SST

Moving Forward with SysML v2

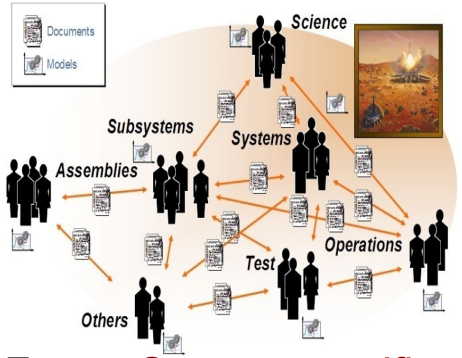
October 19, 2022

Sanford Friedenthal
SST Co-Lead
safriedenthal@gmail.com

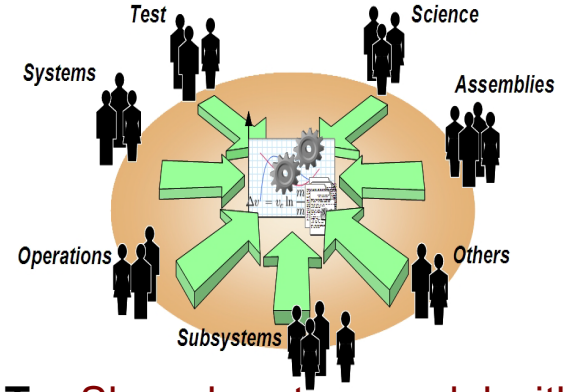
Model-Based Systems Engineering (MBSE)

SST

- A systems engineering approach where information about the system is captured in a system model
  - The model is the source of the data and managed throughout the lifecycle
- Contrasts with a document-based approach where the information is captured in a variety of documents, informal diagrams, and spreadsheets
- Provides a more complete, consistent, and traceable system design



From: System specification and design data related through documents



To: Shared system model with multiple views, and connected to discipline models

Source:
Jet Propulsion Laboratory

SysML v2 Objectives

SST

- **Increase adoption and effectiveness of MBSE by enhancing…**
  - Precision and expressiveness of the language
  - Consistency and integration among language concepts
  - Interoperability with other engineering models and tools
  - Usability by model developers and consumers
  - Extensibility to support domain specific applications
  - Migration path for SysML v1 users and implementors

Key Elements of SysML v2

SST

- New Metamodel that is not constrained by UML
  - Preserves most of UML modeling capabilities with a focus on systems modeling
  - Grounded in formal semantics
- Robust visualizations based on flexible view & viewpoint specification and execution
  - Graphical, Tabular, Textual
- Standardized API to access the model

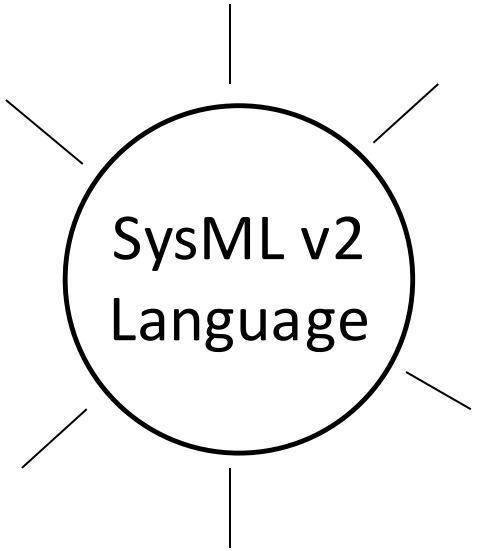
SysML v2 Language Capabilities

SST

Requirements

Behavior
- function-based
- state-based
- sequence-based
- use cases

Structure
- decomposition
- interconnection
- classification



Analysis
- analysis cases
- expression language

Verification
- verification cases

View & Viewpoint

Vehicle Part Definition
Replaces SysML v1 Block

SST

- The vehicle part definition is characterized by different kinds of features including
  - Attributes
  - Ports
  - Actions
  - States
  - …

«part def» Vehicle
attributes mass :> ISQ::mass = dryMass + cargoMass + fuelMass dryMass :> ISQ::mass cargoMass :> ISQ::mass fuelMass :> ISQ::mass position :> ISQ::length velocity :> ISQ::speed acceleration :> ISQ::acceleration avgFuelEconomy :> distancePerVolume electricalPower :> ISQ::power
ports fuelCmdPort : FuelCmdPort ignitionCmdPort : IgnitionCmdPort vehicleToRoadPort : VehicleToRoadPort
perform actions providePower
exhibit states vehicleStates

Vehicle Part Definition
Textual Syntax

SST

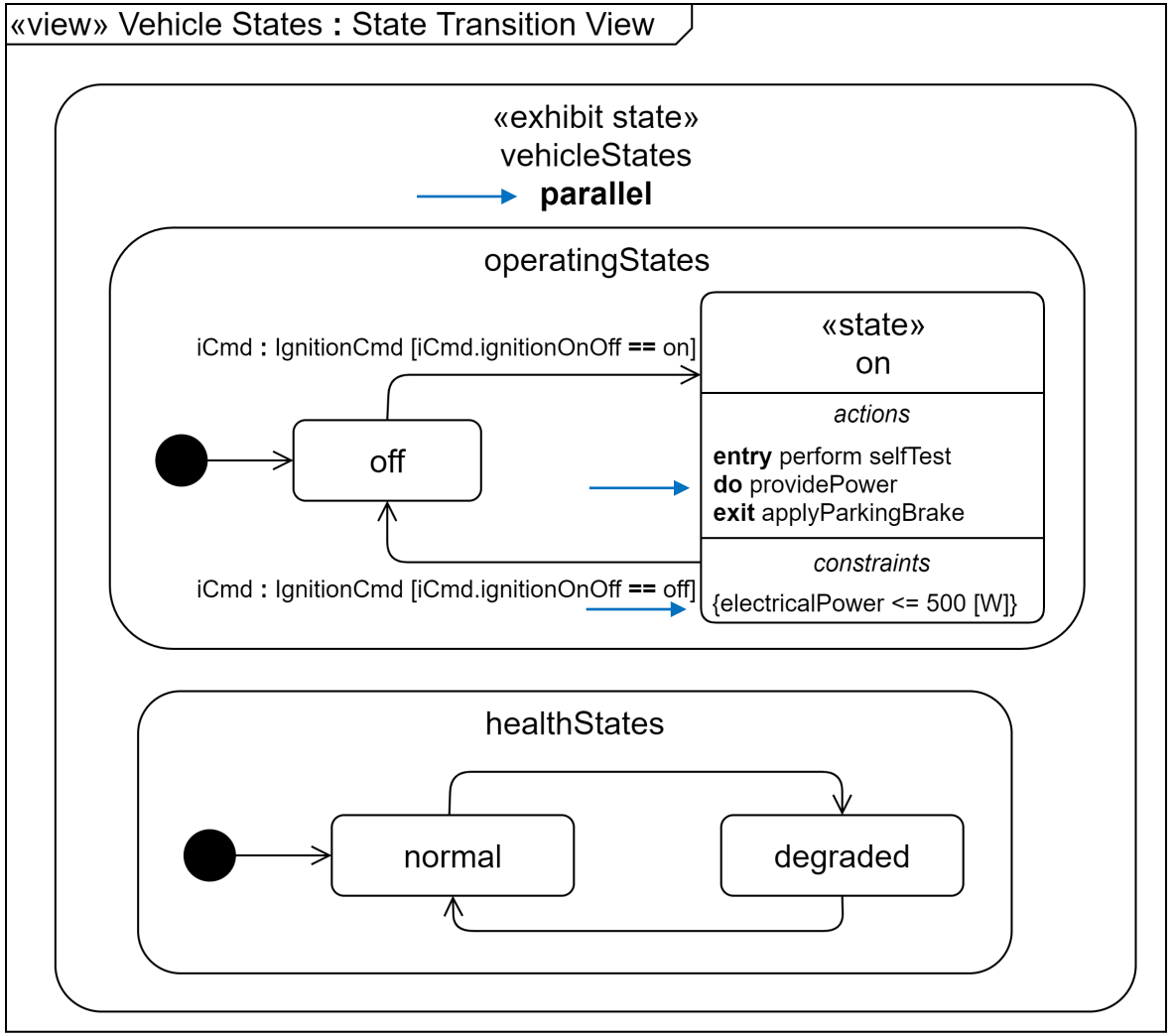
- The textual syntax reflects the same model as the graphical syntax

```
part def Vehicle{
    attribute mass :> ISQ::mass = dryMass + cargoMass + fuelMass;
    attribute dryMass:>ISQ::mass;
    attribute cargoMass:>ISQ::mass;
    attribute fuelMass:>ISQ::mass;
    attribute position:>ISQ::length;
    attribute velocity:>ISQ::speed;
    attribute acceleration:>ISQ::acceleration;
    attribute avgFuelEconomy:>distancePerVolume;
    attribute electricalPower:> ISQ::power;
    port fuelCmdPort:FuelCmdPort;
    port ignitionCmdPort:IgnitionCmdPort;
    port vehicleToRoadPort:VehicleToRoadPort;
    perform action providePower;
    exhibit state vehicleStates parallel {↔}
}
```

Vehicle States

SST

- States are hierarchical and can include:
  - parallel states (e.g., concurrent states) and mutually exclusive states
  - entry, exit, and do actions
  - constraints



Vehicle States
Textual Syntax

SST

```
exhibit state vehicleStates parallel {
    state operatingStates {
        entry action initial;
        state off;
        state on {
            entry action performSelfTest;
            do providePower;
            exit action applyParkingBrake;
            constraint {electricalPower<=500[W]}
        }
        transition initial then off;
        transition off_To_on
            first off
            accept ignitionCmd:IgnitionCmd via ignitionCmdPort
                if ignitionCmd.ignitionOnOff==IgnitionOnOff::on
            then on;
        transition on_To_off
            first on
            accept ignitionCmd:IgnitionCmd via ignitionCmdPort
                if ignitionCmd.ignitionOnOff==IgnitionOnOff::off
            then off;
    }
    state healthStates {
        entry action initial;
        state normal;
        state degraded;
```

SysML v2 Reuse Patterns

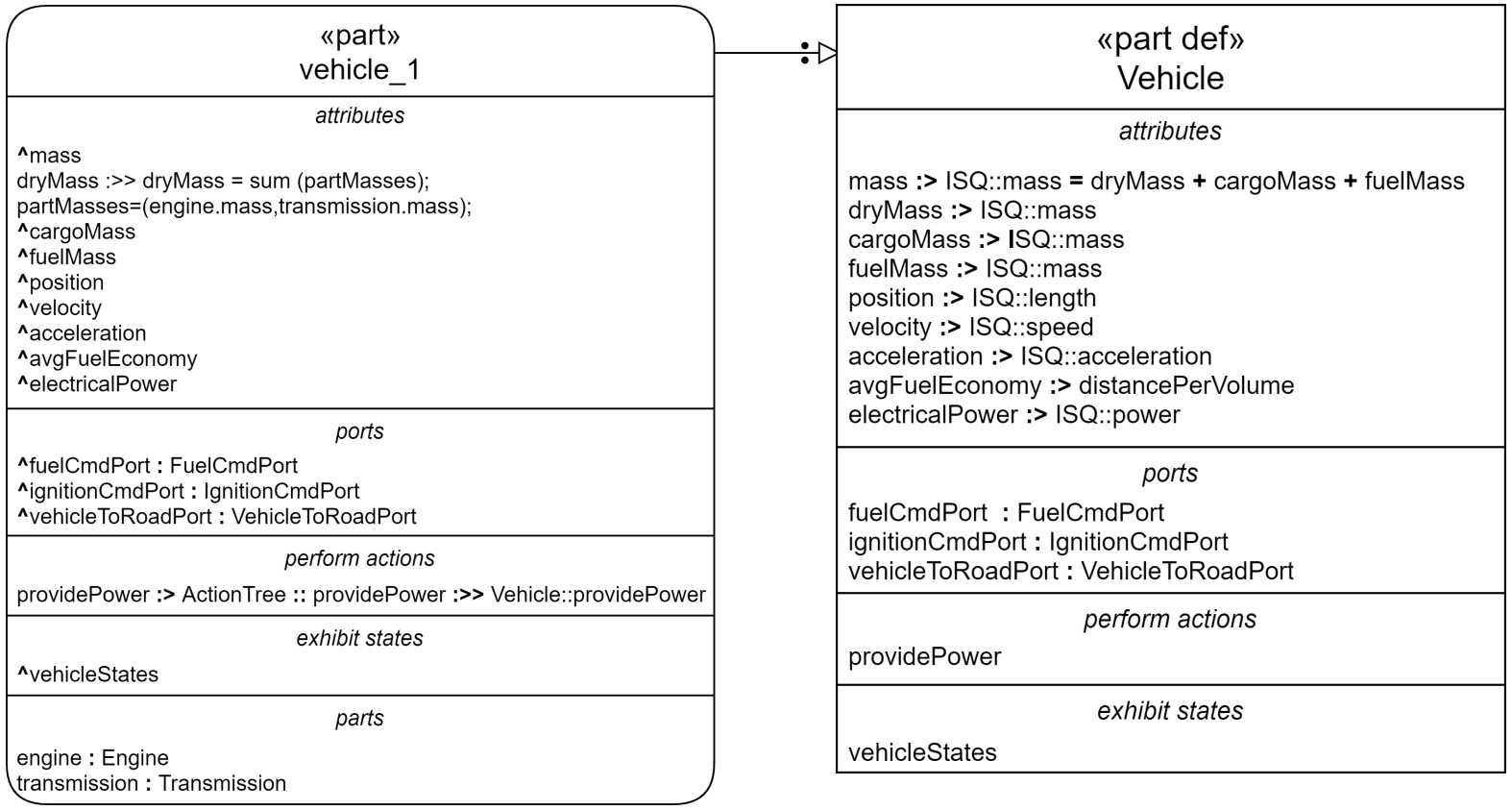
SST

- Definition and usage
  - A definition element defines an element such as a part, action, or requirement
  - A usage element is a usage of a definition element in a particular context
  - Pattern is applied consistently throughout the language
- Variability
  - Variation points represent elements that can vary
    - Variation applies to all definition and usage elements
  - A variant represents a particular choice at a variation point
  - A choice at one variation point can constrain choices at other variation points
  - A system can be configured by making choices at each variation point consistent with the specified constraints

Vehicle Part Defined by Vehicle Part Definition

SST

- Parts are specializations of their definitions (defined by)
  - Enables adaptation of each usage to its context by inheriting and redefining its features



Vehicle Part
Textual Syntax

SST

nested part →

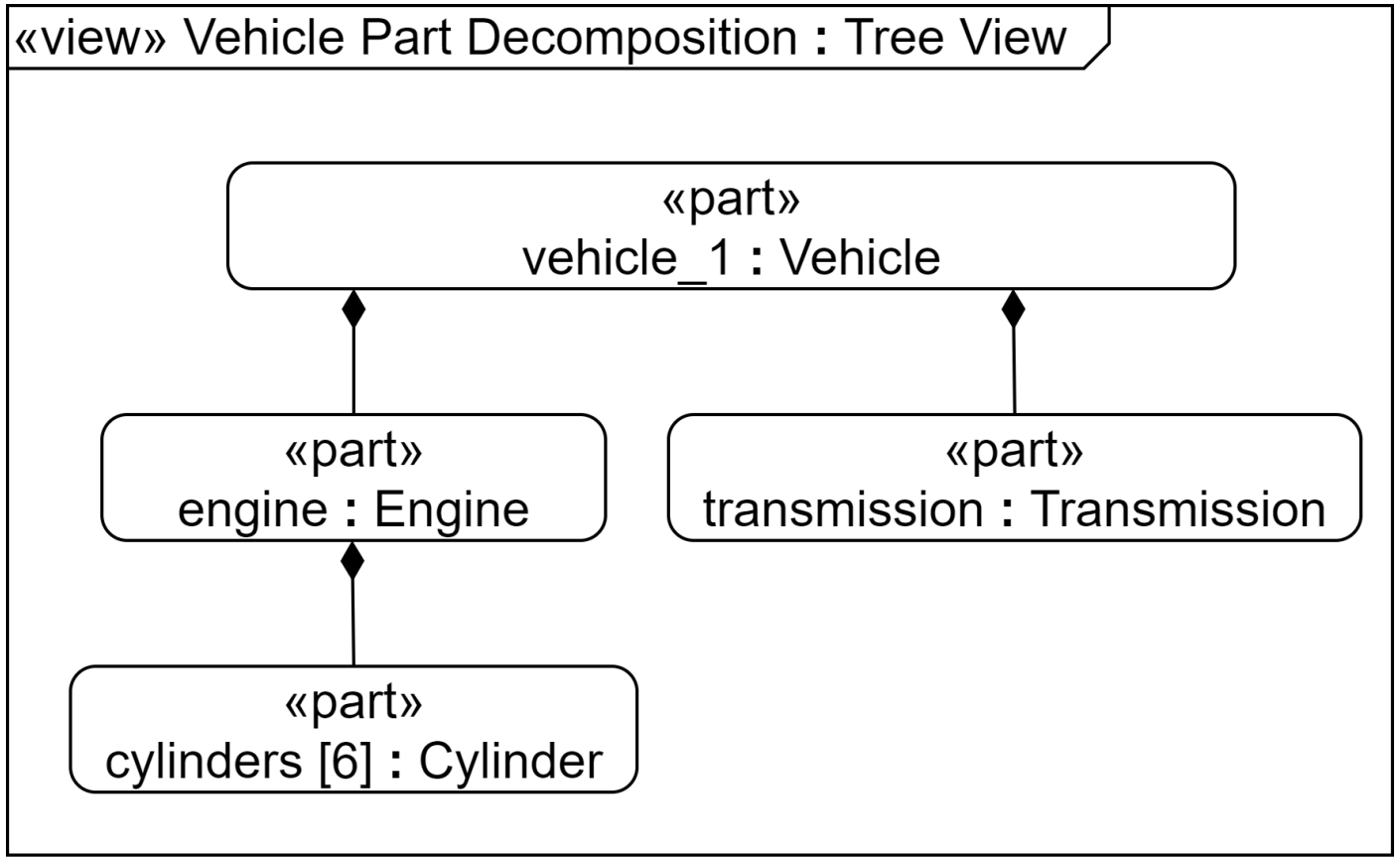
connection →

```
part vehicle_1:Vehicle{
    attribute mass redefines mass;
    attribute dryMass redefines dryMass = sum (partMasses);
    attribute partMasses=(engine.mass,transmission.mass);
    perform ActionTree::providePower redefines providePower;
    part engine:Engine{
        attribute mass redefines mass default 200 [kg];
        port fuelCmdPort:>>fuelCmdPort=vehicle_1.fuelCmdPort;
        port ignitionCmdPort:>>ignitionCmdPort=vehicle_1.ignitionCmdPort;
        perform ActionTree::providePower.generateTorque;
        part cylinders[6]:Cylinder;
    }
    part transmission:Transmission{
        attribute mass redefines mass default 60 [kg];
        perform action amplifyTorque:>> amplifyTorque = ActionTree::providePower.amplifyTorque;

    }
    connect engine.drivePwrPort to transmission.drivePwrPort;
}
```

Vehicle Part Decomposition

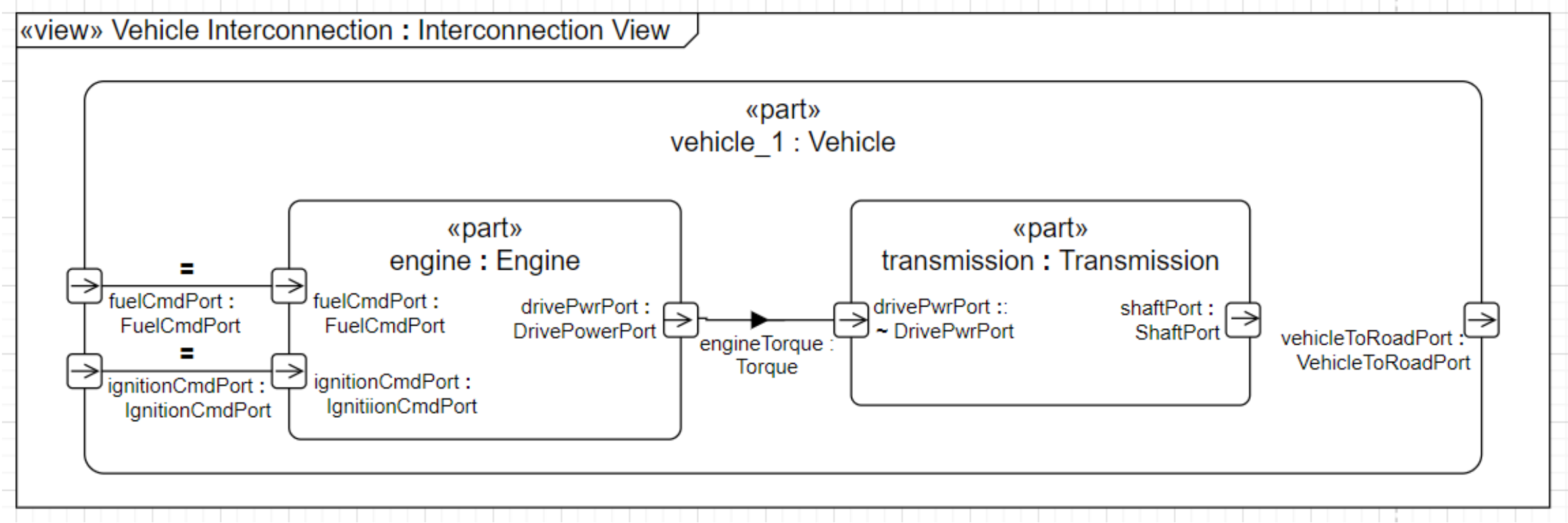
SST



Vehicle Part Interconnection

SST

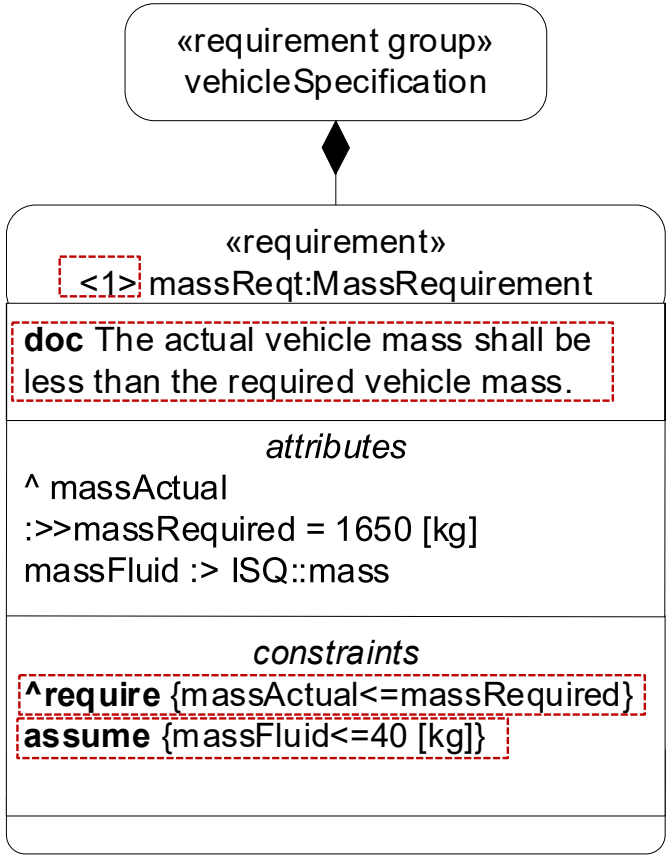
- Part interconnection contains parts, ports, connections, and flows



Requirements

SST

- A constraint definition that a valid design solution must satisfy that can include:
  - Identifier
  - Shall statement
  - Constraint expression that can be evaluated to true or false
    - can apply to performance, functional, interface and other kinds of requirements if desired
  - Assumed constraint expression that is asserted to be true for the requirement to be valid



4D Model

SST

- Each entity has a lifetime
  - Reference clock
  - Can specify timeslices and snapshots
- Spatial entities have spatial extent over different parts of their lifetime
  - Specified by shapes with position and orientation within coordinate frames
- Individuals
  - Unique identify with a lifetime

SysML v2 to v1 Terminology Mapping (partial)

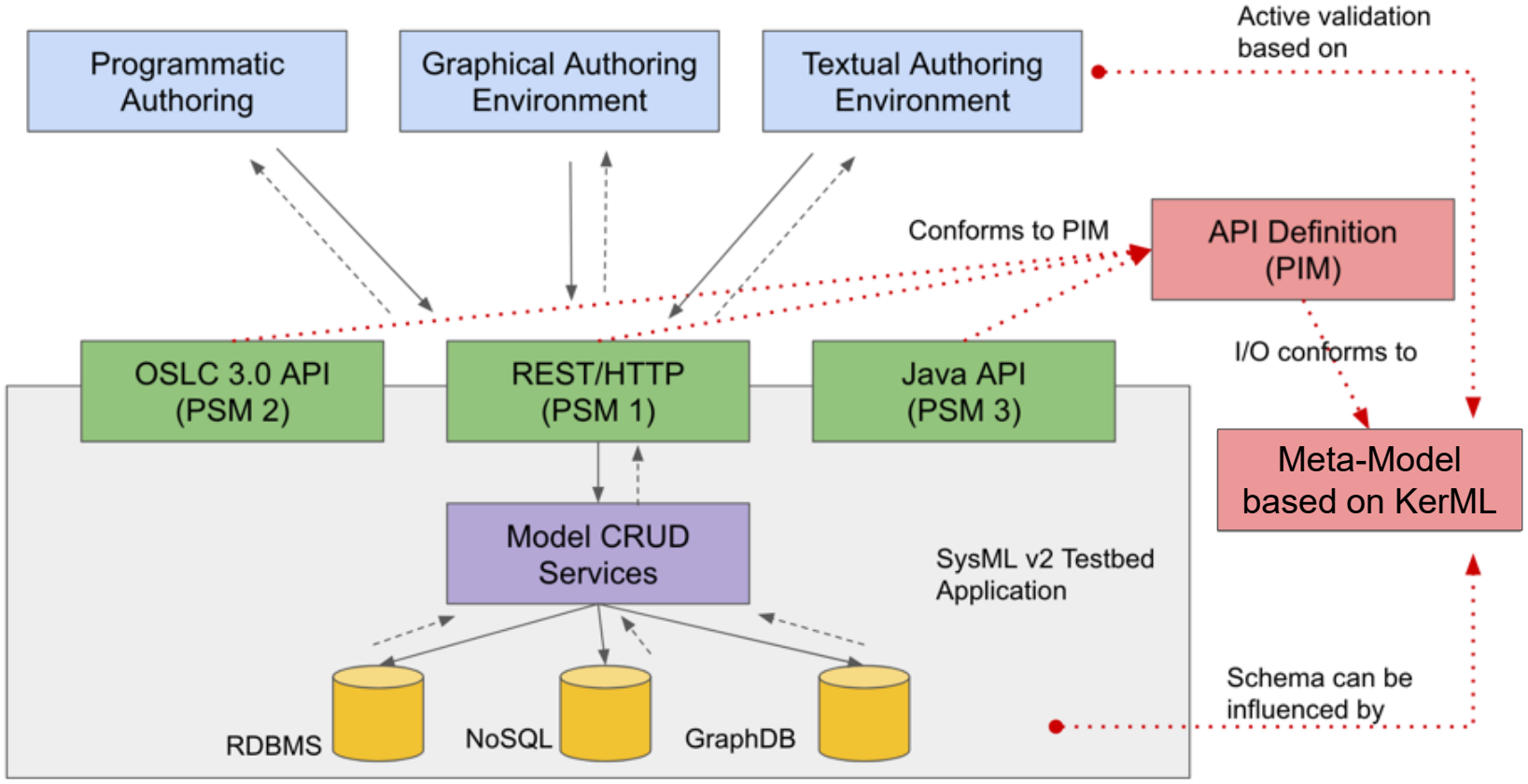
SST

SysML v2	SysML v1
part / part def	part property / block
attribute / attribute def	value property / value type
port / port def	proxy port / interface block
action / action def	action / activity
state / state def	state / state machine
constraint / constraint def	constraint property / constraint block
requirement / requirement def	requirement
connection / connection def	connector / association block
view / view def	view

SysML v2 Pilot Implementation Using Standard API

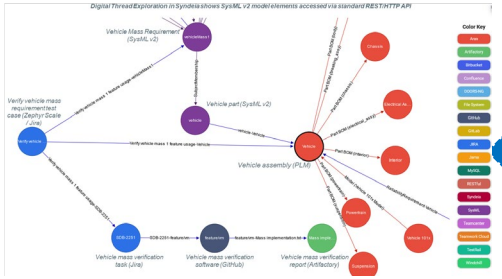
SST

High-Level Architecture of SysML v2 Testbed



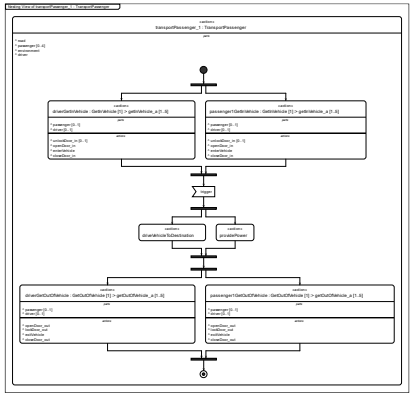
Connecting SysML v2 through the API

SST



PLM/Version Mgmt

Source: Syndeia with SysML v2



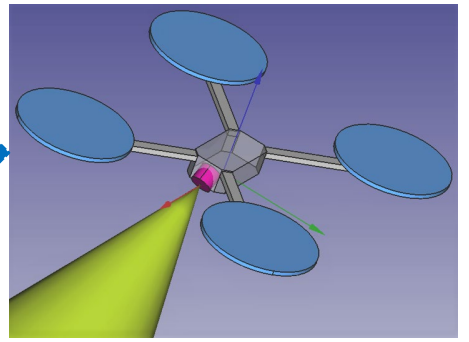
Graph Visualization

Source: Tom Sawyer with SysML v2

Systems Modeling API

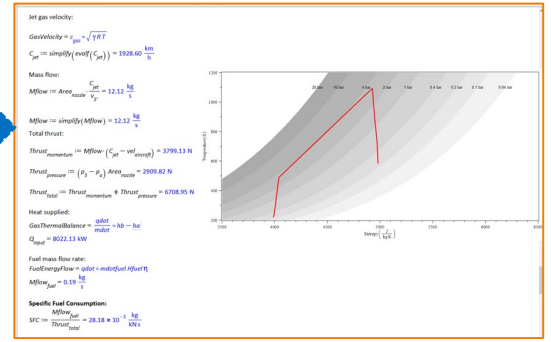
SysML v2

- Structure
- Behavior
- Requirements
- Analysis
- Verification
- View & Viewpoint



CAD/CAD Viewer

Source: FreeCAD with SysML v2



Analysis Solver

Source: Maple with SysML v2

Contrasting SysML v2 with SysML v1

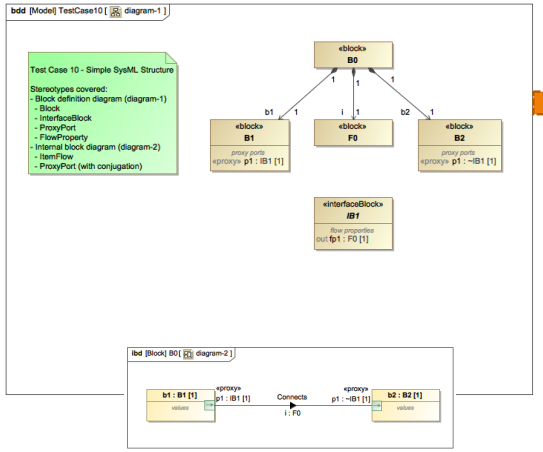
SST

- Simpler to learn and use
  - Systems engineering concepts designed into metamodel versus added-on
  - Consistent application of definition and usage pattern
  - More consistent terminology
  - Ability to decompose parts, actions,
  - More flexible model organization (unowned members, package filters)…
- More precise
  - Textual syntax and expression language
  - Formal semantic grounding
  - Requirements as constraints
- **More expressive**
  - Variant modeling
  - Analysis case
  - Trade-off analysis
  - Individuals, snapshots, time slices
  - More robust quantitative properties (e.g., vectors, ..)
  - Simple geometry
  - Query/filter expressions
  - Metadata
- **More extensible**
  - Simpler language extension capability
    - Based on model libraries
- **More interoperable**
  - Standardized API

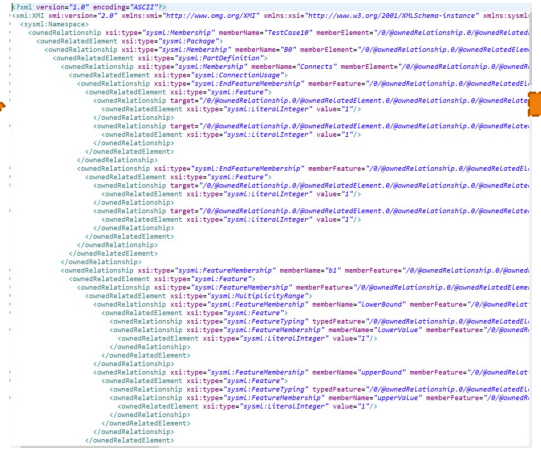
SysML v1 to SysML v2 Transformation

SST

SysML v1 Model



SysML v2 XMI



SysML v2 Textual Notation

```
package eVehicleLibrary {
 attribute def ElectricEnergy;
 attribute def BatteryCapacity :> ScalarValues::Integer;
 attribute def Speed :> ScalarValues::Integer;
 port def PowerOutPort { out energy : ElectricEnergy;
 }
 interface def PowerInterface {
  end supplierPort : PowerOutPort;
  end consumerPort : ~PowerOutPort;
 }
}
package eVehicleDefinitions {
    import eVehicleLibrary::*;
 part def Wheel {
  value size : ScalarValues::Integer;
 }
 part def Battery {
  value capacity : BatteryCapacity;
 }
 part def Engine;
}
....
```

Source: SST Track 3 Presentation
Yves Bernard, Tim Weilkiens
08 February 2022

SysML v2 Milestones

SST

December, 2017	SysML v2 RFP issued
June, 2018	SysML v2 API & Services RFP issued
August, 2020	Initial Submission
February, 2021	Stakeholder Review
August, 2021	Revised Submission
November, 2021	2nd Revised Submission (OMG evaluation initiated)
November, 2022	3rd Revised Submission
1st Qtr 2023	Final Submission (beta specification)
2024	Adopted Specification (pending OMG approvals)

Summary

Summary

SST

- SysML v2 is addressing SysML v1limitations to improve MBSE adoption and effectiveness
  - Precision, expressiveness
  - Regularity, usability
  - Interoperability with other engineering models and tools
- Approach
  - SysML v2 metamodel with formal semantics architected to overcome fundamental UML limitations
  - Flexible graphical notations and textual notation
  - Standardized API for interoperability
  - Transformation specification from SysML v1 to SysML v2
- Plan
  - Final submission (beta specification) - Q1 2023
  - Final adopted specification - 2024

SST Public Repositories
Current Release: 2022-08

SST

- Monthly release repository
  - https://github.com/Systems-Modeling/SysML-v2-Release
- Release content
  - Specification documents (for KerML, SysML and API)
  - Training material for SysML textual notation
  - Training material for SysML graphical notation
  - Example models (in textual notation)
  - Pilot implementation
    - Installer for Jupyter tooling
    - Installation site for Eclipse plug-in
  - Web access to prototype repository via SysML v2 API
  - Web access to Tom Sawyer visualization tooling
- Open-source repositories
  - https://github.com/Systems-Modeling
- Google group for comments and questions
  - https://groups.google.com/g/SysML-v2-Release
    (to request membership, provide name, affiliation and interest)

Questions