2023 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION EUROPE 10 YEAR ANNIVERSARY

SysML v2
An overview with SysMD demonstration

Christoph Grimm, Axel Ratzke, Sebastian Post, Hagen Heermann, Johannes Koch
University of Kaiserslautern, Chair of Cyber-Physical Systems

RPTU Chair of Cyber-Physical Systems
Prof. Dr. Christoph Grimm

accellera SYSTEMS INITIATIVE

Acknowledgements

Special thanks to SST for the open standard and its documentation
& Sanford Friedenthal and Ed Seidewitz for feedback and corrections.

The work was supported by German BMBF and EU within the projects

- Arrowhead Tools,
- GENIAL!, and
- KI4BoardNet.

Objectives of tutorial

1. Understand *why, how,* and *for what* to use SysML v2 in development
   - Use cases
   - Patterns / Anti patterns
2. Get overview of SysML v2 **ecosystem**, not only the language!
   - Know about KerML, SysML v2 textual, Rest API
   - Know about features for modeling tests and use cases
   - Be able to create structural & parameterized model
3. Give impression of future Systems Engineering by example SysMD Notebook
   - Support of MBSE by constraint propagation
   - Integration in HW/SW development Verification & Validation processes

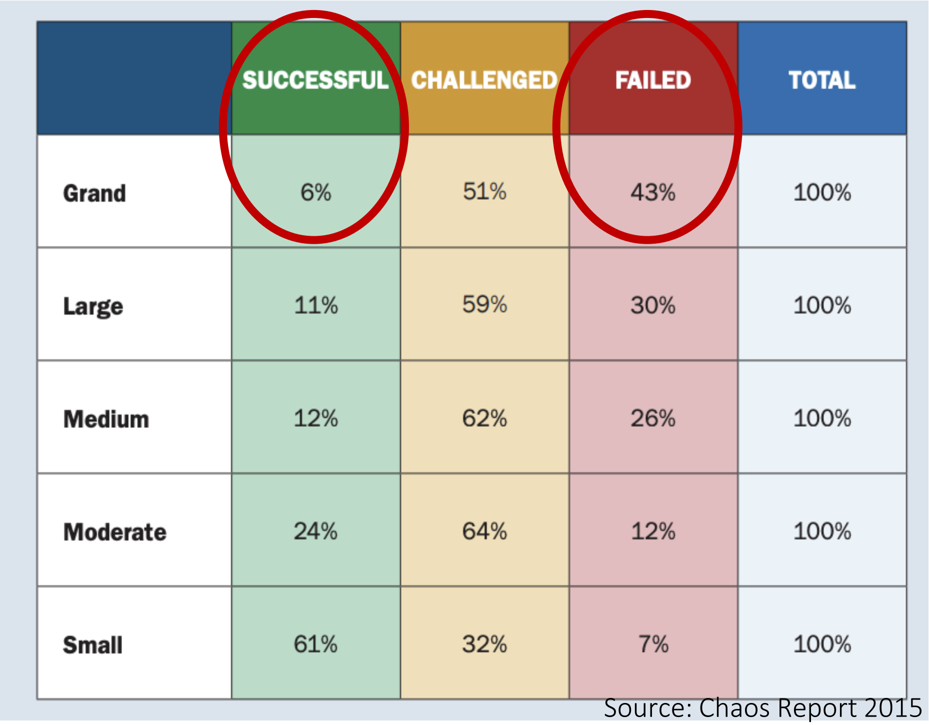
Why, why, why?

First known complex project reported by literature [Genesis 11:1–9] is tower of Bable:

"… let's confuse their language, so that they may not understand one another's speech. … and they left off building the city."

Mutual understanding is key to complex, heterogeneous systems.

Why, why, why?



	SUCCESSFUL	CHALLENGED	FAILED	TOTAL
Grand	6%	51%	43%	100%
Large	11%	59%	30%	100%
Medium	12%	62%	26%	100%
Moderate	24%	64%	12%	100%
Small	61%	32%	7%	100%

Source: Chaos Report 2015

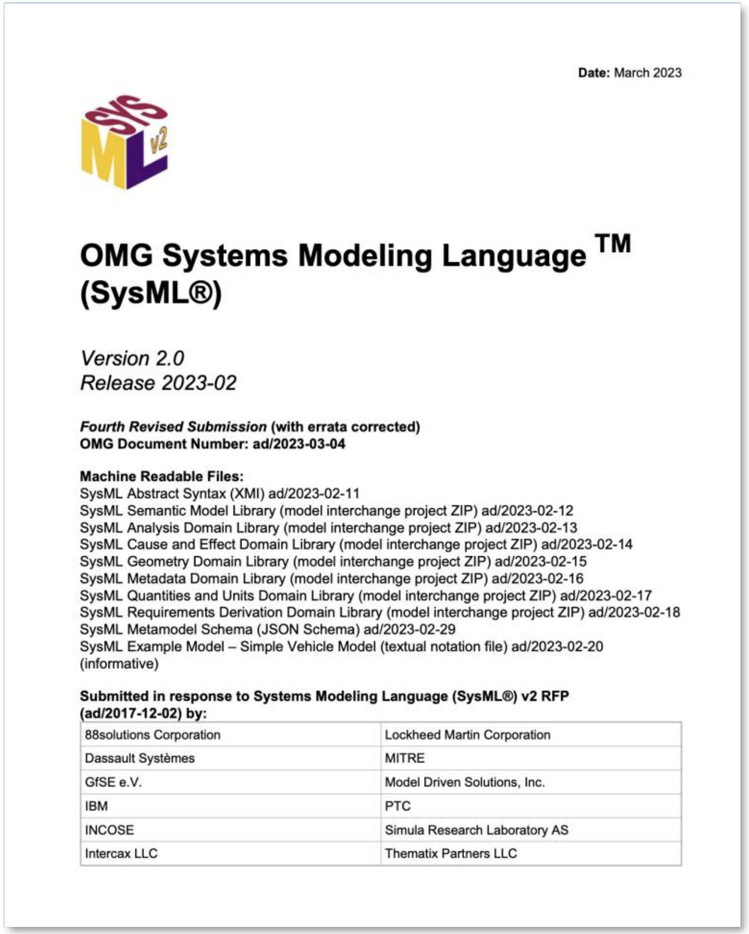
Many “big” projects fail, in all domains incl. HW/SW

- Requirements, use cases, specification
  - are incomplete, unknown,
  - not well understood in beginning,
  - change during development (or operation),
  - have inconsistencies.
- Above issues are expensive to fix lately
  - SysML v2 offers standardized solution
  - US DoD might request SysML v2 models

SysML v2: An Overview with Demonstration

1. **The SysML v2 Eco-System**
2. KerML, the Metamodel
3. SysML v2 textual
4. REST API for model exchange
5. Outlook

What is SysML?



Date: March 2023

OMG Systems Modeling Language ™ (SysML®)

Version 2.0
Release 2023-02

Fourth Revised Submission **(with errata corrected)**
OMG Document Number: ad/2023-03-04

Machine Readable Files:
SysML Abstract Syntax (XMI) ad/2023-02-11
SysML Semantic Model Library (model interchange project ZIP) ad/2023-02-12
SysML Analysis Domain Library (model interchange project ZIP) ad/2023-02-13
SysML Cause and Effect Domain Library (model interchange project ZIP) ad/2023-02-14
SysML Geometry Domain Library (model interchange project ZIP) ad/2023-02-15
SysML Metadata Domain Library (model interchange project ZIP) ad/2023-02-16
SysML Quantities and Units Domain Library (model interchange project ZIP) ad/2023-02-17
SysML Requirements Derivation Domain Library (model interchange project ZIP) ad/2023-02-18
SysML Metamodel Schema (JSON Schema) ad/2023-02-29
SysML Example Model – Simple Vehicle Model (textual notation file) ad/2023-02-20 (informative)

Submitted in response to Systems Modeling Language (SysML®) v2 RFP (ad/2017-12-02) by:

88solutions Corporation	Lockheed Martin Corporation
Dassault Systèmes	MITRE
GfSE e.V.	Model Driven Solutions, Inc.
IBM	PTC
INCOSE	Simula Research Laboratory AS
Intercax LLC	Thematix Partners LLC

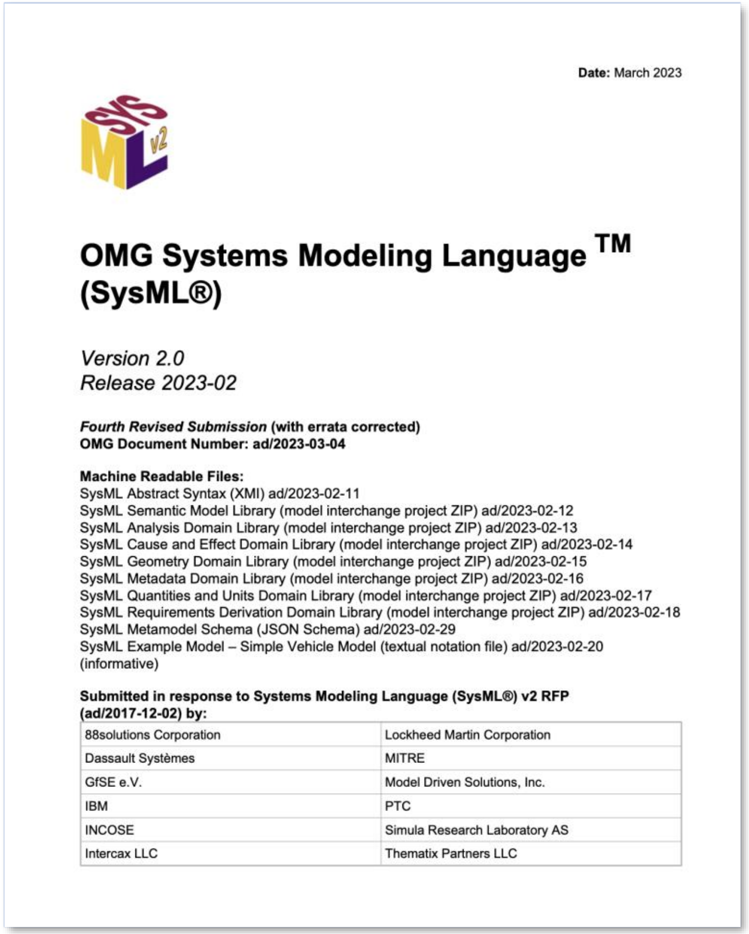
SysML **("Systems Modeling Language")** is a standard for Model-Based Systems Engineering

- Requirements
- Specification
- Use cases
- Test cases

NOT: "Design", NOT "Behavioral modeling"

- But 1: use cases, verification use cases, … should use behavior
- But 2: exchange and versioning of data

What is SysML **“v2”**?



Date: March 2023

OMG Systems Modeling Language ™ (SysML®)

Version 2.0
Release 2023-02

Fourth Revised Submission **(with errata corrected)**
OMG Document Number: ad/2023-03-04

Machine Readable Files:
SysML Abstract Syntax (XMI) ad/2023-02-11
SysML Semantic Model Library (model interchange project ZIP) ad/2023-02-12
SysML Analysis Domain Library (model interchange project ZIP) ad/2023-02-13
SysML Cause and Effect Domain Library (model interchange project ZIP) ad/2023-02-14
SysML Geometry Domain Library (model interchange project ZIP) ad/2023-02-15
SysML Metadata Domain Library (model interchange project ZIP) ad/2023-02-16
SysML Quantities and Units Domain Library (model interchange project ZIP) ad/2023-02-17
SysML Requirements Derivation Domain Library (model interchange project ZIP) ad/2023-02-18
SysML Metamodel Schema (JSON Schema) ad/2023-02-29
SysML Example Model – Simple Vehicle Model (textual notation file) ad/2023-02-20 (informative)

Submitted in response to Systems Modeling Language (SysML®) v2 RFP (ad/2017-12-02) by:

88solutions Corporation	Lockheed Martin Corporation
Dassault Systèmes	MITRE
GfSE e.V.	Model Driven Solutions, Inc.
IBM	PTC
INCOSE	Simula Research Laboratory AS
Intercax LLC	Thematix Partners LLC

Version 2 is **not** a simple “update” of v1.X …

- (Mostly) new, but not entirely different

SysML v2 standard includes

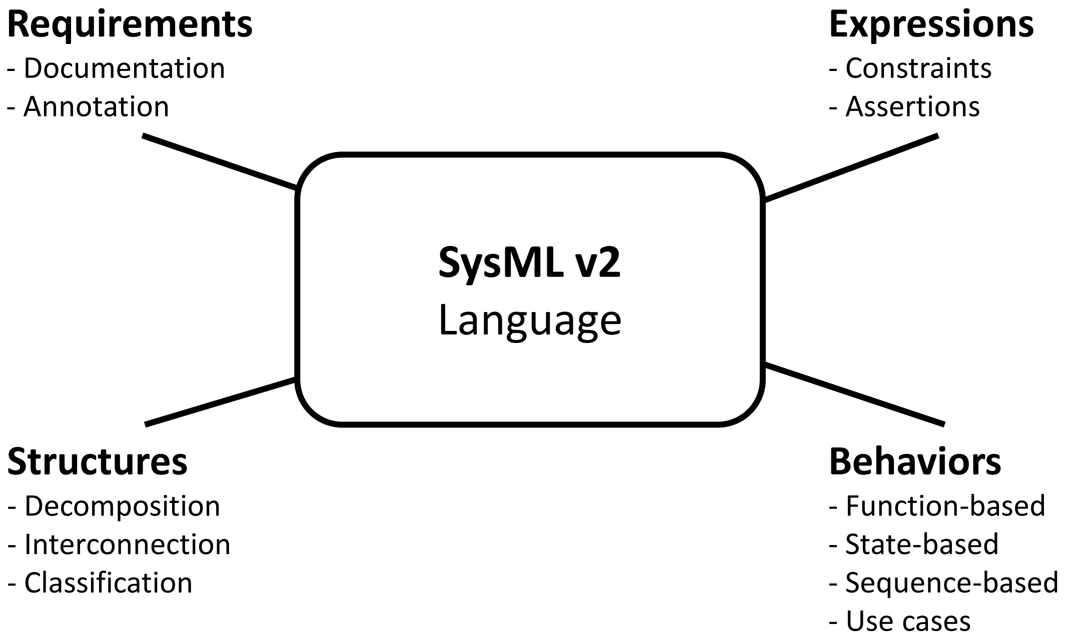
- KerML, a new meta-model
- SysML v2 diagrams
- SysML v2 textual modeling language
- SysML v2 REST API

Some Use Cases and SysML v2 Features

Document requirements

Specify system functions

Specify system structures



Requirements
- Documentation
- Annotation

Expressions
- Constraints
- Assertions

SysML v2
Language

Structures
- Decomposition
- Interconnection
- Classification

Behaviors
- Function-based
- State-based
- Sequence-based
- Use cases

Analyze requirements

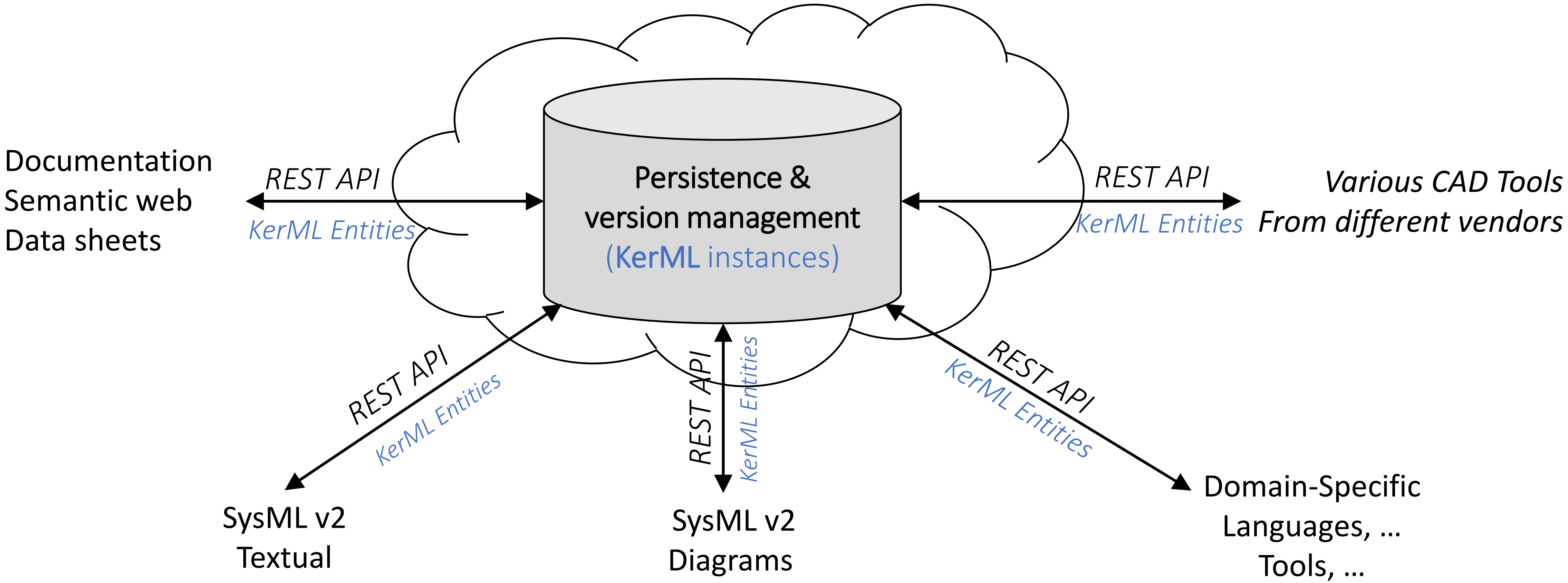
Analyze effect chains

Model use case scenarios & tests

Elements of the SysML v2 eco-system

1. KerML, Kernel modeling language
   - Basic, generic model elements from which all models are built
     → Interoperability, extensibility
2. SysML v2, based on KerML
   - SysML v2 Diagrams
   - SysML v2 Textual notation
3. API
   - Exchange of KerML Elements e.g. REST API, OSLC

The SysML v2 Eco-System (a vision)



Persistence &
version management
(KerML instances)

Documentation
Semantic web
Data sheets

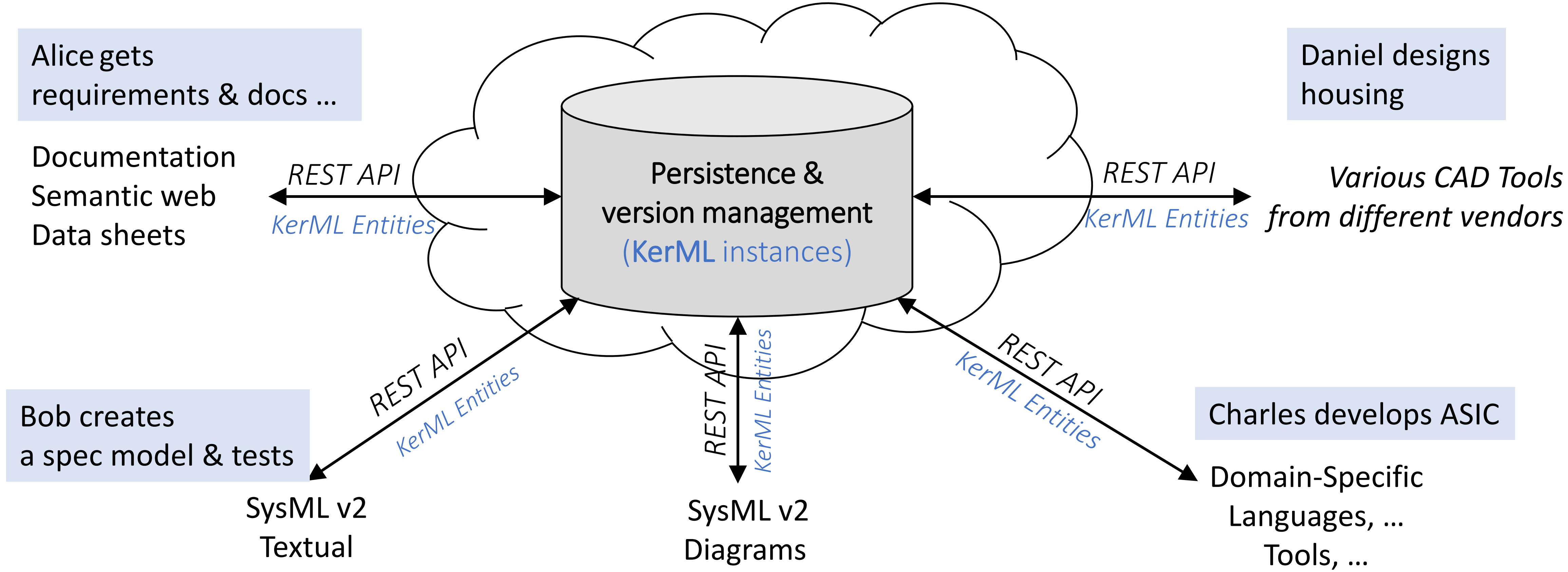
Various CAD Tools
From different vendors

SysML v2
Textual

SysML v2
Diagrams

Domain-Specific
Languages, …
Tools, …

The SysML v2 Eco-System (a vision)



Three Patterns/Anti-Patterns …

Anti-Pattern (at spec-level)

- Non-specific natural language in documents
  - “enough”, “more”, “better”; “as in last project”
- Create only models, or separate from docs
  - Excludes many stakeholders
  - Leads to inconsistencies doc vs. model
- Start “design” by creating behavioral models
  - Reduces solution space for domain experts
  - Creates wasted time for not-needed modeling

Better

- Derive *concrete parameters* for performances
  - X is at least 50, “y more than 60”, “z must be 20”
- Link documents with models
  - Single source of truth for all
- Describe test-cases & use cases by behavior
  - Generate skeletons for domain-specific tools
  - Round-trip for parameters

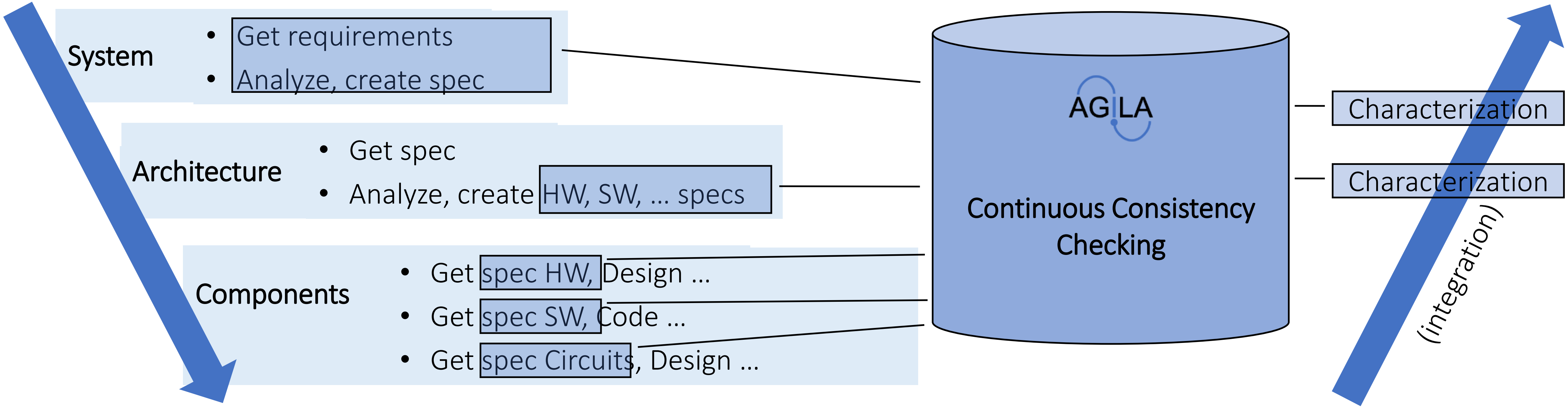
SysML v2 tools

- SysML v2 reference implementation (Java)
  https://github.com/Systems-Modeling/SysML-v2-Release
  - Good for trying and learning SysML v2: reference, comprehensive

➡ SysMD Notebook (Kotlin)
  - Integration MD documents, tables, … & Model
  - Constraint propagation permits analysis and consistency checking
  - Be aware of limitations: much of KerML + little of SysML + *built-in profile for ranges*

… and a number of vendors likely working on commercial tools

SysMD in HW/SW System Design



System
- Get requirements
- Analyze, create spec

Architecture
- Get spec
- Analyze, create HW, SW, … specs

Components
- Get spec HW, Design …
- Get spec SW, Code …
- Get spec Circuits, Design …

AGILA

Continuous Consistency Checking

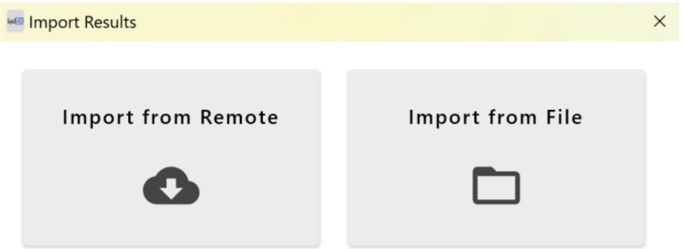
Characterization

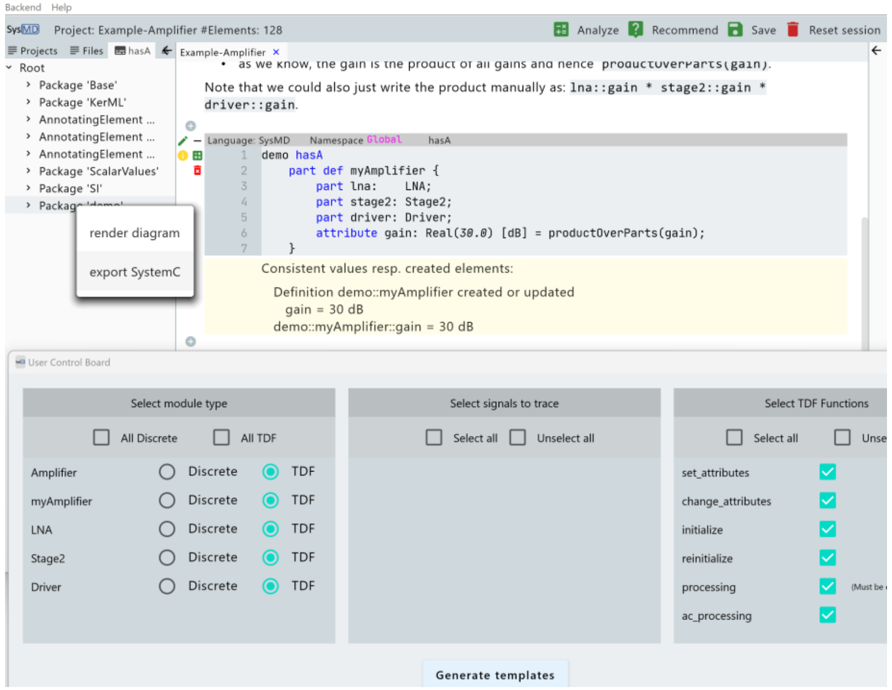
Characterization

(integration)

SysMD Demonstration 1: SystemC Roundtrip

- Overview Documentation (MD, Latex) + Model integration
- SystemC Code generation
- Roundtrip after characterization





Project: Example-Amplifier #Elements: 128

• as we know, the gain is the product of all gains and hence productOverParts(gain).

Note that we could also just write the product manually as: `lna::gain * stage2::gain * driver::gain`.

Language: SysMD Namespace Global hasA

```
demo hasA
    part def myAmplifier {
        part lna:    LNA;
        part stage2: Stage2;
        part driver: Driver;
        attribute gain: Real(30.0) [dB] = productOverParts(gain);
    }
```

Consistent values resp. created elements:

Definition demo::myAmplifier created or updated
gain = 30 dB
demo::myAmplifier::gain = 30 dB

Select module type

□ All Discrete □ All TDF

Amplifier	○ Discrete	● TDF
myAmplifier	○ Discrete	● TDF
LNA	○ Discrete	● TDF
Stage2	○ Discrete	● TDF
Driver	○ Discrete	● TDF

Select signals to trace

□ Select all □ Unselect all

Select TDF Functions

□ Select all □ Unse

set_attributes	☑	
change_attributes	☑	
initialize	☑	
reinitialize	☑	
processing	☑	(Must be
ac_processing	☑	

SysML v2: An Overview with Demonstration

1. The SysML v2 Eco-System
2. **KerML, the Metamodel**
3. SysML v2 textual
4. REST API for model exchange
5. Outlook

SysML v2 Language Architecture

Layer	Adds	Classes added
KerML Root	Syntactic structure	Element, Relationship, Namespace, Annotation, Membership, …
KerML Core	Semantic by logic	Type, Feature, Multiplicity, …
KerML Kernel	Semantic library	Class, Datatype, Expression, Package, Association, Connector, Behavior, …
SysML v2	Domain-specific library	Part, Attribute, Port, Interface, Connection, Constraint, Assertion, Requirement, Variation, View, …

- KerML (and SysML v2) Models represented & exchanged by instances of these classes (e.g. XML, JSON, …)
- Also, concrete (textual) notations: human user-friendly
- Based on KerML, other DSL can be developed i.e. targeting tool interoperability & model exchange!

KerML Textual: Literals

Names

- Start with letter or _, then letters or numbers: `name1`, `_123`
- Unrestricted names: `‘This is a valid name‘` *(no backslash, no single quote in name)*

Qualified names

- Give path from an *Element* to another *Element* (name):
  Inside `ScalarValues`, the Element with name `Real`: ScalarValues::Real

Number; Boolean literals; Strings

- `12.0 e -10`; `true`, `false`; `“this is a string”`

Elements & Relationships

Element (common base class)

- elementId (UUID; unique for all commits)
- declaredName
- declaredShortName
- owner *Identifications*
- ownedElement *Identifications*

```
type <t1> ‘type no.1’;
namespace <n1> namespace1;
```

Relationship (for all relations)

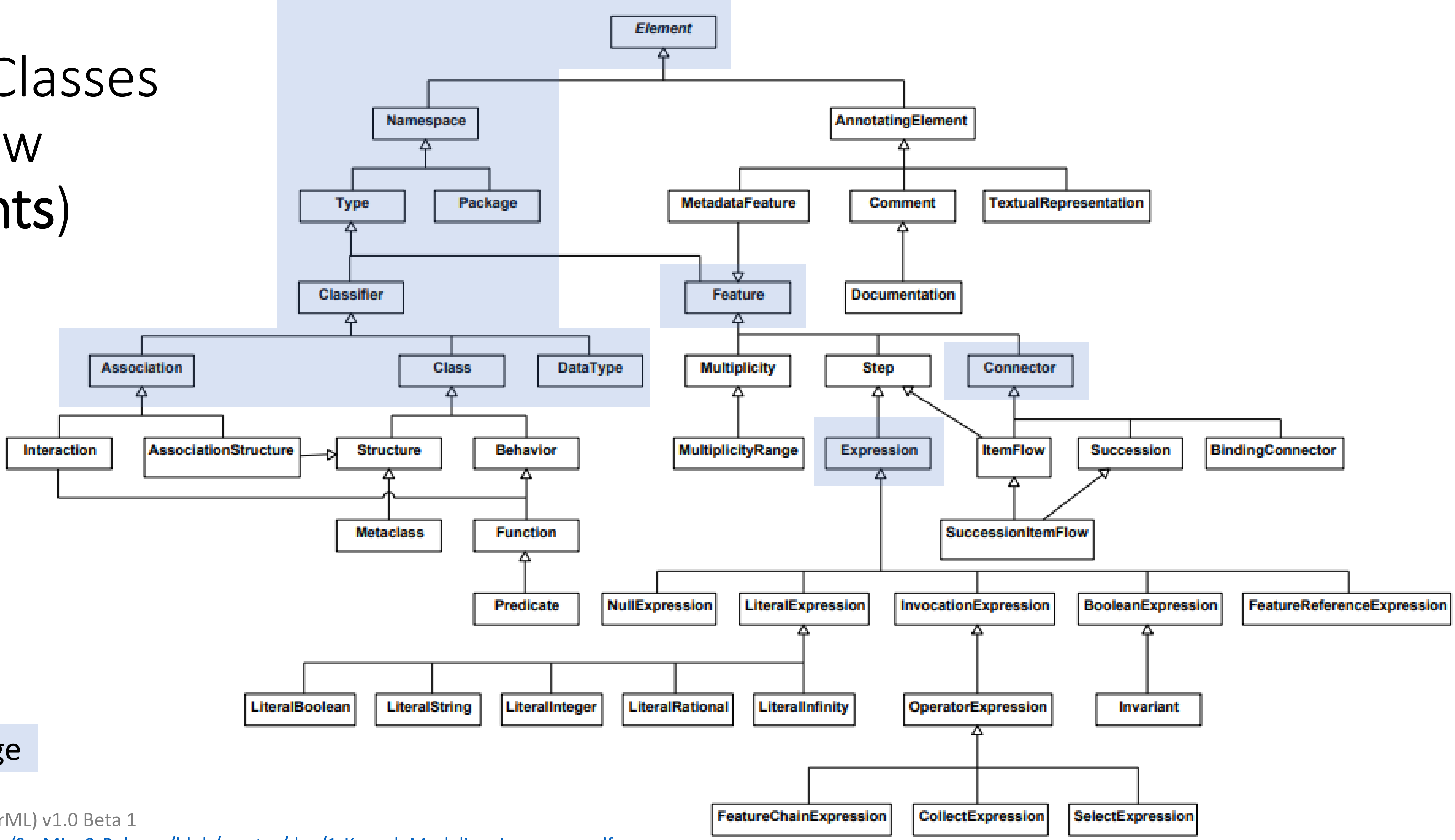
- source *Identifications*
- target *Identifications*

- *at least two related elements*
- *not necessarily directed; both related elements can be source or target*

```
dependency d from t1 to n1;
```

Read more in:
https://github.com/Systems-Modeling/SysML-v2-Release/blob/master/doc/1-Kernel_Modeling_Language.pdf

KerML Classes Overview (**Elements**)



Tutorial coverage

Figure: Kernel modeling language (KerML) v1.0 Beta 1
https://github.com/Systems-Modeling/SysML-v2-Release/blob/master/doc/1-Kernel_Modeling_Language.pdf

KerML Classes Overview (**Relationships**)

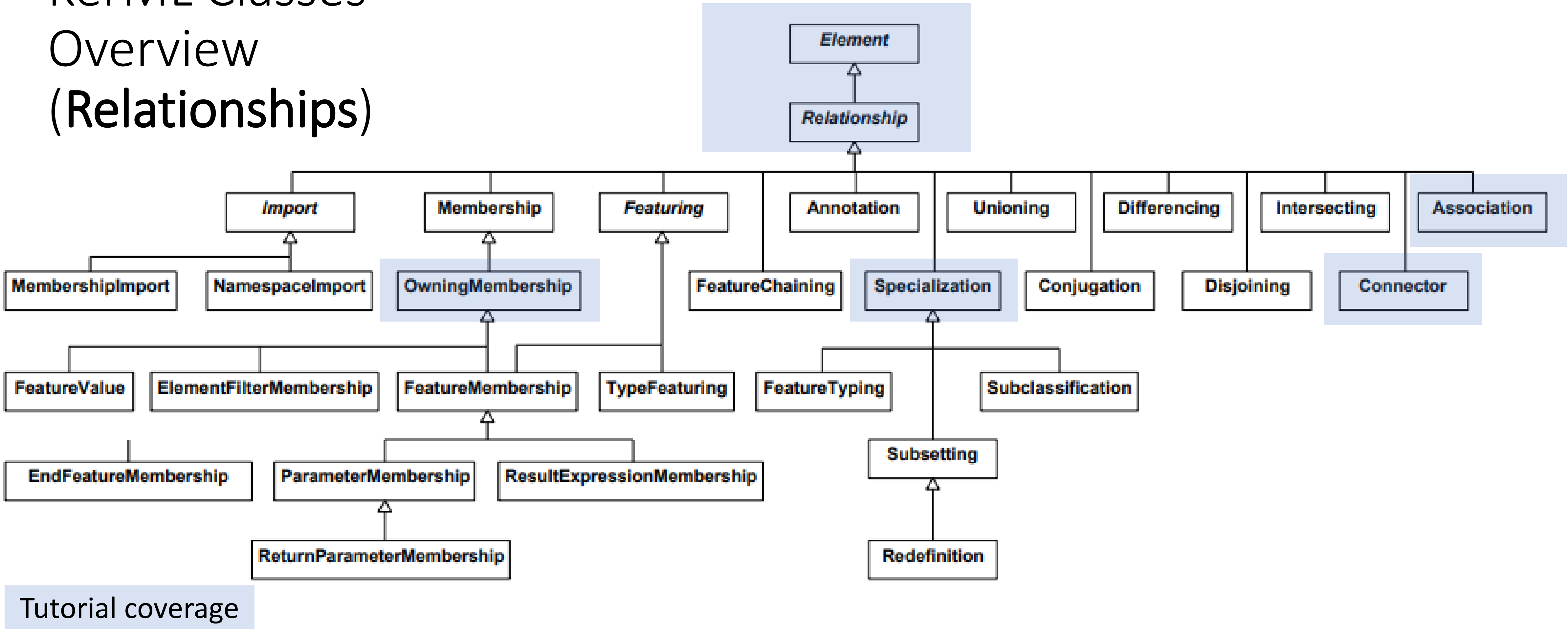
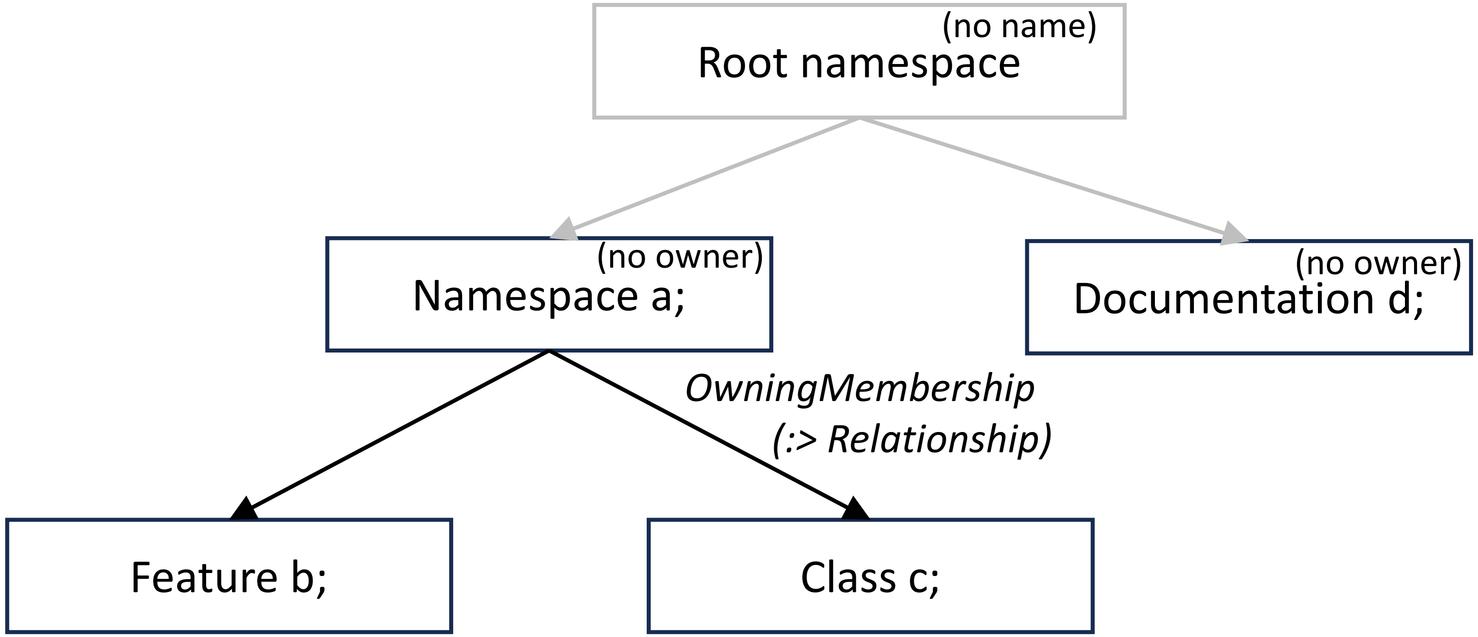


Figure: Kernel modeling language (KerML) v1.0 Beta 1
https://github.com/Systems-Modeling/SysML-v2-Release/blob/master/doc/1-Kernel_Modeling_Language.pdf

Ownership (general)



```
namespace a {
  feature b;
  class c;
}
doc d;
```

- Curly braces ~ ownership hierarchy
- All elements are owned by other element except “root namespace”.
- Some elements imply additional ones.
- Owned elements are deleted if owner is deleted.

Package, Import

Package (and Namespace & subclasses thereof)

- structures model hierarchically,
- permits lookup of elements by its (short)name (“name resolution”),
- can **import** of other elements or namespaces,
- visibility of elements in a namespace can be restricted by **public**, **private**, **protected.**

```
public package p2 {
  class c2;
}

package p {
  class c {
    // Without import:
    // feature f : p2::c2;
    import p2::*;
    // or: import p2::c2;
    feature f: c2;
  }
}
```

Classifiers (DataType, Class)

Classifier models **similarities** between abstract sets of things or data.

- The most general Classifier is *Anything.*
- Classifiers own Relationship *Specialization* between *general* and *specific* class.
- The specific class inherits *public* and *private* memberships from general class.
- Shortcut for specializes: “:>”

```
class Vehicle specializes Base::Anything {
  feature wheels: Wheels;
  feature engine: Engine [0 .. 1];
}

class Car specializes Vehicle {
  // inherits wheels, engine
}
```

→ Read more on type-relationships!

- *Conjugation, Disjoining, …*

Features

Feature is typed by classifier; describes **things** and how they are related:

- *In classifiers (“class featuring”):*
  which things do all things of a class “have”?
- *else:*
  decomposition of a thing into things.

- Furthermore, features
  - may be *redefined*.
  - can be *subsets* of other features.
  - Have direction: *in, out, inout.*
  - Be *abstract*, *composite*, *portion*, …

```
// Common features of all Vehicles
class Vehicle specializes base::Anything {
  feature wheels: Wheels [2 .. *];
  feature engine: Engine [0 .. 1];
}

// Features of a concrete vehicle
feature myCar: Vehicle {
  feature redefines wheels: Wheels[4];
  feature frontWheels subsets wheels;
}
```

Association, Connector

Associations classify relationships by giving source and/or target classes, by end features:

```
assoc Wire { // BinaryLink by default
  // source type & name
   end feature startOfWire: Device;
  // target type & name
   end feature endOfWire: Device;
}
```

Connectors represent concrete relationships, typed by an Association

```
feature sensor1: Device;
feature controller1: Device;

connector wire1: Wire
    from sensor1
    to controller1;
```

Function, Expression

Functions model abstract dependencies between values

- Cannot be evaluated

```
function AreaComputation {
  import ScalarValues::*;
  in w: Real;
  in l: Real;
  return area: Real = w*l;
}
```

Expressions model a concrete dependency between input and output values, typed by a function.

- Can be evaluated if dependent variables are bound to a literal value

```
feature w1: ScalarValues::Real;
feature l1: ScalarValues::Real;
expr areaW1L1: AreaComputation {
  in w; in l; return area;
}
```

SysMD Notebook & Solver

SysMD Notebook extends the ability of KerML/SysML v2 to “execute” models

- Computation on symbolic or abstract values instead of concrete values
- No need to have literal values, instead:
  - Boolean values: 4-Valued logic: unknown, true, false, infeasible
  - Real values: Ranges: Real = [*], [‘lower bound’ .. ‘upper bound’], Empty (no Real)
  - Integer values: Integer ranges, likewise

Live Demonstration 2: SysMD Solver

- Solver on Reals with invariants
  - Simple example: Box with constrained sides and volume and Units

- Solver on Booleans and mixed Boolean/Real
  - Simple example: Satisfiable, Unsatisfiable Boolean combinations
  - Simple example: Predicates with Real inequations & (Un-)Satisfiable inequations
  - Mass roll up in vehicle (bottom-up, top-down, multiplicities)

SysML v2: An Overview with Demonstration

1. The SysML v2 Eco-System
2. KerML, the Metamodel
3. **SysML v2 textual**
4. REST API for model exchange
5. Outlook

SysML v2 Language Architecture

Layer	Adds	Classes added
KerML Root	Syntactic structure	Element, Relationship, Namespace, Annotation, Membership, …
KerML Core	Semantic by logic	Type, Feature, Multiplicity, …
KerML Kernel	Semantic library	Class, Datatype, Expression, Package, Association, Connector, Behavior, …
SysML v2	Domain-specific library	Part, Attributes, Ports, Requirement, Constraint, Assertion, Usages, Connection, Views, …

- KerML (and SysML v2) models represented & exchanged by instances of these classes (e.g. XML, JSON, …)
- Also, concrete (textual) notations: human user-friendly
- Based on KerML, other DSL can be developed i.e. targeting tool interoperability & model exchange!

Part Definition, Part Usage

Part definition creates a *class* of parts.

Part usage creates a *feature*.

Part references create a *reference* to a feature that exists independently from the part.

```
part def Vehicle {
  attribute mass: ISQ: Mass;
  part wheels: Wheel [1 .. *];
  ref part driver: Person;
}
part def Car :> Vehicle;


part myVehicle: Car {
  attribute redefines
    mass: ISQ::Mass = 100 [kg];
}
```

Attribute Definition & Usage

Attribute definition defines a *DataType* that can be used to model systems.

Attribute is a kind of *Feature* typed by a DataType.

- Can be bound to an expression
- Expression can be computed for e.g. analysis

```
attribute def position {
  attribute x: ISQ::Length;
  attribute y: ISQ::Length;
  attribute z: ISQ::Length;
}

part def Car {
  attribute mass: ISQ: Mass
    = 10.0 * wheels.mass;
  part wheels: Wheel [4];
  ref part driver: Person;
}
```

Constraints & Assertions (Both: Usage & Definition)

Constraint is a kind of Boolean expression that can be satisfied or not.

- E.g. some performance we like to have, but that is not guaranteed to be satisfied.
- Note: Definition also possible.

Assertion is a kind of invariant that is always satisfied.

- E.g. a natural law, very hard constraint.
- *Assertions can also specify systems of (e.g. DAE or in-) equations.*
- Note: Definition also possible.

```
part vehicle: Vehicle;
constraint enoughPower {
  vehicle.power > 500.0 [SI::kW]
}

assert constraint maxMass {
  vehicle.mass < 200.0 [SI::t]
}
// TimeOf(…), Duriation(…)
// for time constraints in behavior
```

Requirement Definition & Usage

Requirement definition introduces a class of *constraint definitions*, and can have e.g.

- doc, attributes, features
- *subject*; a feature about which and in whose scope the requirement is formulated
- *assume constraint*
- *require constraint*

Requirement models a *concrete requirement*.

→Learn more: Requirements can be

- *Grouped* and structured
- *Satisfied* by parts to link requirements and design

```
requirement def Slewrate {
  doc /* Max rate of change */
  subject opAmp: OpAmp;
  attribute minRate: Quantity[V/ms];
  require constraint {
    opAmp.slewrate >= minSlewrate
  }
}

requirement slewrate: Slewrate {
  attribute redefine minRate=10 [V/ms];
}
```

Ports, Interfaces

Port (Def) models feature via which a part (definition) makes some of its features available

- Direction of features: in, out, inout
- Referential features!

Interface (Def) models connection between ports

- Can have hierarchy
  → ~ SystemC hierarchical Channels

```
part CPU {
  port clk: Bit {
    attribute fmax: Quantity = 1.0 [GHz];
  }
}

interface clk: Bit
   connect CPU.clk to ClkGen.clk;
```

Behavior (State Machines, …)

Comprehensive set of options to model behavior and synchronization

- State machines
  - entry, state
  - accept … then ..
  - Hierarchy, also parallel
  - Guard & Effect actions
  - …
  - Clocks, Timing constraints

→ **Interaction (~sequence diagrams)**

→ ***(too much for brief tutorial)***

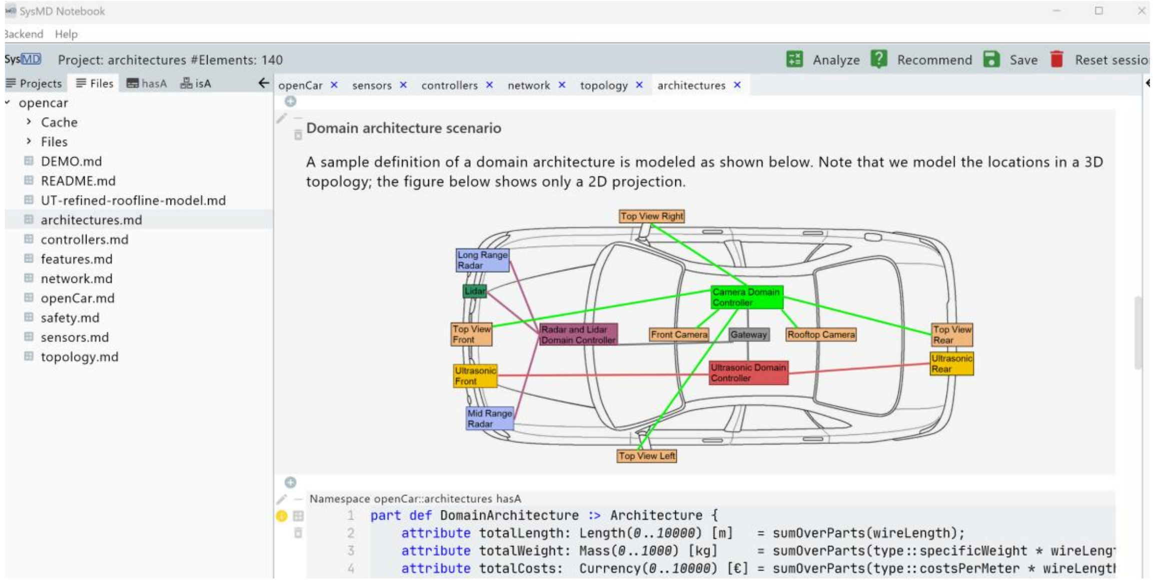
```
state def VehicleStates;

state vehicleStates: VehicleStates {
  entry; then off;
  state off;
  accept VehicleStartSignal then starting;
  state starting;
  accept VehicleOnSignal then on;
  state on;
  accept VehicleOffSignal then off;
}
```

Demonstration 3 - Domain vs. Zonal Architecture

Model of distributed in-car network

- Processors, cable tree, sensors
- Mapping of SW Features to Processors

- Analysis of cable length, cost and weight
- Analysis of performance bottlenecks
  - Latencies
  - Data rates



Domain architecture scenario

A sample definition of a domain architecture is modeled as shown below. Note that we model the locations in a 3D topology; the figure below shows only a 2D projection.

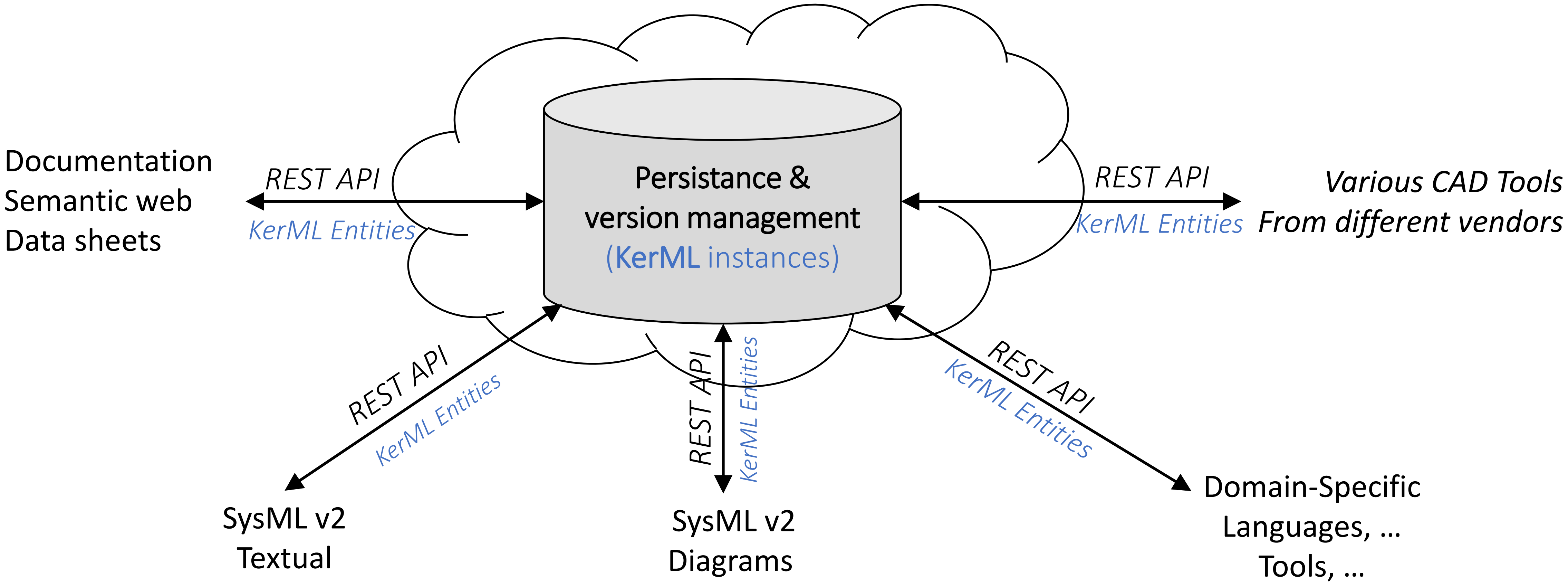
Namespace openCar::architectures hasA

```
part def DomainArchitecture :> Architecture {
    attribute totalLength: Length(0..10000) [m]   = sumOverParts(wireLength);
    attribute totalWeight: Mass(0..1000) [kg]     = sumOverParts(type::specificWeight * wireLeng
    attribute totalCosts:  Currency(0..10000) [€] = sumOverParts(type::costsPerMeter * wireLengtl
```

SysML v2: An Overview with Demonstration

1. The SysML v2 Eco-System
2. KerML, the Metamodel
3. SysML v2 textual
4. **REST API for model exchange**
5. Demonstration SysMD Notebook
6. Outlook

The SysML v2 Eco-System (a vision; reminder)



Persistance & version management (KerML instances)

Documentation
Semantic web
Data sheets

REST API
KerML Entities

Various CAD Tools
From different vendors

REST API
KerML Entities

SysML v2
Textual

REST API
KerML Entities

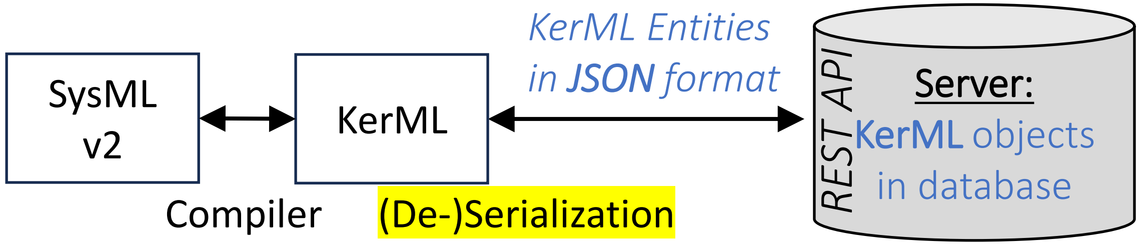
SysML v2
Diagrams

REST API
KerML Entities

Domain-Specific
Languages, …
Tools, …

REST API
KerML Entities

JSON



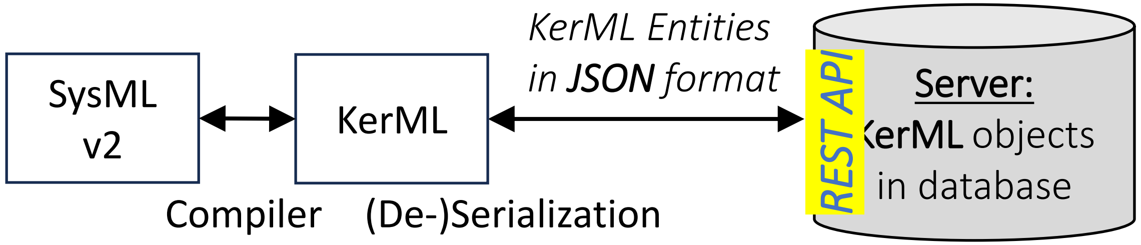
JavaScript Object Notation (JSON)

- Format for exchange of data across platforms
- Commonly used in internet to represent serialized data
- Schema defines structure & fields
- *SysML v2 std. gives Schema for exchange*

UUID 4 or 5 (std. libraries)

```
[
 {
   "@type": "Package",
   "@elementId": "54947df8-0e9e-4471-a2f9-9af509fb5889",
   "name": "myPackage",
   "owner": "13447df8-0145-a451-b2fg-9bf50dfb5784",
   …
 } …
]
```

REST API



SysML v2 std. gives API for different platforms and platform-independent

- Popular platform: REST API (→ Cloud)

Endpoints are URL via which data in e.g. JSON format can be exchanged

`https://mycompany.com/specs/projects/$ID`

Operations with URL

- POST – transfer new, complete element
- PUT – transfer complete existing element
- PATCH – transfer changed fields of element
- GET – get an element
- DELETE – delete an element

SysML v2 Version Management API Services

(Platform Specific Model (PSM))

Element Endpoints

Operations	Endpoint
GET	/projects/<projectId>/commits/ <commitId>/elements
GET	/projects/<projectId>/commits/ <commitId>/elements/<elementId>
GET	/projects/<projectId>/commits/ <commitId>/elements/<elementId>/relationships
GET	/projects/<projectId>/commits/ <commitId>/roots

Marked operation gets all elements of a SysML v2 repository in JSON format (see above!)

- projectId identifies a project – e.g. by GET /projects
- commitId identifies a commit – e.g. by GET /projects/$ID/commits

SysML v2 Version Management API Services

(Platform Specific Model (PSM))

Project Endpoints

Operations	Endpoint
POST, GET	/projects
GET, PUT, DELETE	/projects/<projectId>

Commit Endpoints

Operations	Endpoint
POST, GET	/projects/<projectId>/commits
GET	/projects/<projectId>/commits/<commitId>
GET	/projects/<projectId>/commits/<commitId>/changes
GET	/projects/<projectId>/commits/<commitId>/changes/<changeId>

SysML v2 Version Management API Services

(Platform Specific Model (PSM))

Branch Endpoints

Operations	Endpoint
POST, GET	/projects/<projectId>/branches
GET, DELETE	/projects/<projectId>/branches/<branchId>

Tag Endpoints

Operations	Endpoint
POST, GET	/projects/<projectId>/tags
GET, DELETE	/projects/<projectId>/tags/<tagId>

SysML v2: An Overview with Demonstration

1. The SysML v2 Eco-System
2. KerML, the Metamodel
3. SysML v2 textual
4. REST API for model exchange
5. **Outlook**

Summary

SysML v2 is not just an “update” for SysML

- SysML v2 – *includes also textual language*

- SysML v2 brings a comprehensive ecosystem beyond the modelling language
  - KerML – *also basis for exchange and collaboration across tools and domains*
  - REST API – *allows us to use single source of truth with versioning in the cloud*

- Too much has not been shown in too short tutorial
  - Denotational semantics with formal foundations
  - Get more information and details from GITHUB (link: see last slide)

Outlook

- Many tool vendors work on adaption of tools

- SysMD Notebook as demonstrated will be open-source
    - Small, but growing supported subset
    - Deeper integration of Documents and Models
    - Improvement of solver and its integration with KerML

- *I wonder, what LLM can do with Documentation + linked SysML v2 model* ☺

References & Resources

Github repository of SysML v2 Submission Team (SST)

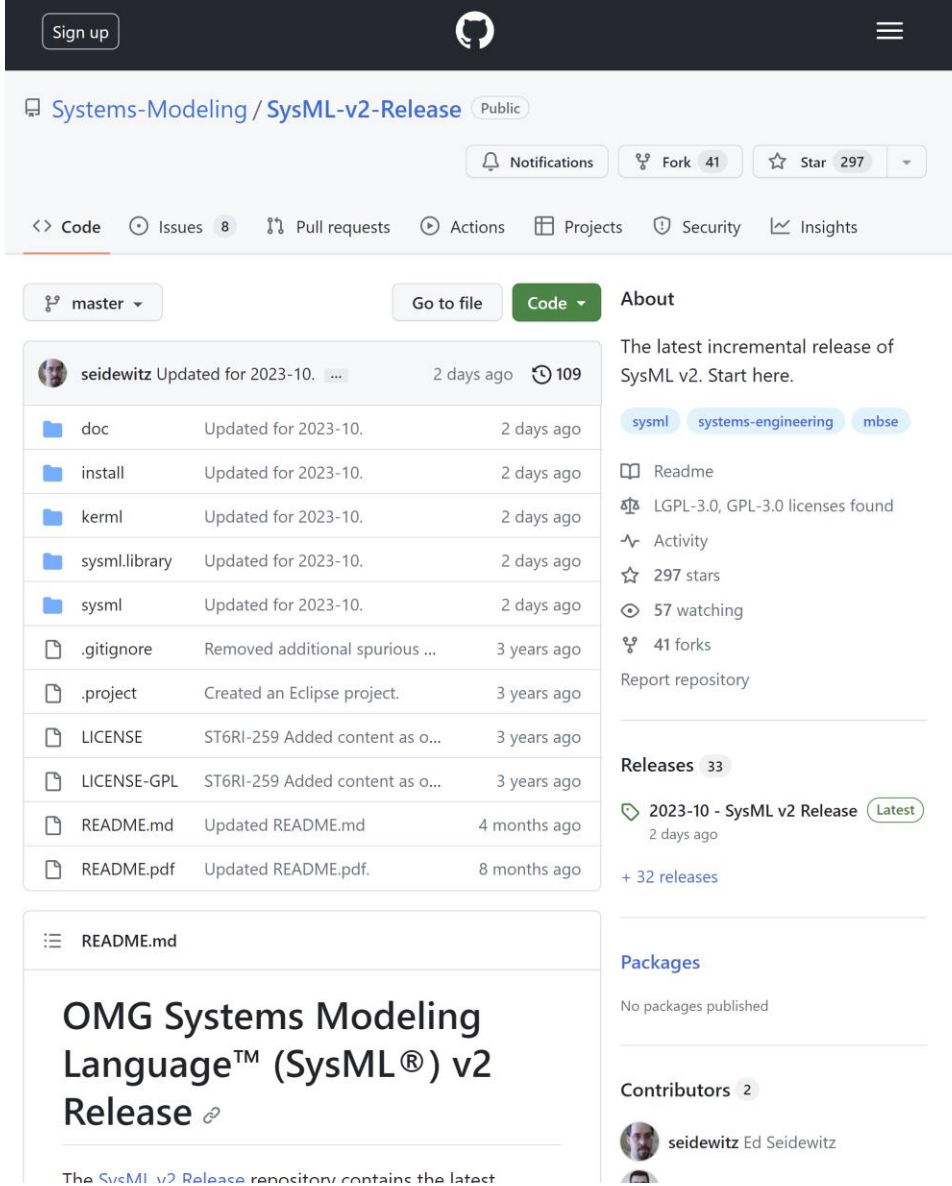
https://github.com/Systems-Modeling/SysML-v2-Release

In “doc”

- Comprehensive introduction to SysML v2 Diagrams
- Comprehensive introduction to SysML v2 Textual
- Detailed documents for KerML, SysMLv2 and API

Google group:

https://groups.google.com/g/sysml-v2-release?pli=1



Sign up

Systems-Modeling / SysML-v2-Release Public

Notifications · Fork 41 · Star 297

Code · Issues 8 · Pull requests · Actions · Projects · Security · Insights

master · Go to file · Code

seidewitz Updated for 2023-10. ... 2 days ago 109

doc	Updated for 2023-10.	2 days ago
install	Updated for 2023-10.	2 days ago
kerml	Updated for 2023-10.	2 days ago
sysml.library	Updated for 2023-10.	2 days ago
sysml	Updated for 2023-10.	2 days ago
.gitignore	Removed additional spurious ...	3 years ago
.project	Created an Eclipse project.	3 years ago
LICENSE	ST6RI-259 Added content as o...	3 years ago
LICENSE-GPL	ST6RI-259 Added content as o...	3 years ago
README.md	Updated README.md	4 months ago
README.pdf	Updated README.pdf.	8 months ago

README.md

OMG Systems Modeling Language™ (SysML®) v2 Release

The SysML v2 Release repository contains the latest

About

The latest incremental release of SysML v2. Start here.

sysml · systems-engineering · mbse

Readme

LGPL-3.0, GPL-3.0 licenses found

Activity

297 stars

57 watching

41 forks

Report repository

Releases 33

2023-10 - SysML v2 Release Latest
2 days ago

+ 32 releases

Packages

No packages published

Contributors 2

seidewitz Ed Seidewitz

Feel free to ask your question

Thank you

cgrimm@rptu.de