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
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?

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

### Table: Project Success

<table>
<thead>
<tr>
<th>Size</th>
<th>Successful</th>
<th>Challenged</th>
<th>Failed</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>Grand</td>
<td>6%</td>
<td>51%</td>
<td>43%</td>
<td>100%</td>
</tr>
<tr>
<td>Large</td>
<td>11%</td>
<td>59%</td>
<td>30%</td>
<td>100%</td>
</tr>
<tr>
<td>Medium</td>
<td>12%</td>
<td>62%</td>
<td>26%</td>
<td>100%</td>
</tr>
<tr>
<td>Moderate</td>
<td>24%</td>
<td>64%</td>
<td>12%</td>
<td>100%</td>
</tr>
<tr>
<td>Small</td>
<td>61%</td>
<td>32%</td>
<td>7%</td>
<td>100%</td>
</tr>
</tbody>
</table>

Source: Chaos Report 2015
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?

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”? 

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

- Specify system functions
- Specify system structures
- Document requirements

**SysML v2 Language**

- **Requirements**
  - Documentation
  - Annotation

- **Structures**
  - Decomposition
  - Interconnection
  - Classification

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

- **Expressions**
  - Constraints
  - Assertions

- 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
     \(\rightarrow\) 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

SysML v2
Textual

SysML v2
Diagrams

REST API
KerML Entities

Various CAD Tools
From different vendors

Domain-Specific
Languages, ...
Tools, ...

KerML Entities

KerML Entities

KerML Entities

KerML Entities

REST API

REST API

REST API

REST API
The SysML v2 Eco-System (a vision)

- Persistence & version management (KerML instances)

Documentation
- Semantic web
- Data sheets

REST API
- KerML Entities

Bob creates a spec model & tests
- SysML v2 Textual
- SysML v2 Diagrams

Alice gets requirements & docs ...

Daniel designs housing
- Various CAD Tools from different vendors

Charles develops ASIC
- Domain-Specific Languages, ...
- Tools, ...

Slide 12
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)
  - 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 ...

Continuous Consistency Checking
SysMD Demonstration 1: SystemC Roundtrip

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

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

- 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

**Relationship** (for all relations)
- source Identifications
- target Identifications
- at least two related elements
- not necessarily directed; both related elements can be source or target

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

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

Read more in:
KerML Classes
Overview
(Elements)

Tutorial coverage

Figure: Kernel modeling language (KerML) v1.0 Beta 1
KerML Classes
Overview
(Relationships)

Figure: Kernel modeling language (KerML) v1.0 Beta 1
Ownership (general)

- 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**.

```java
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: “:>”

→ Read more on type-relationships!
  - *Conjugation, Disjoining, ...*

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

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

```cpp
// 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;
}
```
Associations classify relationships by giving source and/or target classes, by end features:

```plaintext
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:

```plaintext
feature sensor1: Device;
feature controller1: Device;
connector wire1: Wire
  from sensor1
to controller1;
```
**Function, Expression**

**Functions** model abstract dependencies between values

- Cannot be evaluated

```c
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

```c
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

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

- 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

```plaintext
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.

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

assert constraint maxMass {
    vehicle.mass < 200.0 [SI::t]
}
// TimeOf(...), Duration(...)
// 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

---

```plaintext
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

```systemc
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)

```plaintext
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
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)

Documentation
Semantic web
Data sheets

Persistance & version management
(KerML instances)

REST API
KerML Entities

SysML v2 Textual

REST API
KerML Entities

SysML v2 Diagrams

REST API
KerML Entities

Various CAD Tools
From different vendors

Domain-Specific Languages, ...
Tools, ...

Slide 41
**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*

```json
{
    "@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**

<table>
<thead>
<tr>
<th>Operations</th>
<th>Endpoint</th>
</tr>
</thead>
<tbody>
<tr>
<td>GET</td>
<td>/projects/&lt;projectId&gt;/commits/ &lt;commitId&gt;/elements</td>
</tr>
<tr>
<td>GET</td>
<td>/projects/&lt;projectId&gt;/commits/ &lt;commitId&gt;/elements/&lt;elementId&gt;</td>
</tr>
<tr>
<td>GET</td>
<td>/projects/&lt;projectId&gt;/commits/ &lt;commitId&gt;/elements/&lt;elementId&gt;/relationships</td>
</tr>
<tr>
<td>GET</td>
<td>/projects/&lt;projectId&gt;/commits/ &lt;commitId&gt;/roots</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Operations</th>
<th>Endpoint</th>
</tr>
</thead>
<tbody>
<tr>
<td>POST, GET</td>
<td>/projects</td>
</tr>
<tr>
<td>GET, PUT, DELETE</td>
<td>/projects/&lt;projectId&gt;</td>
</tr>
</tbody>
</table>

### Commit Endpoints

<table>
<thead>
<tr>
<th>Operations</th>
<th>Endpoint</th>
</tr>
</thead>
<tbody>
<tr>
<td>POST, GET</td>
<td>/projects/&lt;projectId&gt;/commits</td>
</tr>
<tr>
<td>GET</td>
<td>/projects/&lt;projectId&gt;/commits/&lt;commitId&gt;</td>
</tr>
<tr>
<td>GET</td>
<td>/projects/&lt;projectId&gt;/commits/&lt;commitId&gt;/changes</td>
</tr>
<tr>
<td>GET</td>
<td>/projects/&lt;projectId&gt;/commits/&lt;commitId&gt;/changes/&lt;changeId&gt;</td>
</tr>
</tbody>
</table>
SysML v2 Version Management API Services
(Platform Specific Model (PSM))

**Branch Endpoints**

<table>
<thead>
<tr>
<th>Operations</th>
<th>Endpoint</th>
</tr>
</thead>
<tbody>
<tr>
<td>POST, GET</td>
<td>/projects/&lt;projectId&gt;/branches</td>
</tr>
<tr>
<td>GET, DELETE</td>
<td>/projects/&lt;projectId&gt;/branches/&lt;branchId&gt;</td>
</tr>
</tbody>
</table>

**Tag Endpoints**

<table>
<thead>
<tr>
<th>Operations</th>
<th>Endpoint</th>
</tr>
</thead>
<tbody>
<tr>
<td>POST, GET</td>
<td>/projects/&lt;projectId&gt;/tags</td>
</tr>
<tr>
<td>GET, DELETE</td>
<td>/projects/&lt;projectId&gt;/tags/&lt;tagId&gt;</td>
</tr>
</tbody>
</table>
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
Feel free to ask your question

Thank you

cgrimm@rptu.de