SimulinkModel(
      aID,      //aObjectID
      author,   //Author
      filePath, //FilePathWithinResource
      notes,    //Notes
      _,        //Path
      _         //UsesResource
    )
  .

  out1.ComponentTypeToSignalFlowModelContainment(outComp, outModel) :-
    in1.Contains(inComp, inModel),
    AVM2CyPhyML(inComp, outComp), outComp:
        out1.ComponentTypeToSignalFlowModelContainment_Container,
```

```
    AVM2CyPhyML(inModel, outModel), outModel: out1.SignalFlowModel.

  out1.SignalFlowModel(a,b,c,d,e,f,g) :- AVM2CyPhyML(_,
      out1.SignalFlowModel(a,b,c,d,e,f,g)).
```

Specifies mapping of avm.simulink.InputSignal to CyPhyML InSignal.

```
AVM2CyPhyML(
    avmInSig,
    out1.InSignal(
      name, //name
      "",   //Description
      "",   //objectID
      aID   //iObjectID
    )
  ) :-
    avmInSig is in1.avm_simulink_InputSignal(
      aID,  //aObjectID
      _,    //DataType
      _,    //Definition
      _,    //ID
      _,    //Locator
      name, //Name
      _     //Notes
    )
  .

  out1.SignalFlowModelToIOSignalContainment(outModel, outPort) :-
    in1.Contains(inModel, inPort),
    AVM2CyPhyML(inModel, outModel), outModel:
        out1.SignalFlowModelToIOSignalContainment_Container,
    AVM2CyPhyML(inPort, outPort), outPort: out1.InSignal.

  out1.InSignal(a,b,c,d) :- AVM2CyPhyML(_, out1.InSignal(a,b,c,d)).
```

Specifies mapping of avm.simulink.OutputSignal to CyPhyML OutSignal.

```
AVM2CyPhyML(
    avmOutSig,
    out1.OutSignal(
      name, //name
      "",   //Description
      "",   //objectID
      aID   //iObjectID
    )
  ) :-
    avmOutSig is in1.avm_simulink_OutputSignal(
      aID,  //aObjectID
      _,    //DataType
      _,    //Definition
      _,    //ID
      _,    //Locator
      name, //Name
      _     //Notes
    )
  .

  out1.SignalFlowModelToIOSignalContainment(outModel, outPort) :-
    in1.Contains(inModel, inPort),
    AVM2CyPhyML(inModel, outModel), outModel:
        out1.SignalFlowModelToIOSignalContainment_Container,
    AVM2CyPhyML(inPort, outPort), outPort: out1.OutSignal.

  out1.OutSignal(a,b,c,d) :- AVM2CyPhyML(_, out1.OutSignal(a,b,c,d)).
```

Specifies mapping of avm.simulink.Parameter to CyPhyML SF_Parameter.

```
AVM2CyPhyML(
    avmParm,
    out1.SF_Parameter(
      "",     //name
      "",     //RefId
      value,  //Value
      "",     //objectID
      aID     //iObjectID
    )
  ) :-
    avmParm = in1.avm_simulink_Parameter(
      aID, //aObjectID
      _,   //Locator
      _    //Notes
    ),
    in1.ValueContainerValue(avmParm, value)
  .

  out1.SignalFlowModelToParameterContainment(outModel, outParmRef) :-
    in1.Contains(inModel, inParm),
    AVM2CyPhyML(inModel, outModel), outModel:
        out1.SignalFlowModelToParameterContainment_Container,
    AVM2CyPhyML(inParm, outParm), outParm: out1.SF_Parameter, outParmRef is
        out1.SF_ParameterRef(_, outParm, _, _).

  out1.SF_Parameter(a,b,c,d,e), out1.SF_ParameterRef("", sfParm, "", 0) :-
      AVM2CyPhyML(_, sfParm), sfParm = out1.SF_Parameter(a,b,c,d,e).
```

6.2.1.3 Design Mapping

Specifies mapping of avm.ComponentInstance to CyPhyML ComponentRef. The mapping requires that the ComponentInstance refers to a Component in the model. An error message is created otherwise.

```
AVM2CyPhyML(
    avmCompInst,
    out1.ComponentRef(
      name,    //name
      outComp, //target
      "",      //Cardinality
      idInt,   //ID
      "",      //InstanceGUID
      "",      //NumAssociatedConfigs
      "",      //objectID
      aID      //iObjectID
    )
  ) :-
    avmCompInst is in1.avm_ComponentInstance(
      aID,    //aObjectID
      compID, //ComponentID
      idStr,  //ID
      name    //Name
    ),
    idInt = toNatural(idStr),
    AVM2CyPhyML(in1.avm_Component(_, compID, _, _, _), outComp), outComp: out1.Component
  .

  err1.Error(msg) :-
    in1.avm_ComponentInstance(_, compID, _, _), no in1.avm_Component(_, compID, _, _, _),
    msg = strJoin(strJoin("No␣match␣for␣Component␣ID␣'", compID), "'␣in␣Component␣
        Model.").

  out1.DesignContainerToComponentContainment(outCont, outComp) :-
    in1.Contains(inCont, inComp),
    AVM2CyPhyML(inCont, outCont), outCont:
        out1.DesignContainerToComponentContainment_Container,
    AVM2CyPhyML(inComp, outComp), outComp: out1.Component.
```

```
    out1.ComponentRef(a,b,c,d,e,f,g,h) :- AVM2CyPhyML(_,
        out1.ComponentRef(a,b,c,d,e,f,g,h)).
```

An error message is created for any element in the source model that falls into the category of types that we map, but was not mapped by the transform, and for any element that is mapped to more than one element in the CyPhyML model.

```
err1.Error(msg) :-
      obj is MappedTypes, no AVM2CyPhyML(obj, _),
      msg = strJoin(strJoin("Object␣", toString(obj)), "␣did␣not␣satisfy␣the␣necessary␣
          conditions␣to␣map␣to␣CyPhyML.").

    err1.Error(msg) :-
      obj is MappedTypes, AVM2CyPhyML(obj, a), AVM2CyPhyML(obj, b), a != b,
      msg = strJoin(strJoin("Object␣", toString(obj)), "␣mapped␣to␣more␣than␣one␣object␣
          in␣CyPhyML.").
```

Chapter 7

Revision History

- Version 2.5
  - Schema changes
    Added "Supercedes" list to Component
    Added "ValueNode", "Formula", "SimpleFormula", and "ComplexFormula" classes
    DerivedValue now refers to a "ValueNode"
    XPosition and YPosition added to several classes
    ITAR tag has been simplied – its presence alone indicates ITAR status
    Added DoDDistributionStatement
    Added "Name" attribute to avm.DomainModel
- Version 2.4
  - Schema changes
    Component.ID is now xs:string, instead of xs:ID
    DomainModel.FilePathWithinResource has been removed
    ParametricValue.AssignedValue is now a containment relation with a Value object (used to be association)
    Added "SurfaceReverseMap" association to avm.cad.Plane
  - Domain Documentation changes
    Added Modelica Modeling Guidelines
    Added iFAB documentation in native format
    Updated CAD Modeling Guidelines
- Version 2.3
  - Updated and expanded documentation
  - No schema changes