MAP “System Structure and Parameterization” – Current Status and Plans

presented by Pierre R. Mai (PMSF)

MA User Meeting 2018-10-11
Cambridge, MA

T. Sommer

M. Najafi

M. Deppe

J. Köhler

J. Krasser

P. R. Mai

M. Nagasawa

M. Henningsson

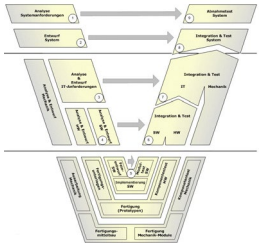
J.N. Jäschke

MA Project “System Structure and Parameterization”

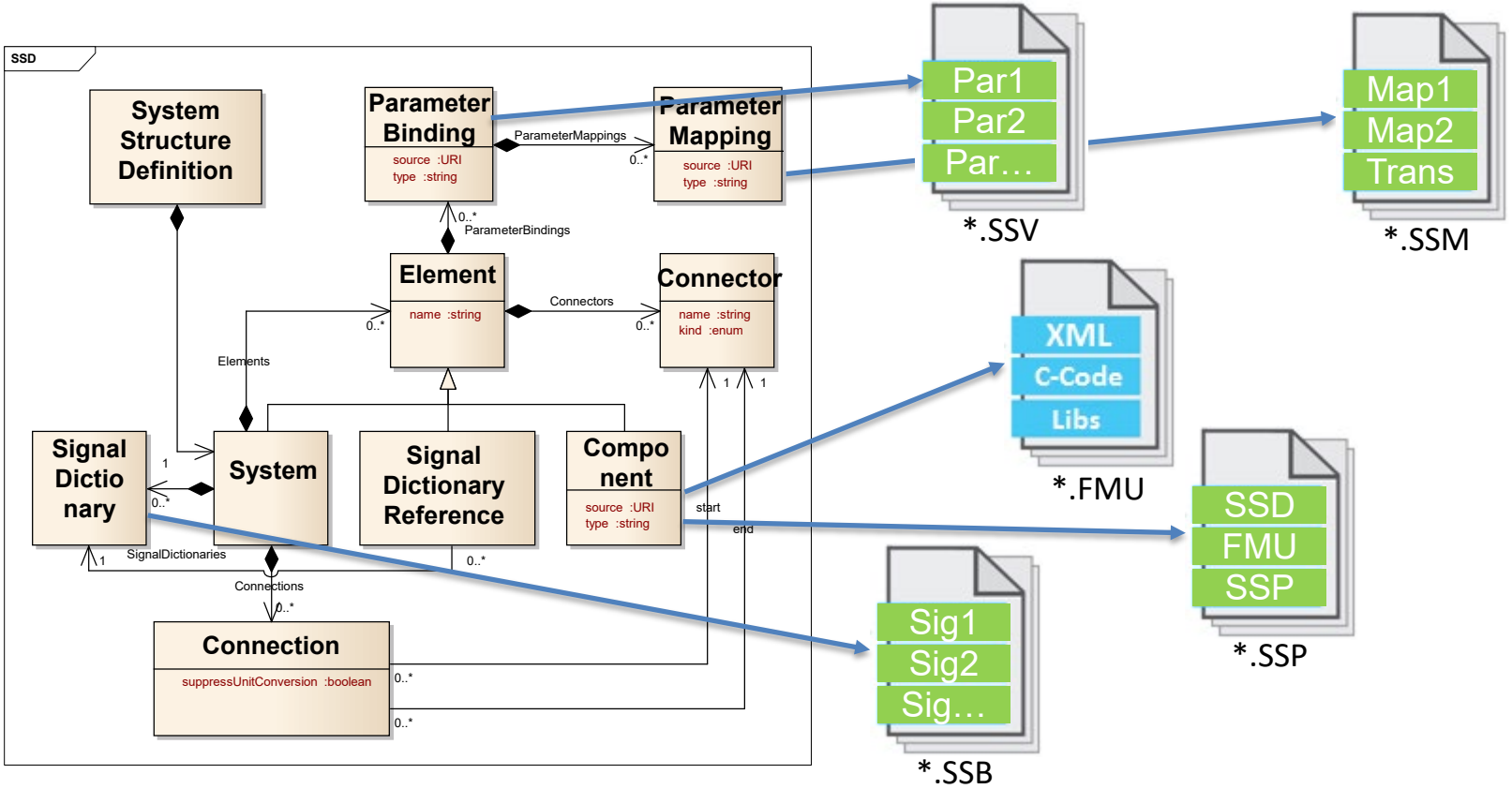
- Initial kick-off of discussions in 2014, start of MA project
- Use case definitions in 2015
- Major technical design work during 2016-2017
- Refining, prototypes, documentation, review during 2017-2018
- Currently ~2-3 face-to-face meetings per year
- Major work now being done via git, web-meetings
- Mostly European participants, but some Japanese, US members

Main Purposes of SSP – Based on FMI standard

- Define a standardized format for the connection structure of a network of components (FMUs in particular).
- Define a standardized way to store and apply parameters to these components.
- The developed standard / APIs should be usable in all stages of development process (architecture definition, integration, simulation, test in MiL, SiL, HiL).
- The work in this project shall be coordinated with other standards and organizations (FMI, ASAM, OMG).

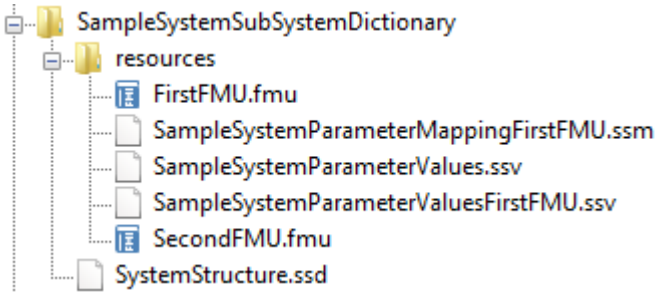


Overview of File Definitions



File Definitions – System Structure Package (*.SSP)

*.SSP



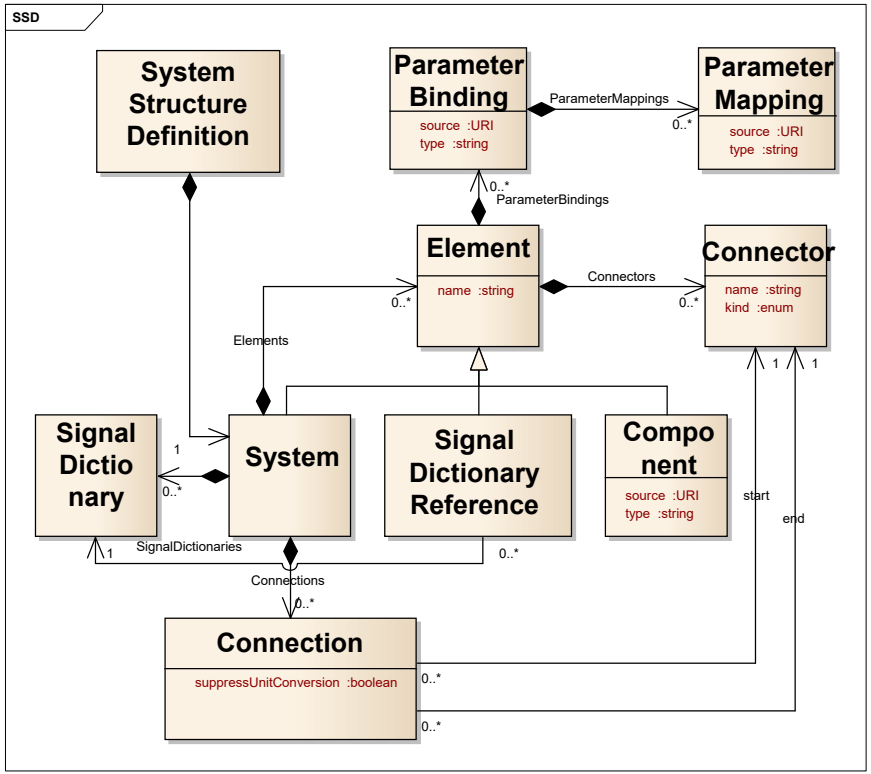
Use case

- Exchange of Complete Systems with Variants and all Related Resources

Features

- All information (FMUs, system structure definition, parameters) can be stored in one archive (zip-file)
- Multiple SSDs in one SSP allows for variant modeling
- Common relative URI addressing of content for unified access, integration into PLM/PDM

File Definitions – System Structure Definition (*.SSD)



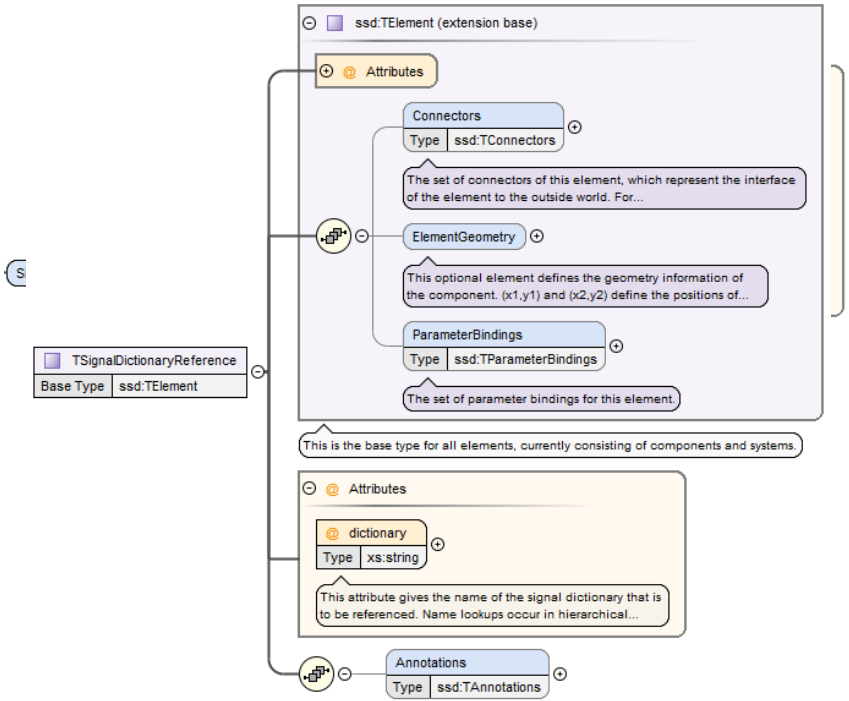
Use case

- Defining a Network of FMUs/Models

Features

- Hierarchical sub-systems
- Empty components as interface templates
- SignalDictionaries for flexible connection definitions (e.g. bus databases, non-hierarchical connection setup)
- External resources via URIs
- Connections with unit conversions and optional linear/map transformations
- Optional: Diagram geometry
- Optional: Basic simulation settings
- All Formats: Extension via Domain-Specific Annotations

File Definitions – Signal Dictionaries (*.SSB)



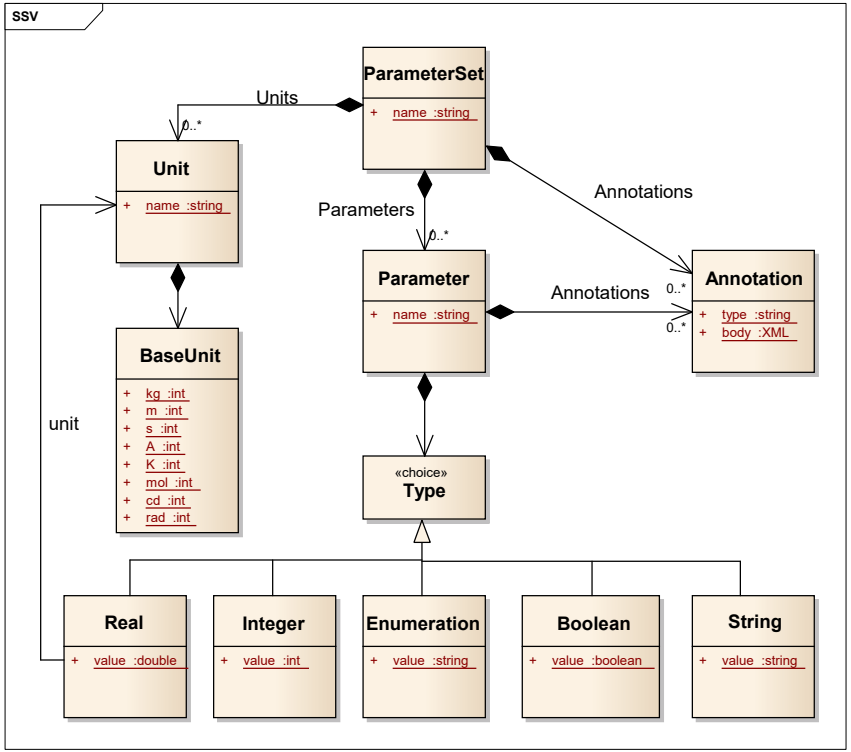
Use case

- Collecting Control Signals in a Central Location with Reuse across Projects

Features

- Causality is checked by tool automatically
- Connections expandable to actual point-to-point connections prior to runtime
- Crosses hierarchies without need for downward passing of signals
- Well-suited for e.g. ECU control busses
- Can be inline or in external file

File Definitions – Parameter Values (*.SSV)



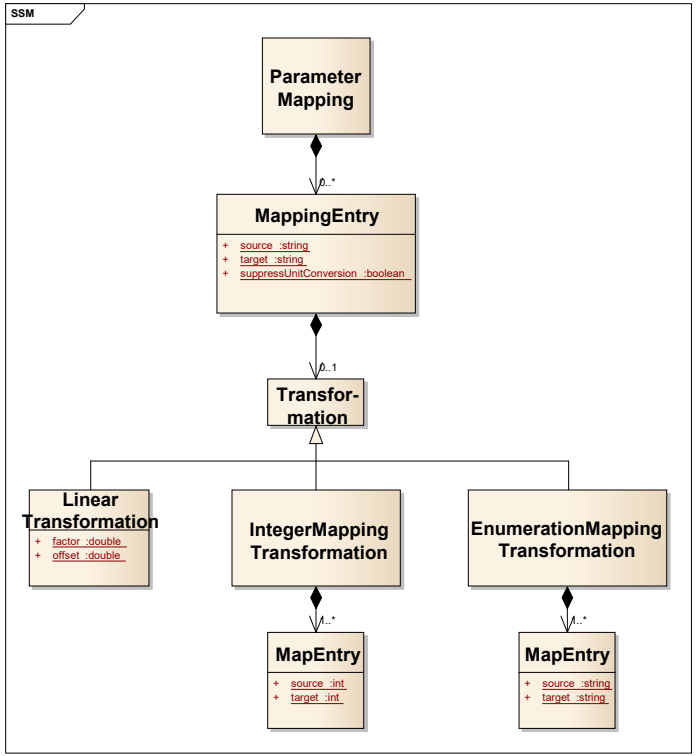
Use case

- Tool-independent Exchange of Parameter Data with Reuse across Projects

Features

- Neutral exchange format between parameter sources
- Compatible to FMI standard data types
- Provides some meta data
- Access to Parameter DBs via HTTP/S (-> Parameter REST API)
- Can be inline or in external file

File Definitions – Parameter Mapping (*.SSM)



Use case

- Mapping Parameters to FMUs / Components when Parameter Names differ or Parameter Values require Transformations (e.g. due to Reuse)

Features

- Neutral exchange format between tools
- Optional manual linear and mapping transformations
- Can be inline or in external file

List of features

- Hierarchical description of systems of connected components
- Components: FMUs and external SSPs/SSDs, extensible to models, …
- Parameter bindings both at component and system-level, including transformations and name/unit-mapping
- Signal dictionaries support cross-hierarchical data pools (e.g for busses)
- Packaging of SSDs, FMUs, Parameters, … into one bundle (SSP)
- Light-weight support for variant handling at SSP level (multiple SSDs sharing components, parameters, resources)
- Optional exchange of graphical information (similar display across tools)
- URI references to all resources: Integration with other systems via URIs

Existing prototypes with support of SSP

- Model.CONNECT™ by AVL – Scope
- Integration Tool FMI Bench by PMSF
- Co-Simulation Player (Cybernet)
- TLK MoBA Lab by TLK
- FMI Composer by Modelon
- FMI Kit – FMI Library for Simulink by Dassault
- SSP support by solidThinking Activate® by Altair
- FMPy – Open Source Python FMI/SSP Library/Tool by Dassault

Current Roadmap for First Release

Topic	Due Date
Limited Distribution “Final Committee Draft” w.r.t. technical contents	Q1/2018
Publicly available “Beta Version”, Website	Q4/2018
End of Beta Phase, Release Candidate	Q1/2019
Release version 1.0 (aligned with FMI 1.0/2.0, partial support for FMI 3.0)	Q2/2019
Tentative 1.1 follow up release, full alignment with all of FMI 3.0	Q4/2019 – Q1/2020

Q&A