Variability Modeling and Automated Simulation Experiments

Faculty of Engineering / Research Group CEA

Thorsten Pawletta
Hendrik Folkerts
Christina Deatcu

E-Mail:
{thorsten.pawletta, christina.deatcu}@hs-wismar.de
hendrik.folkerts@cea-wismar.de

Web:
www.hs-wismar.de / www.cea-wismar.de

Outline

1. Introduction
2. The case study
3. Basics of SES/MB based modeling
4. Practical modeling: implementation of an SES
5. Model selection and model generation
6. The free Matlab SES toolbox
7. Organization of a simulator-independent MB
8. Full automation of simulation experiments
9. Conclusion

Outline

1. **Introduction** (T. Pawletta)
2. The case study
3. Basics of SES/MB based modeling
4. Practical modeling: implementation of an SES
5. Model selection and model generation
6. The free Matlab SES toolbox
7. Organization of a simulator-independent MB
8. Full automation of simulation experiments
9. Conclusion

Introduction

Introduction

- Systems, such as IOT, CPS or Industry 4.0, have a **high degree of variability**
  - Modern cars have more than 50 electronic control units (ECU), each may be instantiated in at least 10.000 different ways.[1]
    → **Millions of configurations (variants)**

[1] Oster S. (2011) „Feature Model-based Software Product Line Testing“. PhD thesis, TU Darmstadt.

Introduction

- Systems, such as IOT, CPS or Industry 4.0, have a **high degree of variability**
  - Modern cars have more than 50 electronic control units (ECU), each may be instantiated in at least 10.000 different ways.[1]
    → **Millions of configurations (variants)**
- **Variability** is defined in Software Engineering as:

  the ability of a system to be configurable, extendable or adaptable depending on its purpose and objective. → **SPL engineering** (SPLE)

[1] Oster S. (2011) „Feature Model-based Software Product Line Testing“. PhD thesis, TU Darmstadt.

Introduction

- Systems, such as IOT, CPS or Industry 4.0, have a **high degree of variability**
  - Modern cars have more than 50 electronic control units (ECU), each may be instantiated in at least 10.000 different ways.[1]
    → **Millions of configurations (variants)**
- **Variability** is defined in Software Engineering as:

  the ability of a system to be configurable, extendable or adaptable depending on its purpose and objective. → **SPL engineering** (SPLE)
- **M&S:** Management of model families (set of models with common features)

[1] Oster S. (2011) „Feature Model-based Software Product Line Testing“. PhD thesis, TU Darmstadt.

Introduction

- Systems, such as IOT, CPS or Industry 4.0, have a **high degree of variability**
  - Modern cars have more than 50 electronic control units (ECU), each may be instantiated in at least 10.000 different ways.[1]
    → **Millions of configurations (variants)**
- **Variability** is defined in Software Engineering as:

  the ability of a system to be configurable, extendable or adaptable depending on its purpose and objective. → **SPL engineering** (SPLE)
- **M&S:** Management of model families (set of models with common features)
  - Today methods often transferred 1:1 from SPLE to M&S domain 

[1] Oster S. (2011) „Feature Model-based Software Product Line Testing“. PhD thesis, TU Darmstadt.

Introduction

- Systems, such as IOT, CPS or Industry 4.0, have a **high degree of variability**
  - Modern cars have more than 50 electronic control units (ECU), each may be instantiated in at least 10.000 different ways.[1]
    → **Millions of configurations (variants)**
- **Variability** is defined in Software Engineering as:

  the ability of a system to be configurable, extendable or adaptable depending on its purpose and objective. → **SPL engineering** (SPLE)
- **M&S:** Management of model families (set of models with common features)
  - Today methods often transferred 1:1 from SPLE to M&S domain 
  - More suitable are methods from **model based systems engineering** (→ **SES/MB approach**)

[1] Oster S. (2011) „Feature Model-based Software Product Line Testing“. PhD thesis, TU Darmstadt.

Introduction (2)

Introduction (2)

- **Why shouldn‘t SPLE be transferred 1:1 to the M&S domain?**

Introduction (2)

- **Why shouldn‘t SPLE be transferred 1:1 to the M&S domain?**
  - **Specific paradigms apply to M&S:**

Introduction (2)

- **Why shouldn‘t SPLE be transferred 1:1 to the M&S domain?**
  - **Specific paradigms apply to M&S:**
    - Separation of model structures & model dynamics
      (violation in 150% modeling approach)

Introduction (2)

- **Why shouldn‘t SPLE be transferred 1:1 to the M&S domain?**
  - **Specific paradigms apply to M&S:**
    - Separation of model structures & model dynamics
      (violation in 150% modeling approach)
    - Separation of model and experiment (use of a model in different context)

Introduction (2)

- **Why shouldn‘t SPLE be transferred 1:1 to the M&S domain?**
  - **Specific paradigms apply to M&S:**
    - Separation of model structures & model dynamics
      (violation in 150% modeling approach)
    - Separation of model and experiment (use of a model in different context)
    - Separation of model and simulator

Introduction (2)

- **Why shouldn‘t SPLE be transferred 1:1 to the M&S domain?**
  - **Specific paradigms apply to M&S:**
    - Separation of model structures & model dynamics
      (violation in 150% modeling approach)
    - Separation of model and experiment (use of a model in different context)
    - Separation of model and simulator
    - Modular-hierarchical modeling of configurations
      (variants of structure & parameter settings)

Introduction (2)

- **Why shouldn‘t SPLE be transferred 1:1 to the M&S domain?**
  - **Specific paradigms apply to M&S:**
    - Separation of model structures & model dynamics
      (violation in 150% modeling approach)
    - Separation of model and experiment (use of a model in different context)
    - Separation of model and simulator
    - Modular-hierarchical modeling of configurations
      (variants of structure & parameter settings)
- **Why should a model be used with different simulators?**

Introduction (2)

- **Why shouldn‘t SPLE be transferred 1:1 to the M&S domain?**
  - **Specific paradigms apply to M&S:**
    - Separation of model structures & model dynamics
      (violation in 150% modeling approach)
    - Separation of model and experiment (use of a model in different context)
    - Separation of model and simulator
    - Modular-hierarchical modeling of configurations
      (variants of structure & parameter settings)
- **Why should a model be used with different simulators?**
  - domain dependency, (?) simulator correctness

Introduction (2)

- **Why shouldn‘t SPLE be transferred 1:1 to the M&S domain?**
  - **Specific paradigms apply to M&S:**
    - Separation of model structures & model dynamics
      (violation in 150% modeling approach)
    - Separation of model and experiment (use of a model in different context)
    - Separation of model and simulator
    - Modular-hierarchical modeling of configurations
      (variants of structure & parameter settings)
- **Why should a model be used with different simulators?**
  - domain dependency, (?) simulator correctness
- **Study of variants is time-consuming**

Introduction (2)

- **Why shouldn‘t SPLE be transferred 1:1 to the M&S domain?**
  - **Specific paradigms apply to M&S:**
    - Separation of model structures & model dynamics (violation in 150% modeling approach)
    - Separation of model and experiment (use of a model in different context)
    - Separation of model and simulator
    - Modular-hierarchical modeling of configurations (variants of structure & parameter settings)
- **Why should a model be used with different simulators?**
  - domain dependency, (?) simulator correctness
- **Study of variants is time-consuming**

  - **Automation** of simulation experiments (selection of configurations, model generation & simulation, analysis of simulation results)

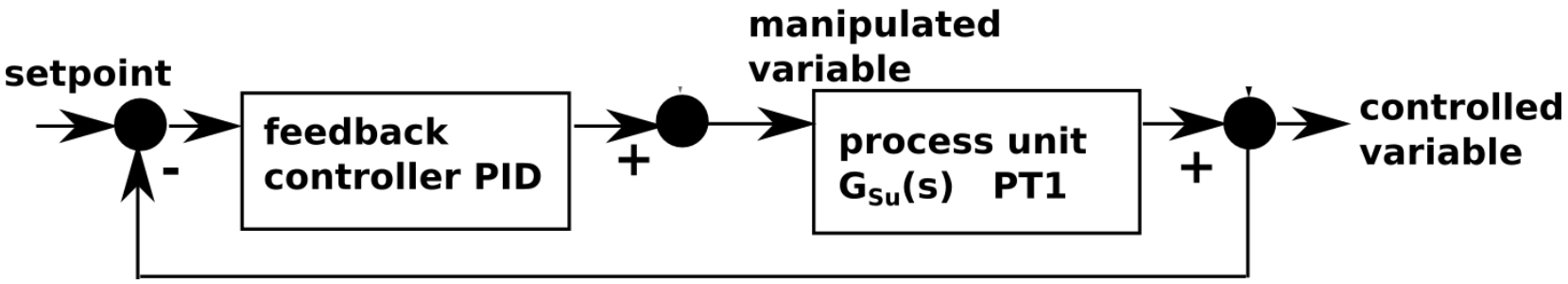
Outline

1. Introduction
2. **The case study** (T. Pawletta)
3. Basics of SES/MB based modeling
4. Practical modeling: implementation of an SES
5. Model selection and model generation
6. The free Matlab SES toolbox
7. Organization of a simulator-independent MB
8. Full automation of simulation experiments
9. Conclusion

Case Study

Case Study

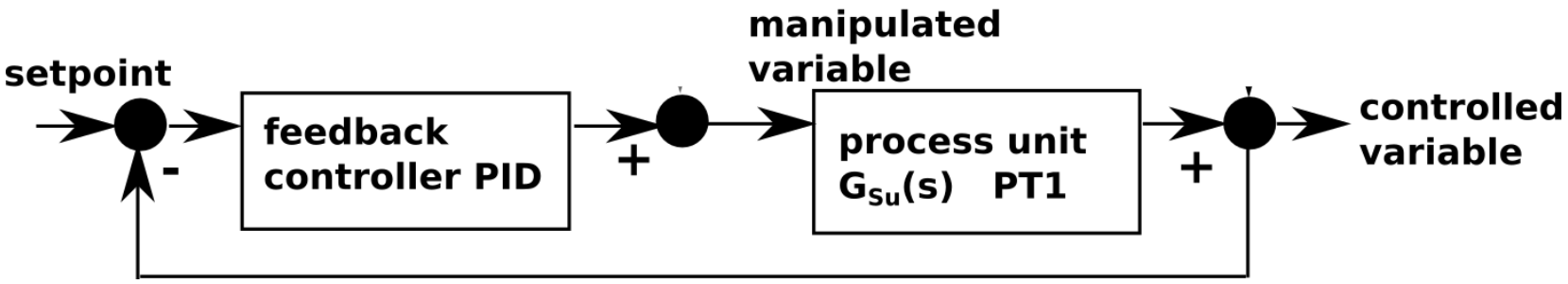
- Feedback control system



Case Study

- Feedback control system
- Described by transfer functions

$$G_{Su}(s) = \frac{1}{20 \cdot s + 1}$$

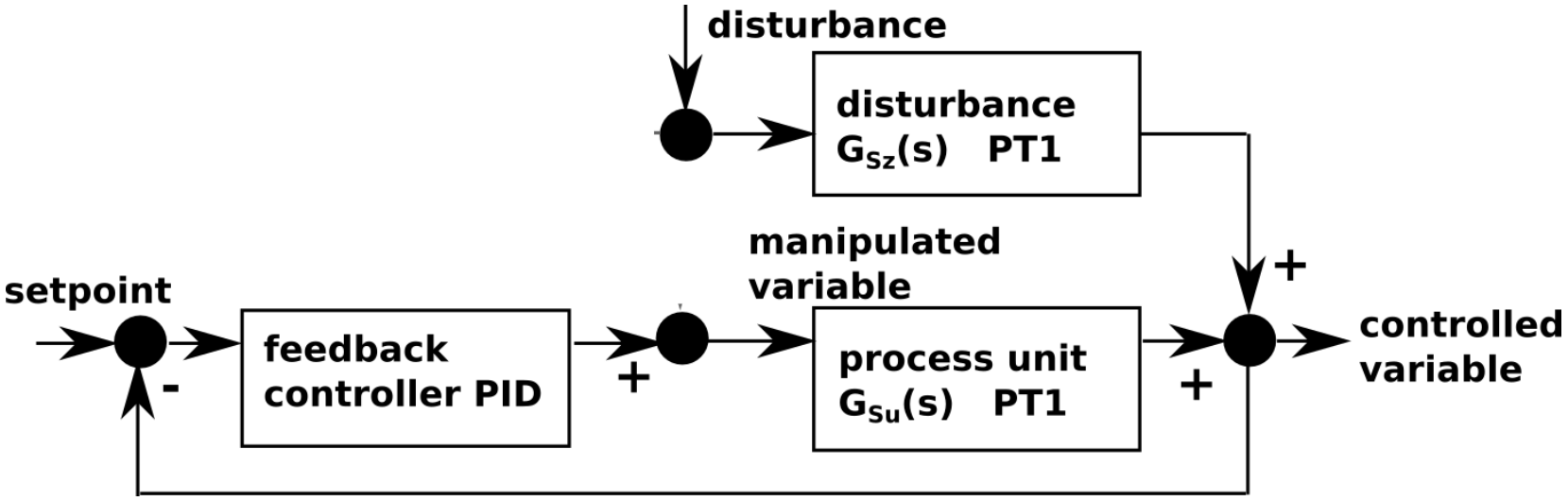


Case Study

- Feedback control system
- Described by transfer functions
- Influenced by disturbances

$$G_{Su}(s) = \frac{1}{20 \cdot s + 1}$$

$$G_{Sz}(s) = \frac{1}{10 \cdot s + 1}$$



Case Study

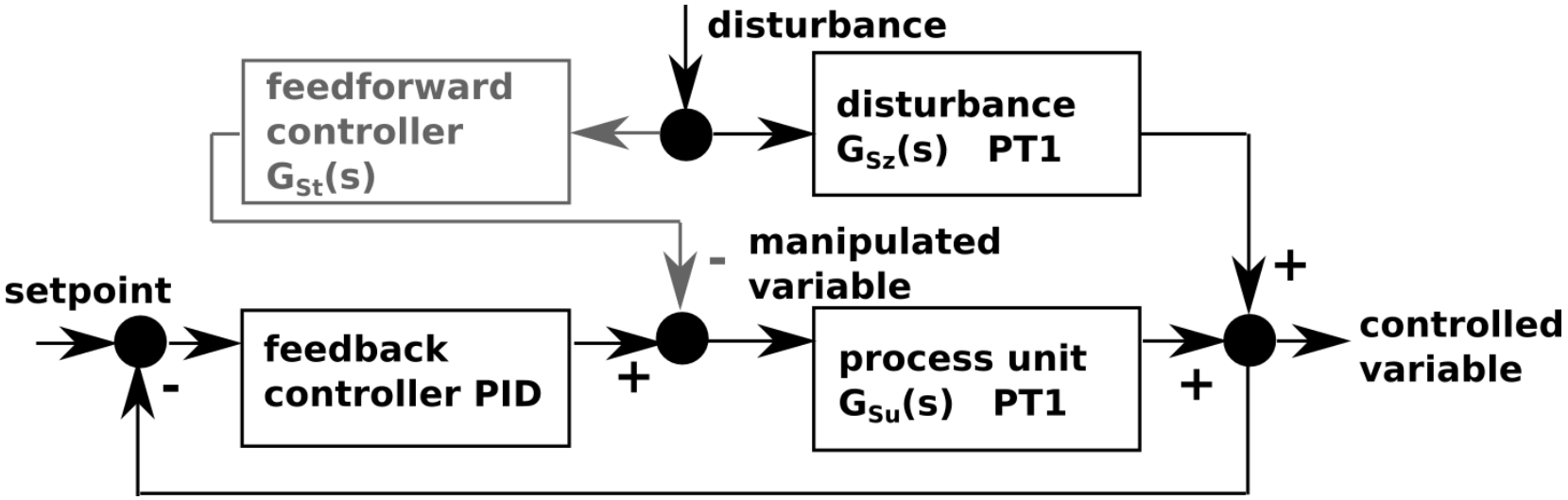
- Feedback control system
- Described by transfer functions
- Influenced by disturbances
- Measurable disturbances

  → Compensated with feedforward control

$$G_{Su}(s) = \frac{1}{20 \cdot s + 1}$$

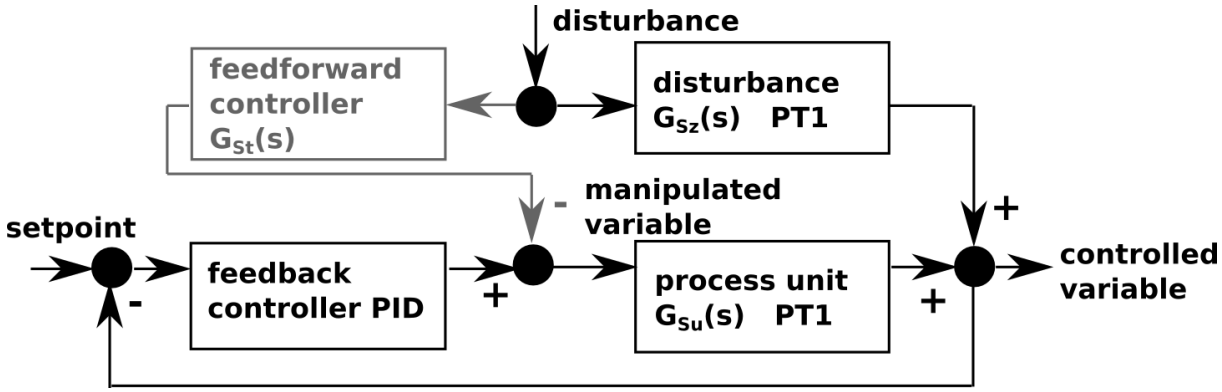
$$G_{Sz}(s) = \frac{1}{10 \cdot s + 1}$$

$$G_{St}(s) = \frac{20 \cdot s + 1}{10 \cdot s + 1}$$



Case Study (2)

- **Two system structure variants**
  - Without feedforward control: `feedforward=0`
  - With feedforward control: `feedforward=1`

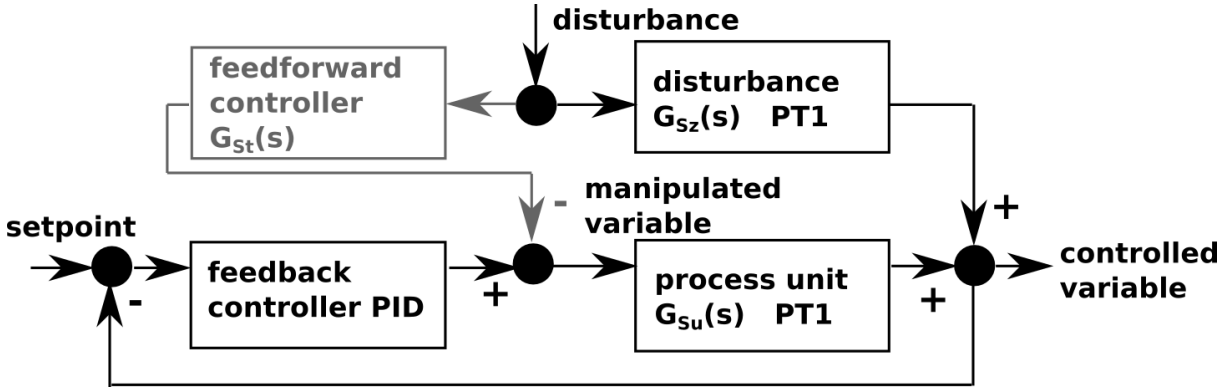


Case Study (2)

- **Two system structure variants**
  - Without feedforward control: `feedforward=0`
  - With feedforward control: `feedforward=1`
- For every structure variant

  → **Different parameter configurations of PID controller**

  (we consider two)



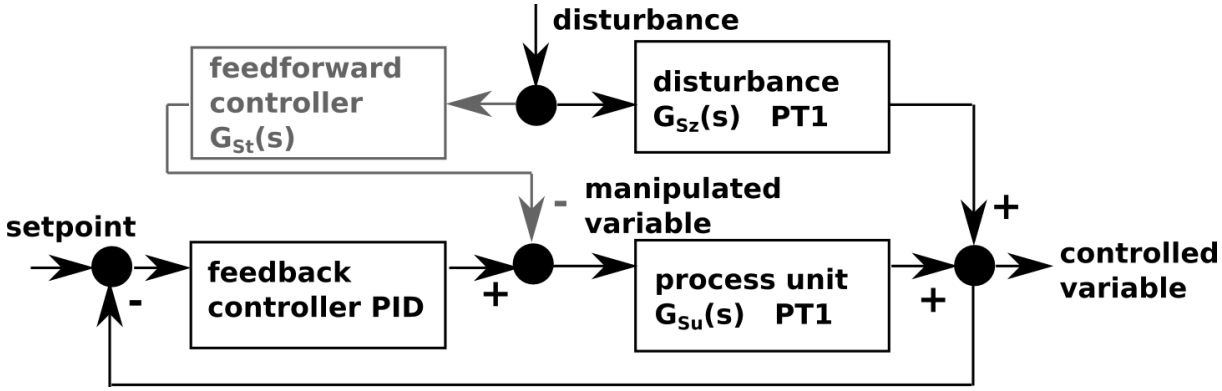
Case Study (2)

Design objective:
Find best control configuration.

- **Two system structure variants**
  - Without feedforward control: `feedforward=0`
  - With feedforward control: `feedforward=1`
- For every structure variant

  → **Different parameter configurations of PID controller**

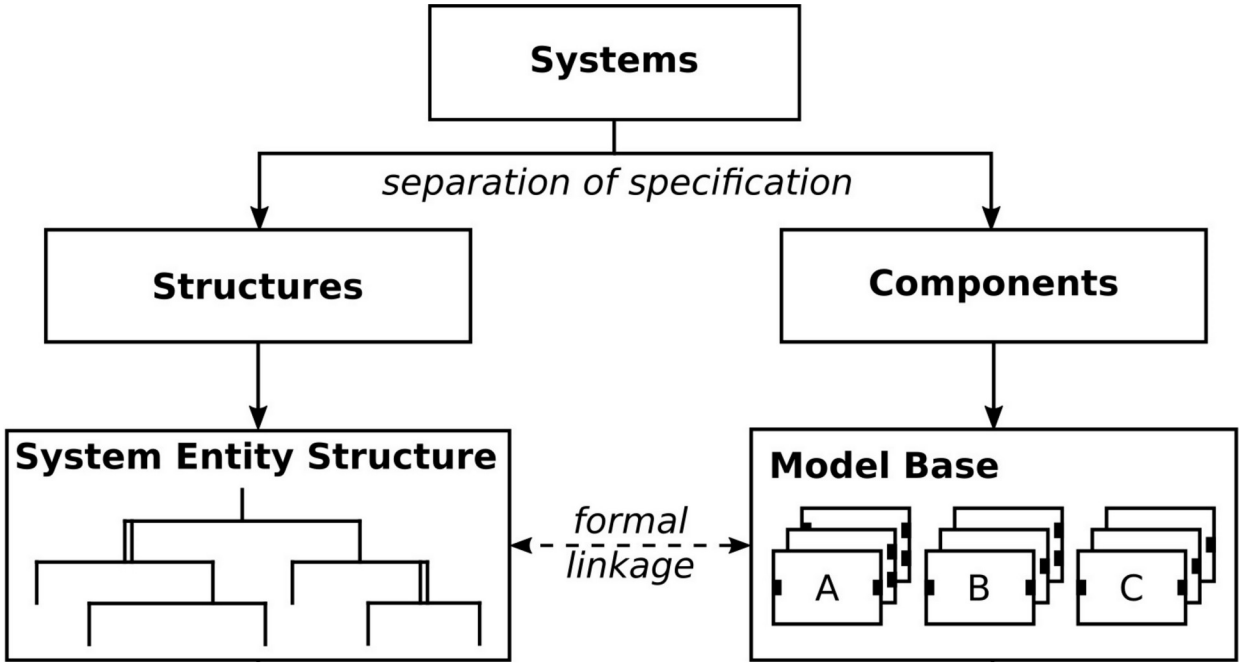
  (we consider two)



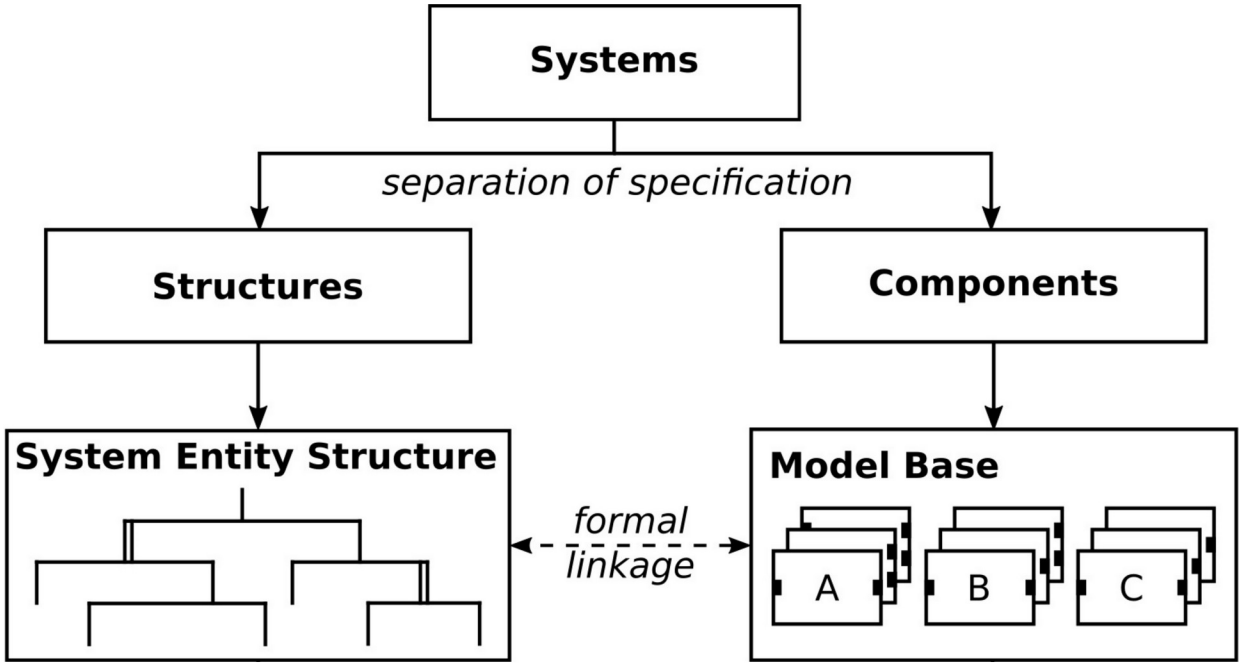
Outline

1. Introduction
2. The case study
3. **Basics of SES/MB based modeling** (T. Pawletta)
4. Practical modeling: implementation of an SES
5. Model selection and model generation
6. The free Matlab SES toolbox
7. Organization of a simulator-independent MB
8. Full automation of simulation experiments
9. Conclusion

SES/MB Modeling Approach

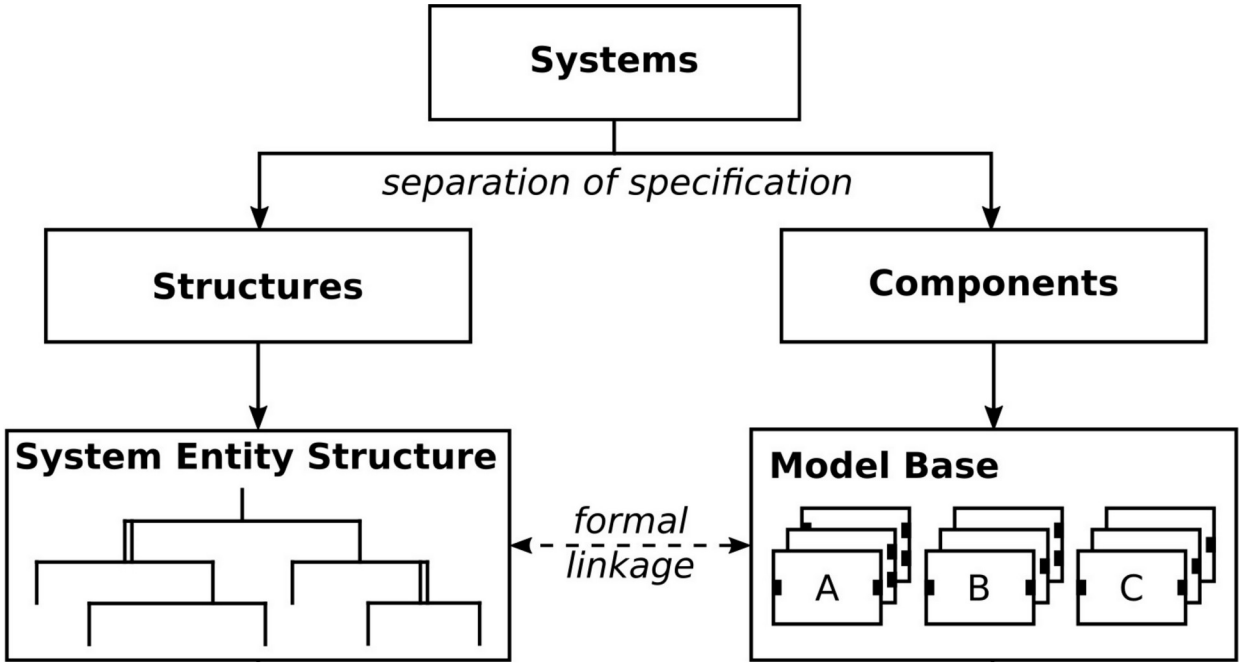


SES/MB Modeling Approach



- **SES** describes permissible structure & parameter variants (simulator-independent)

SES/MB Modeling Approach



- **SES** describes permissible structure & parameter variants (simulator-independent)
- **MB** defines basic dynamic models (usually simulator dependent)

Basics of the System Entity Structure (SES)

Basics of the System Entity Structure (SES)

- SES introduced by B.P. Zeigler and J. Rozenblit

Basics of the System Entity Structure (SES)

- SES introduced by B.P. Zeigler and J. Rozenblit
- Amongst others extended by research group CEA (Wismar)

Basics of the System Entity Structure (SES)

- SES introduced by B.P. Zeigler and J. Rozenblit
- Amongst others extended by research group CEA (Wismar)
- **SES is a tree structure**

Basics of the System Entity Structure (SES)

- SES introduced by B.P. Zeigler and J. Rozenblit
- Amongst others extended by research group CEA (Wismar)
- **SES is a tree structure**
  - Well defined by axioms

Basics of the System Entity Structure (SES)

- SES introduced by B.P. Zeigler and J. Rozenblit
- Amongst others extended by research group CEA (Wismar)
- **SES is a tree structure**
  - Well defined by axioms
  - Two types of nodes
    - Entity nodes
    - Descriptive nodes

Basics of the System Entity Structure (SES)

- SES introduced by B.P. Zeigler and J. Rozenblit
- Amongst others extended by research group CEA (Wismar)
- **SES is a tree structure**
  - Well defined by axioms
  - Two types of nodes
    - Entity nodes
    - Descriptive nodes

Entity nodes
real or imaginary objects

Descriptive nodes
Aspect
(Multi-aspect)
Specialization

Basics of the System Entity Structure (SES)

- SES introduced by B.P. Zeigler and J. Rozenblit
- Amongst others extended by research group CEA (Wismar)
- **SES is a tree structure**
  - Well defined by axioms
  - Two types of nodes
    - Entity nodes
    - Descriptive nodes
  - Three types of edges (relations betw. nodes)

Entity nodes
real or imaginary objects

Descriptive nodes
Aspect
(Multi-aspect)
Specialization

Basics of the System Entity Structure (SES)

- SES introduced by B.P. Zeigler and J. Rozenblit
- Amongst others extended by research group CEA (Wismar)
- **SES is a tree structure**
  - Well defined by axioms
  - Two types of nodes
    - Entity nodes
    - Descriptive nodes
  - Three types of edges (relations betw. nodes)
  - Node/Edge specific attributes

Entity nodes
real or imaginary objects

Descriptive nodes
Aspect
(Multi-aspect)
Specialization

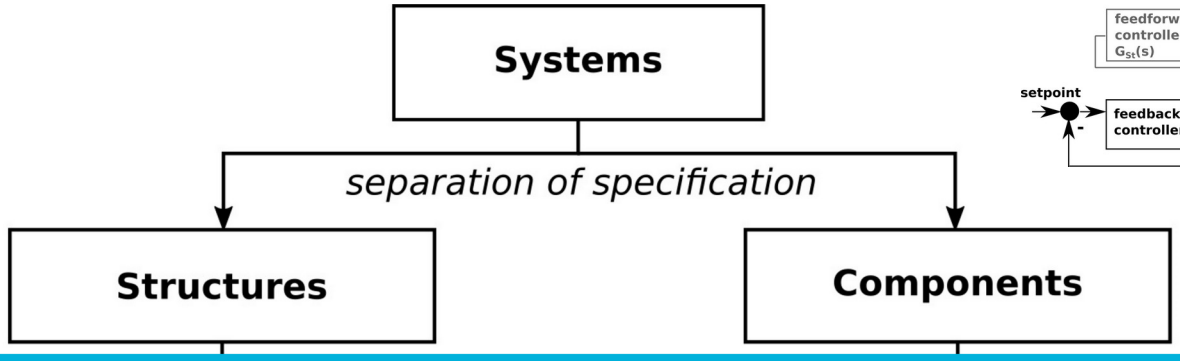
Basics of the System Entity Structure (SES)

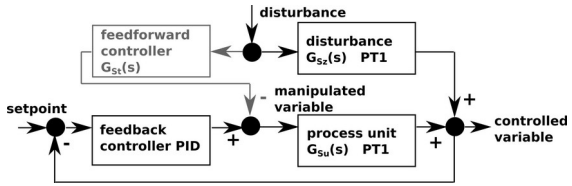
- SES introduced by B.P. Zeigler and J. Rozenblit
- Amongst others extended by research group CEA (Wismar)
- **SES is a tree structure**
  - Well defined by axioms
  - Two types of nodes
    - Entity nodes
    - Descriptive nodes
  - Three types of edges (relations betw. nodes)
  - Node/Edge specific attributes
  - Global variables, functions, constraints, ...

Entity nodes
real or imaginary objects

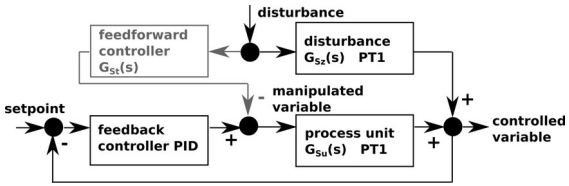
Descriptive nodes
Aspect
(Multi-aspect)
Specialization

SES/MB-based Modeling of the Case Study

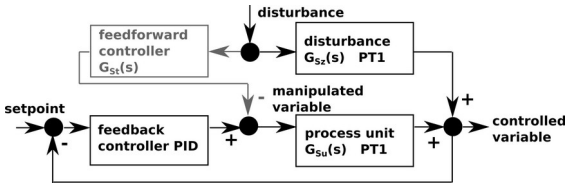




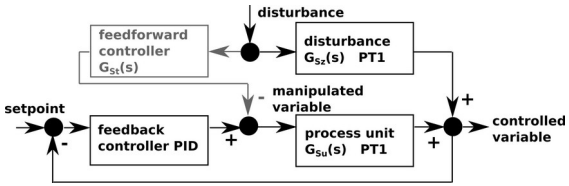
SES/MB-based Modeling of the Case Study

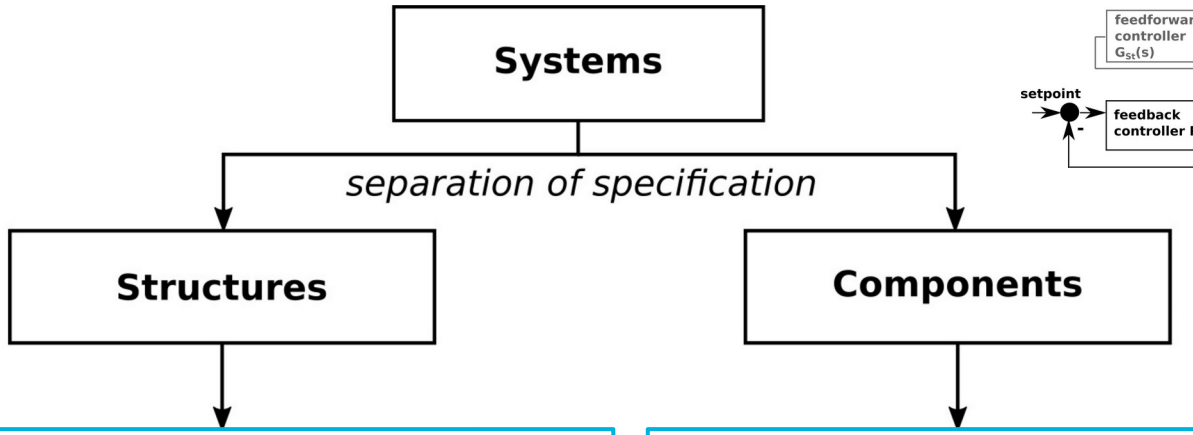


SES/MB-based Modeling of the Case Study

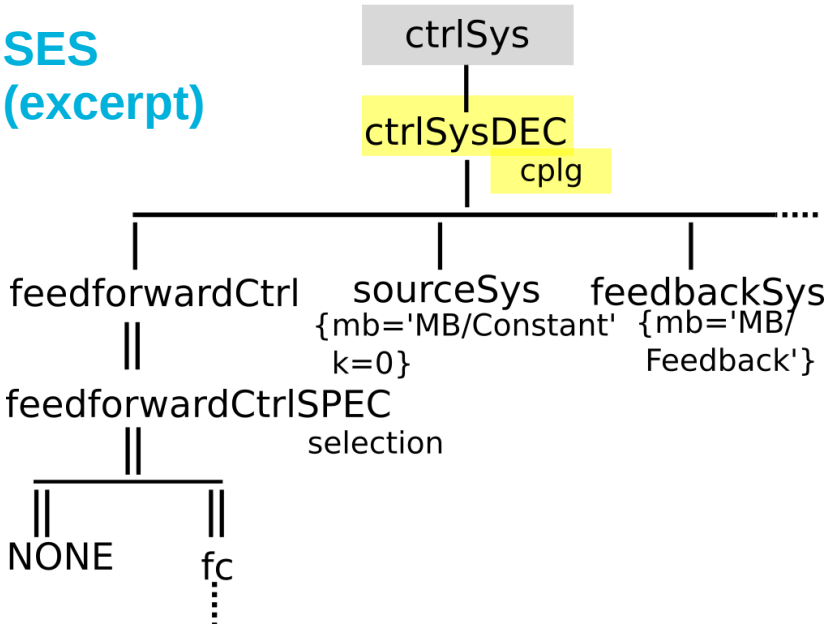


SES/MB-based Modeling of the Case Study

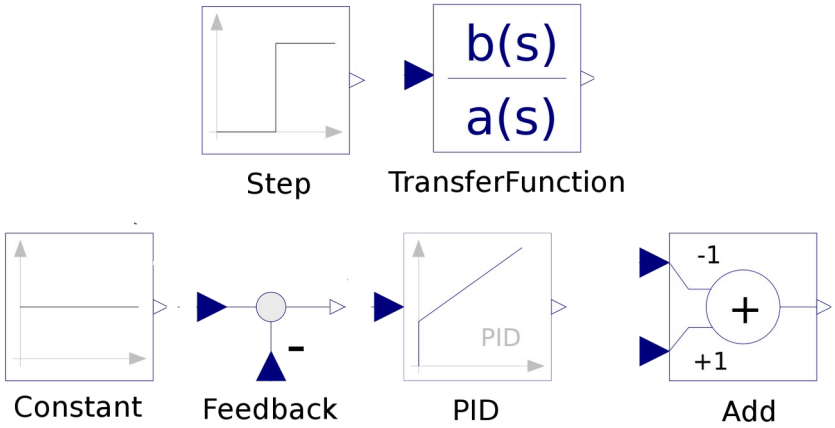




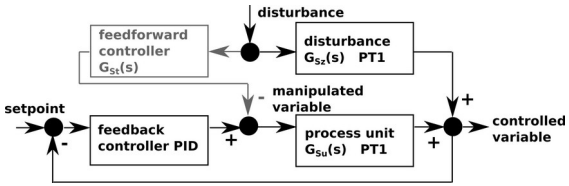
SES (excerpt)



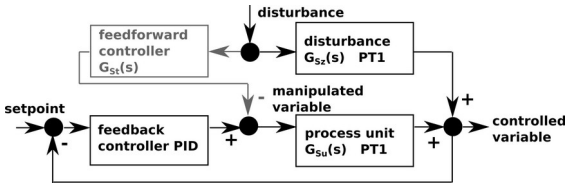
MB



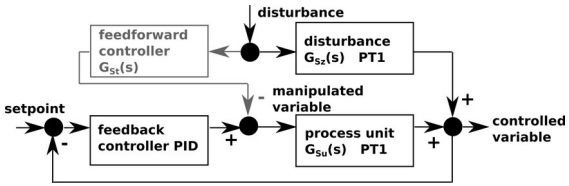
SES/MB-based Modeling of the Case Study



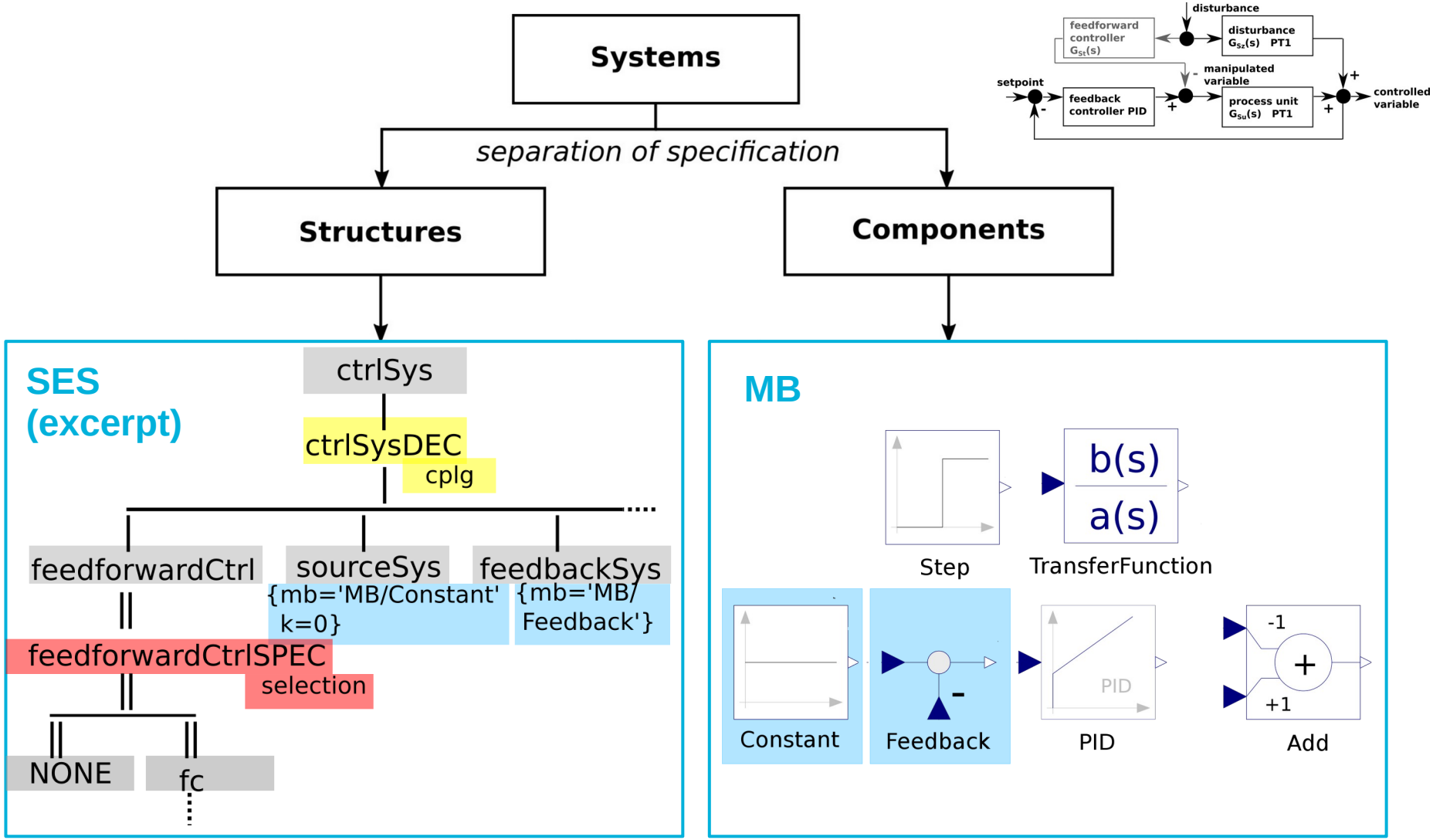
SES/MB-based Modeling of the Case Study



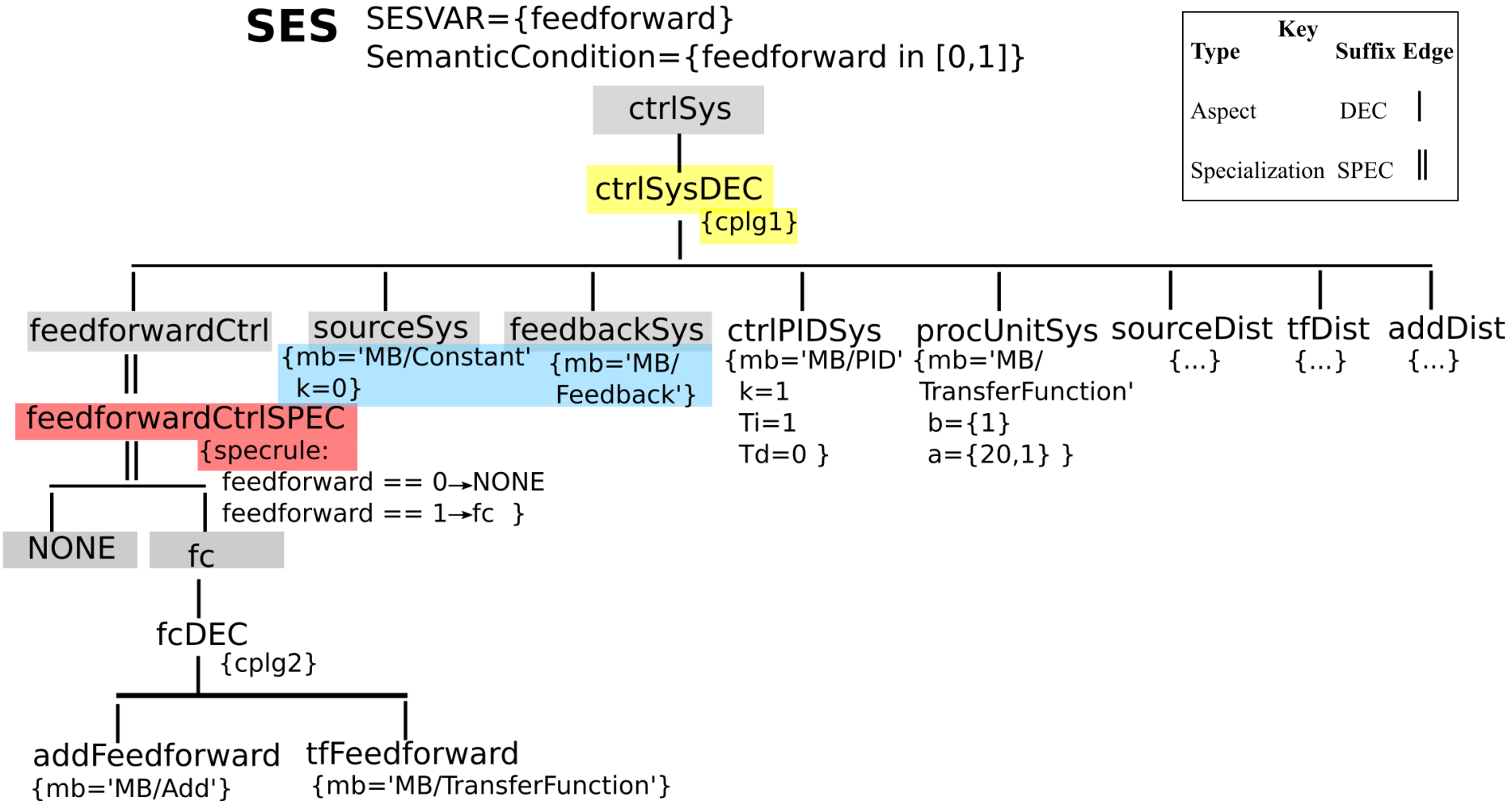
SES/MB-based Modeling of the Case Study



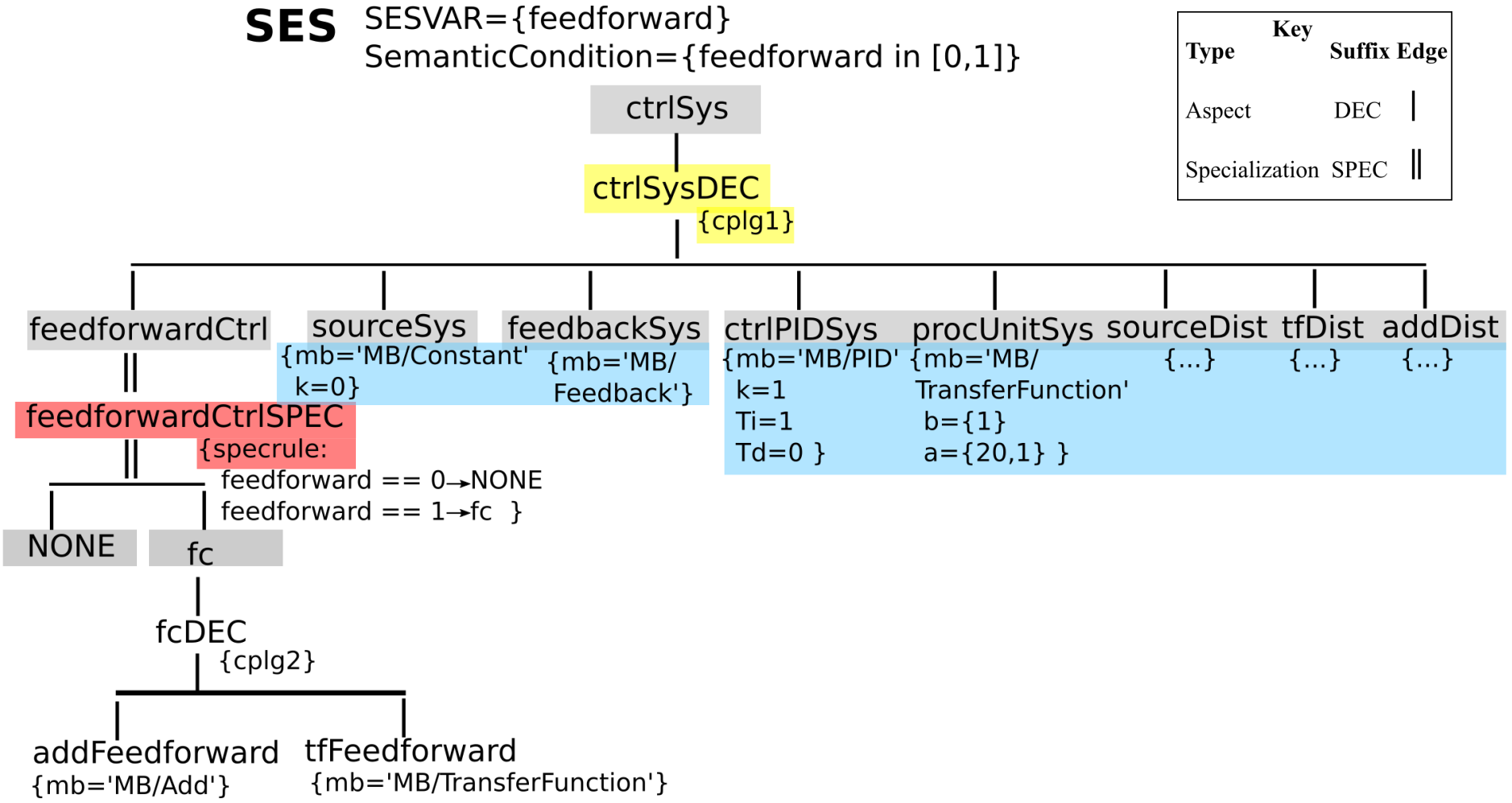
SES/MB-based Modeling of the Case Study



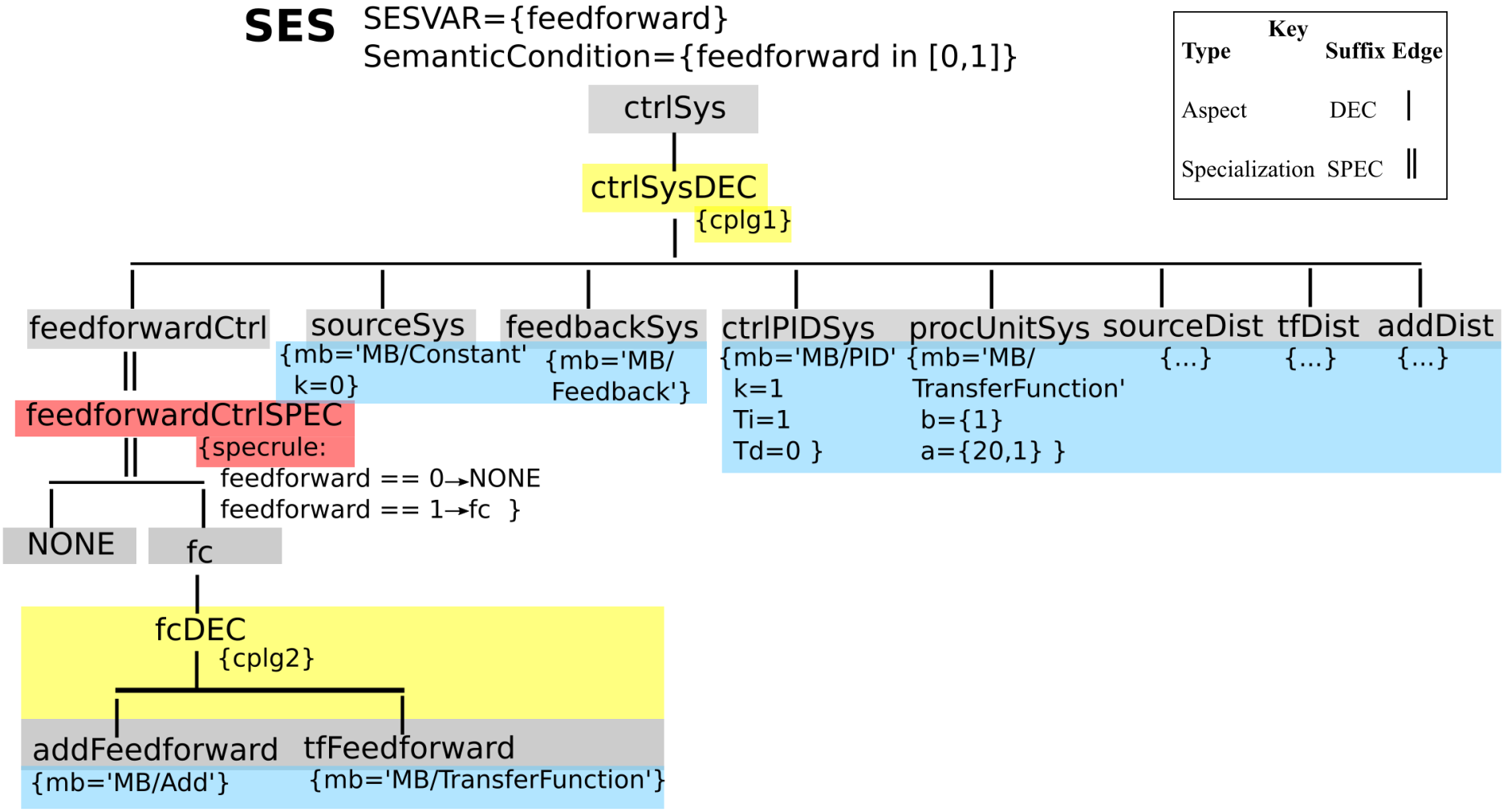
More Detailed Extract of the SES



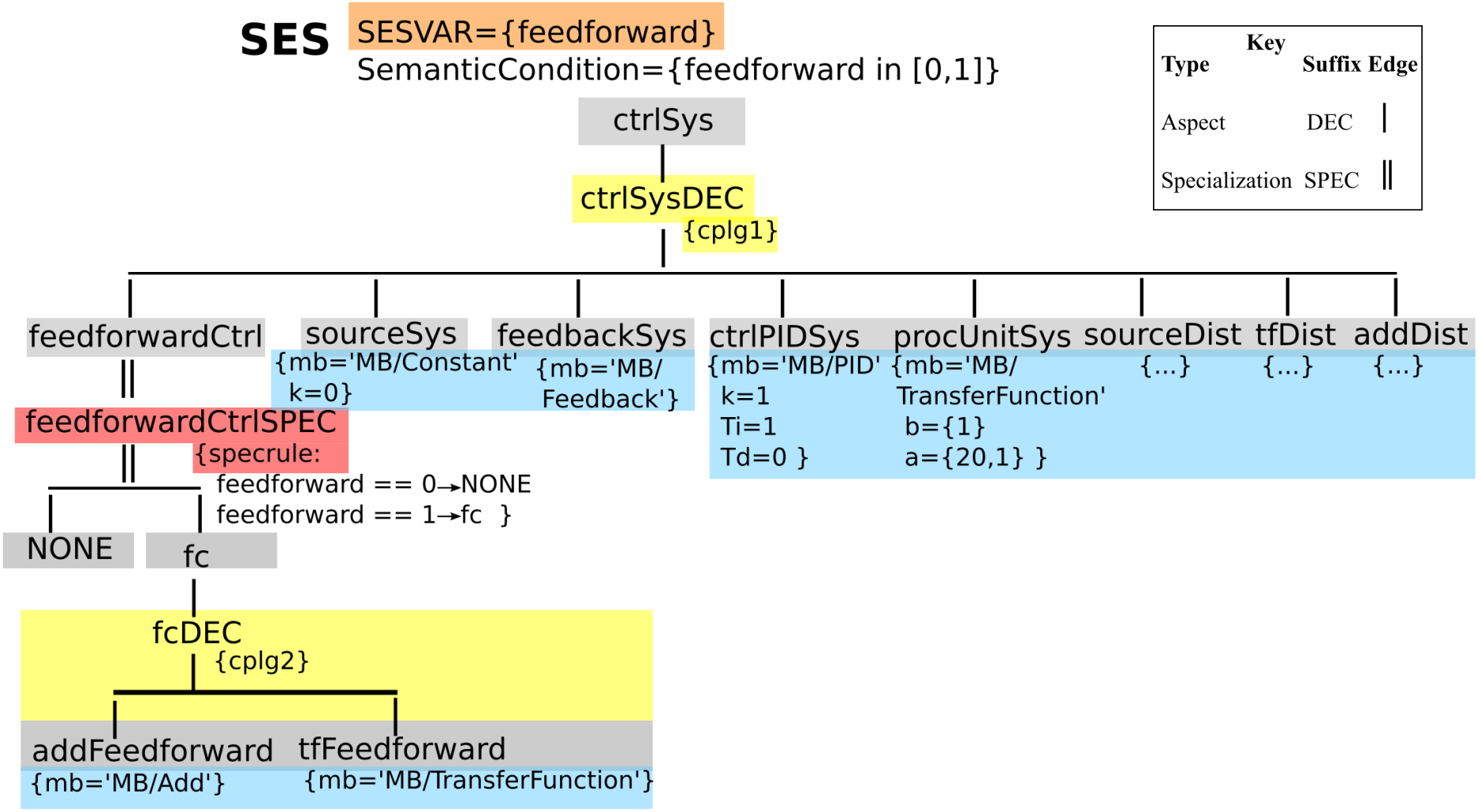
More Detailed Extract of the SES



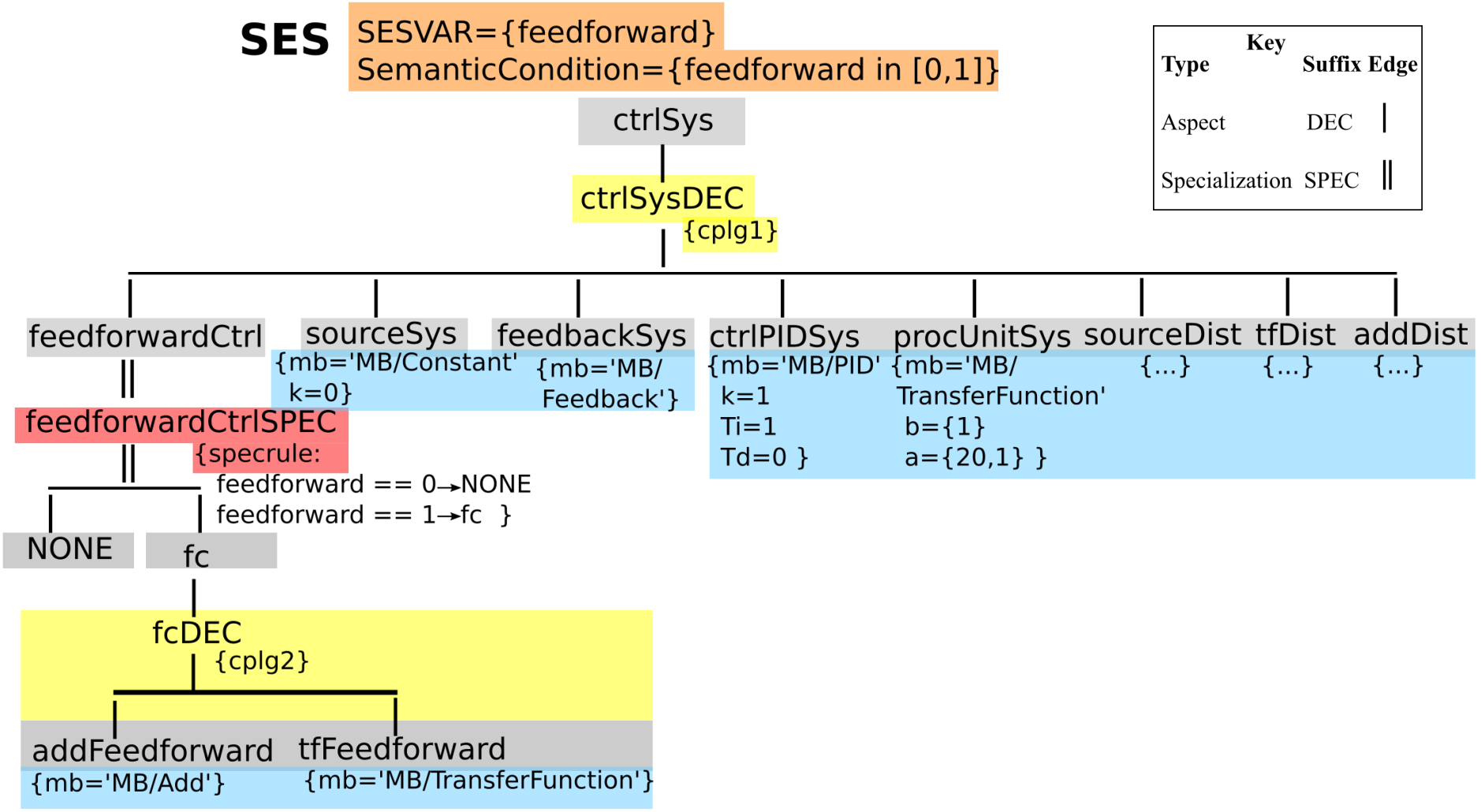
More Detailed Extract of the SES



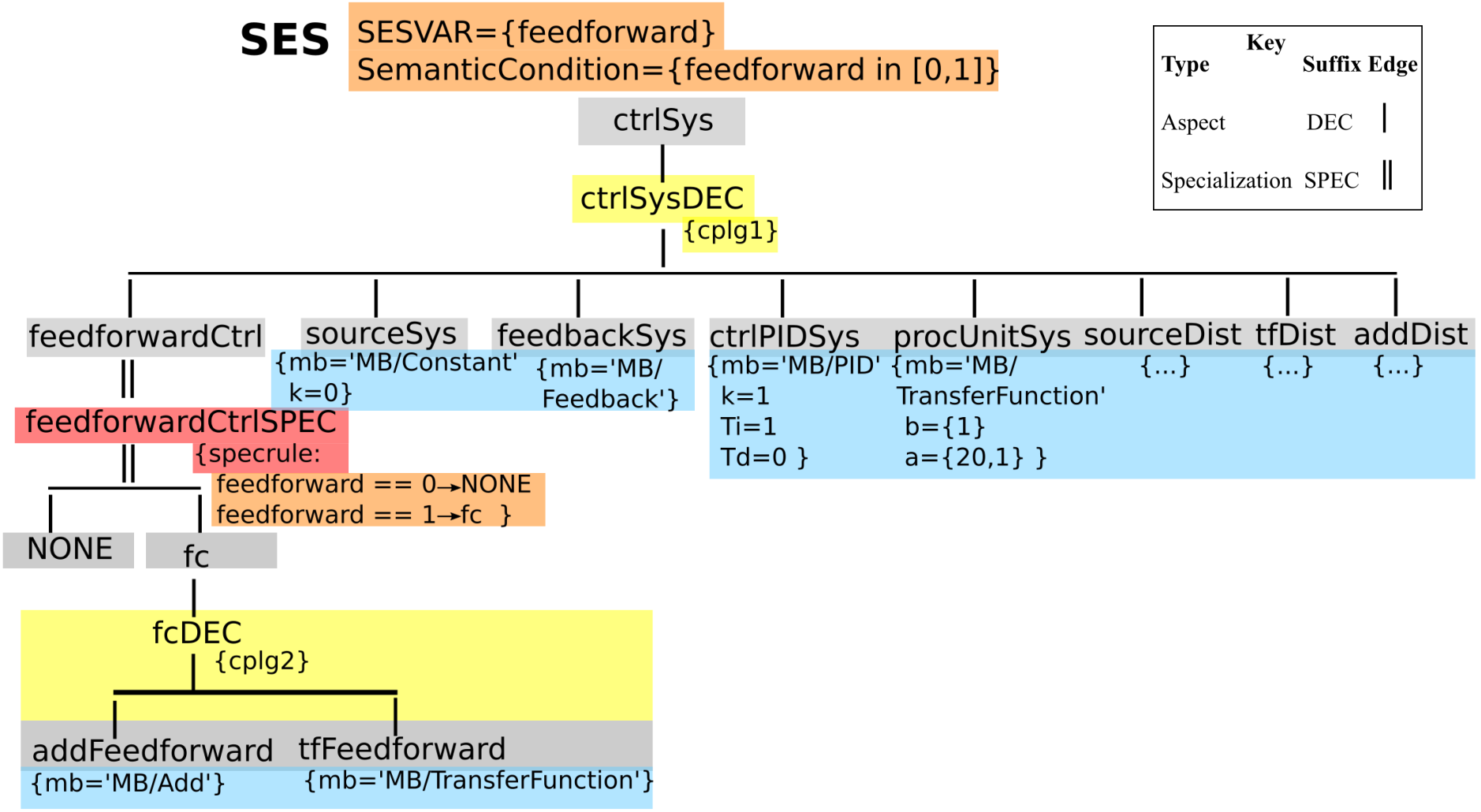
More Detailed Extract of the SES



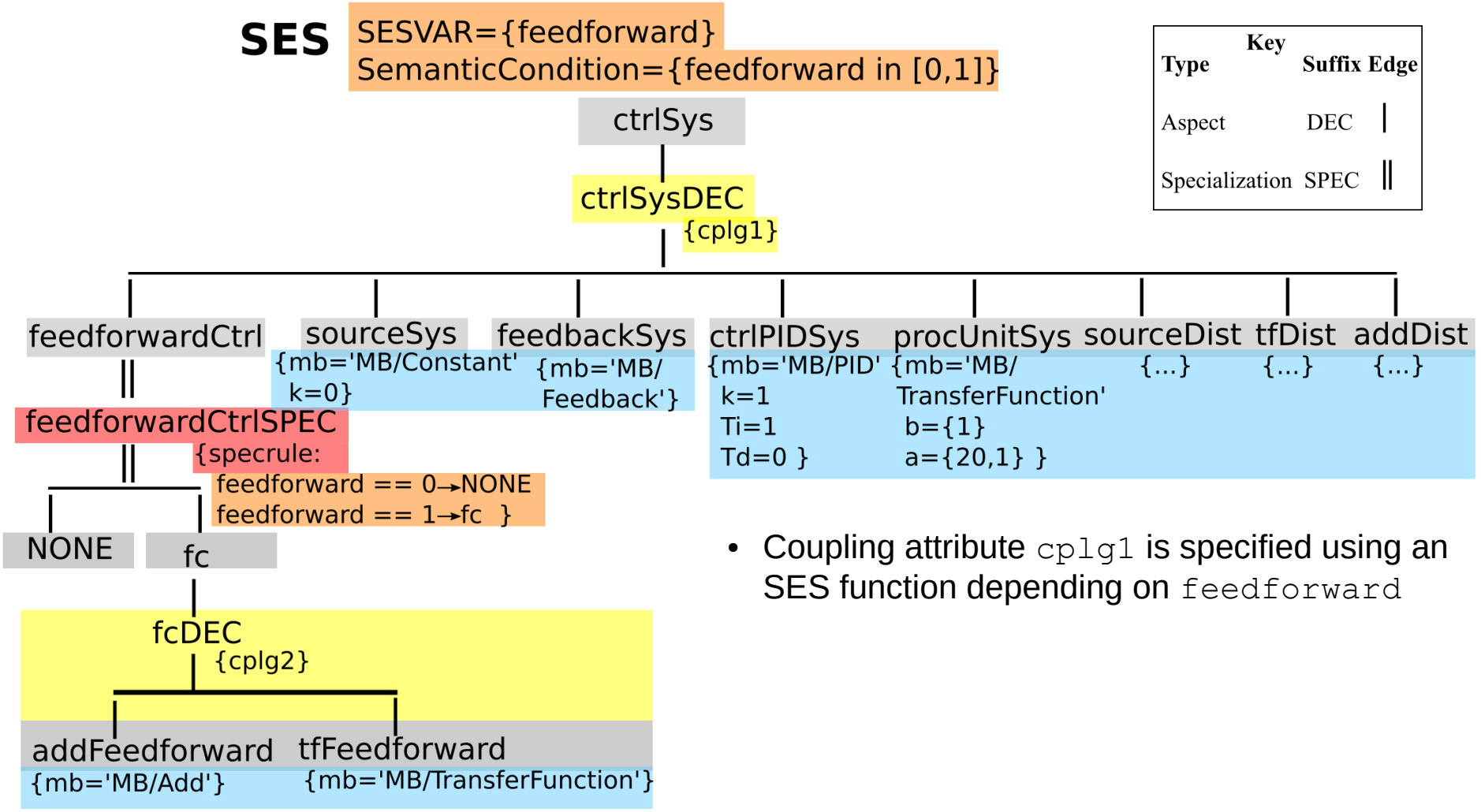
More Detailed Extract of the SES



More Detailed Extract of the SES

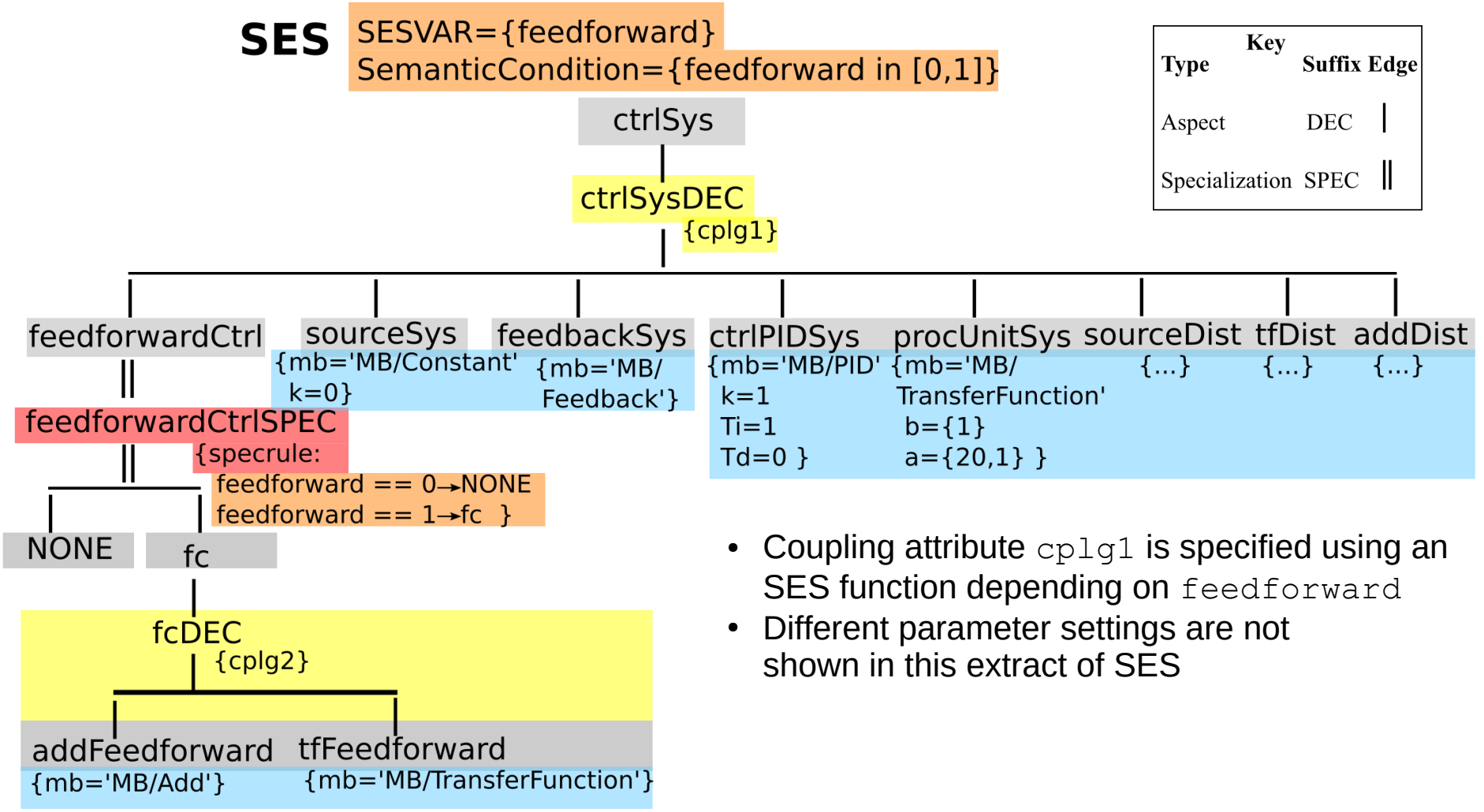


More Detailed Extract of the SES



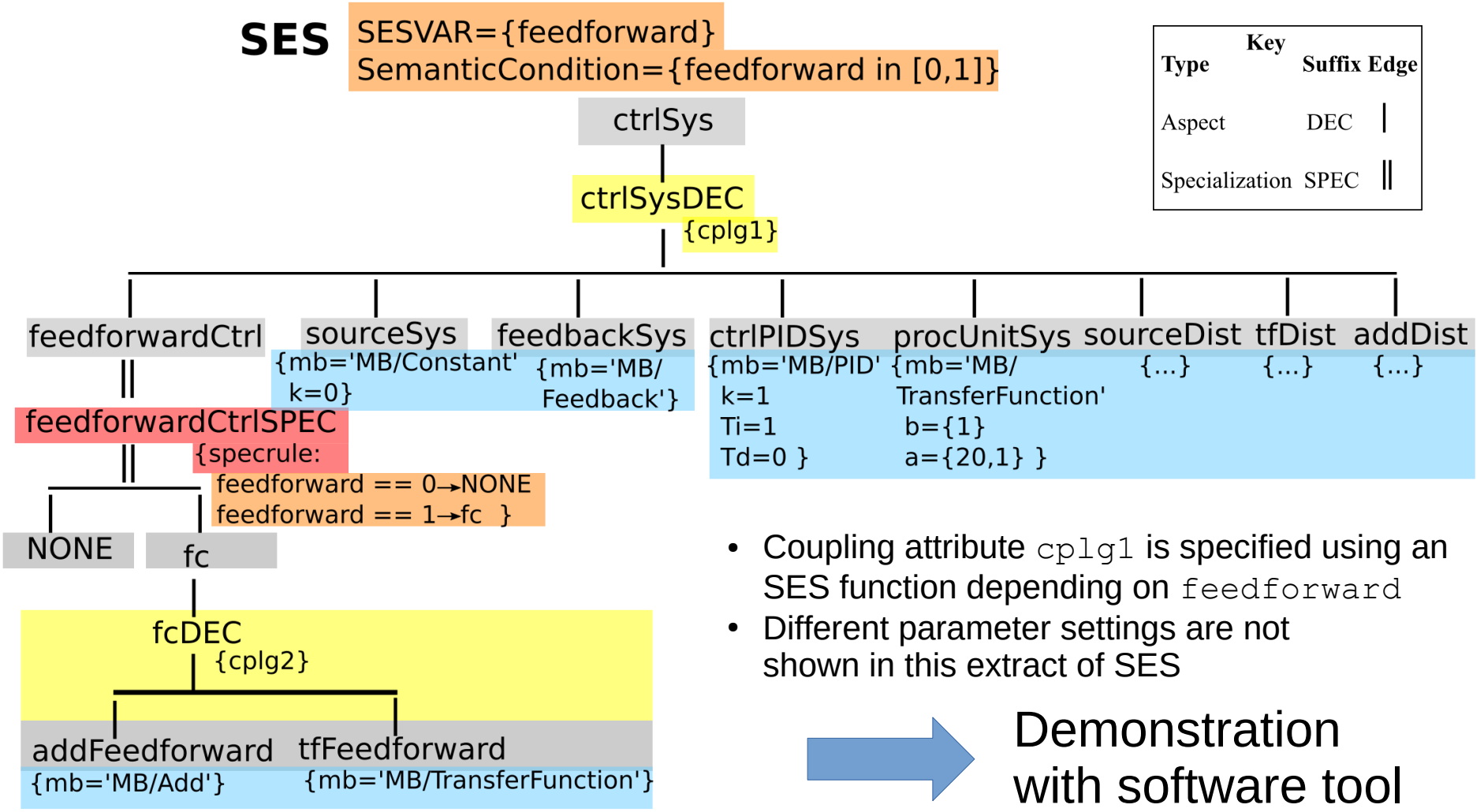
- Coupling attribute `cplg1` is specified using an SES function depending on `feedforward`

More Detailed Extract of the SES



- Coupling attribute `cplg1` is specified using an SES function depending on `feedforward`
- Different parameter settings are not shown in this extract of SES

More Detailed Extract of the SES



- Coupling attribute `cplg1` is specified using an SES function depending on `feedforward`
- Different parameter settings are not shown in this extract of SES

Demonstration with software tool

Outline

1. Introduction
2. The case study
3. Basics of SES/MB based modeling
4. **Practical modeling: implementation of an SES**
   (H. Folkerts)
5. Model selection and model generation
6. The free Matlab SES toolbox
7. Organization of a simulator-independent MB
8. Full automation of simulation experiments
9. Conclusion

Python Toolset

- Available: `https://github.com/hendrikfolkerts`
- Tools
  - **SESToPy** → SES editor and IDE
  - **SESViewEl** → SES tree viewer
  - SESMoPy
  - SESEuPy
  - SESEcPy

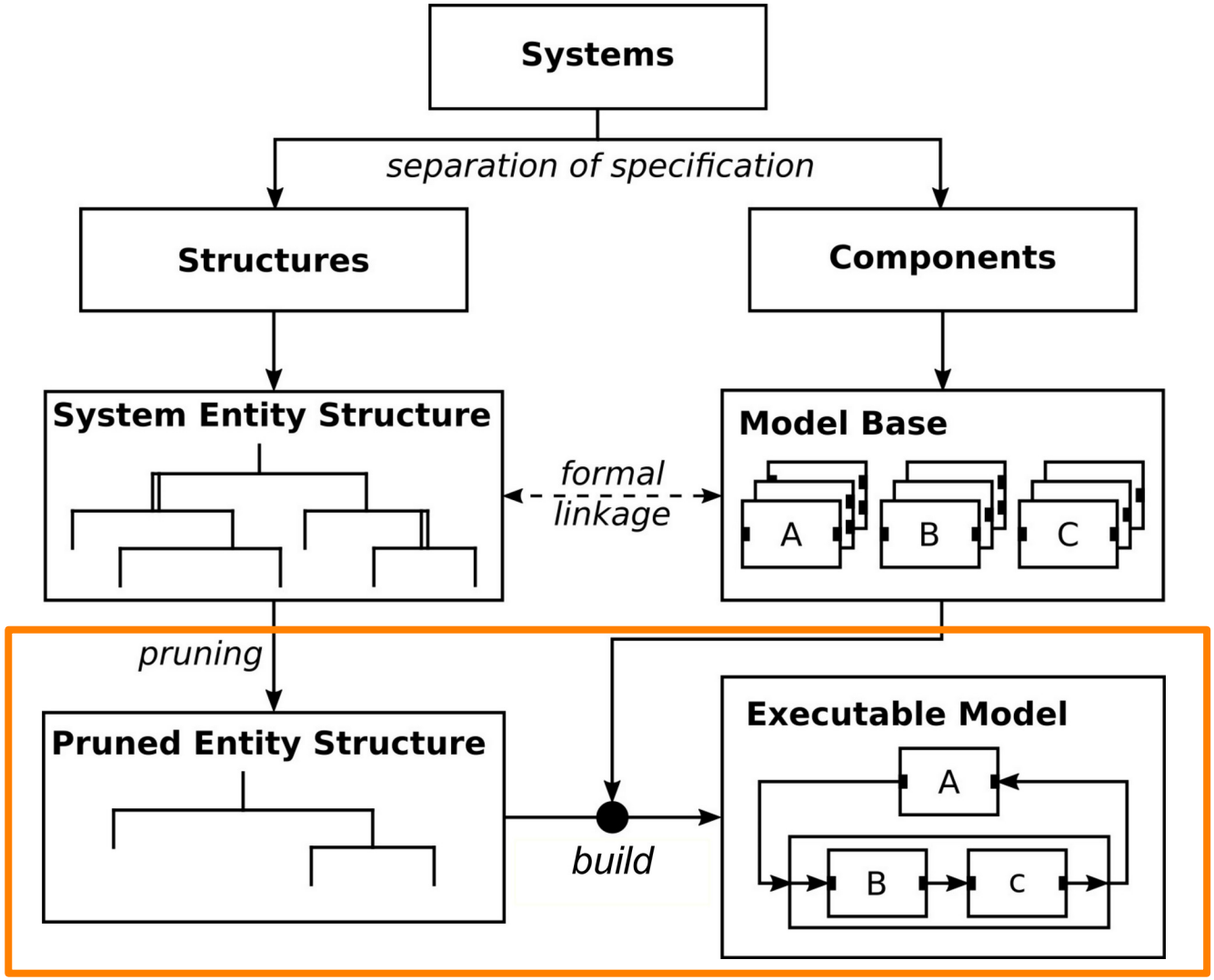
Demonstration of SESToPy with SESViewEl (case study)

- Connect SESToPy with SESViewEl (show SESToPy and SESViewEl next to each other)
- Add sub node, add sibling node, change type of node, rename node, delete node, inflate tree, deflate tree
- Edit entity node, descriptive node (aspect, specialization)
- Empty current model
- Save/Load model (JSON) → load `Feedback.jsonsestree` example
- (Export/Import model (XML))
- Maximize SESToPy
- Use the feedback example to show:
  - SES Variables, Semantic Conditions
  - Selection rules → here: specrule
  - NONE node
  - Attributes, mb-attribute (decouple name of node and name of basic model)
  - Coupling list (composition of basic models)
  - SES function to set couplings (dynamic coupling) → procedural knowledge
- Mention merging

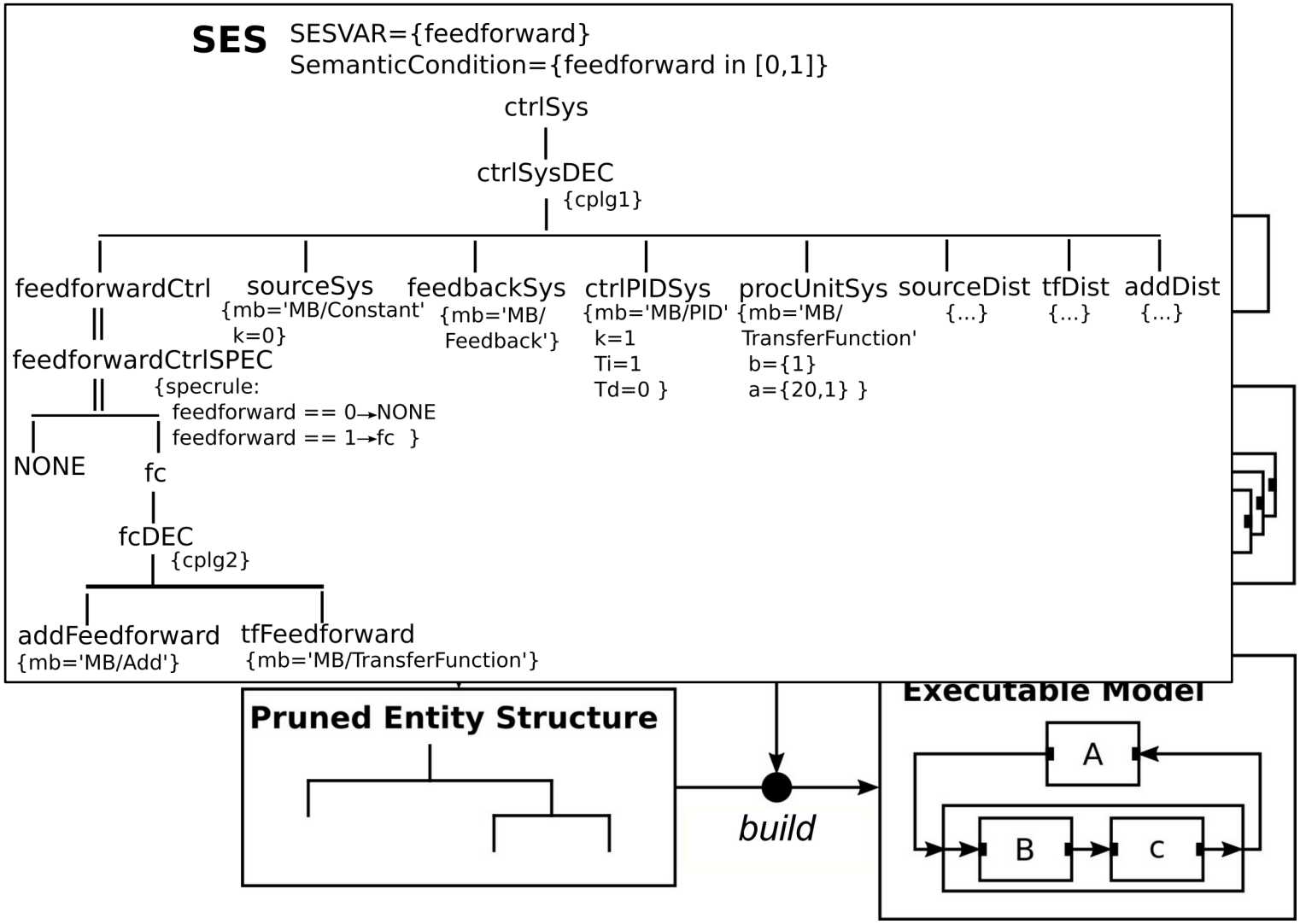
Outline

1. Introduction
2. The case study
3. Basics of SES/MB based modeling
4. Practical modeling: implementation of an SES
5. **Model selection and model generation** (H. Folkerts)
6. The free Matlab SES toolbox
7. Organization of a simulator-independent MB
8. Full automation of simulation experiments
9. Conclusion

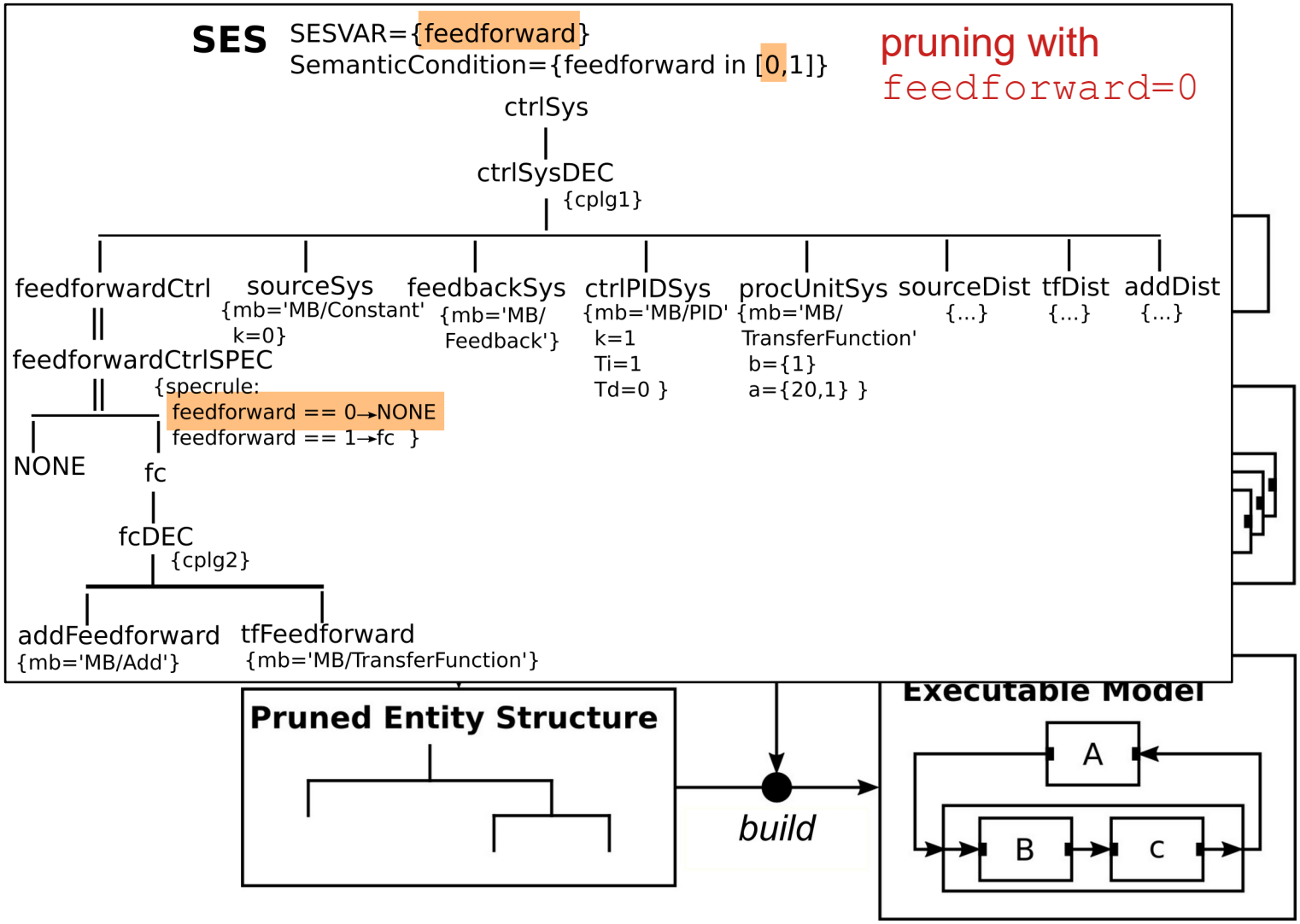
Model Selection and Generation



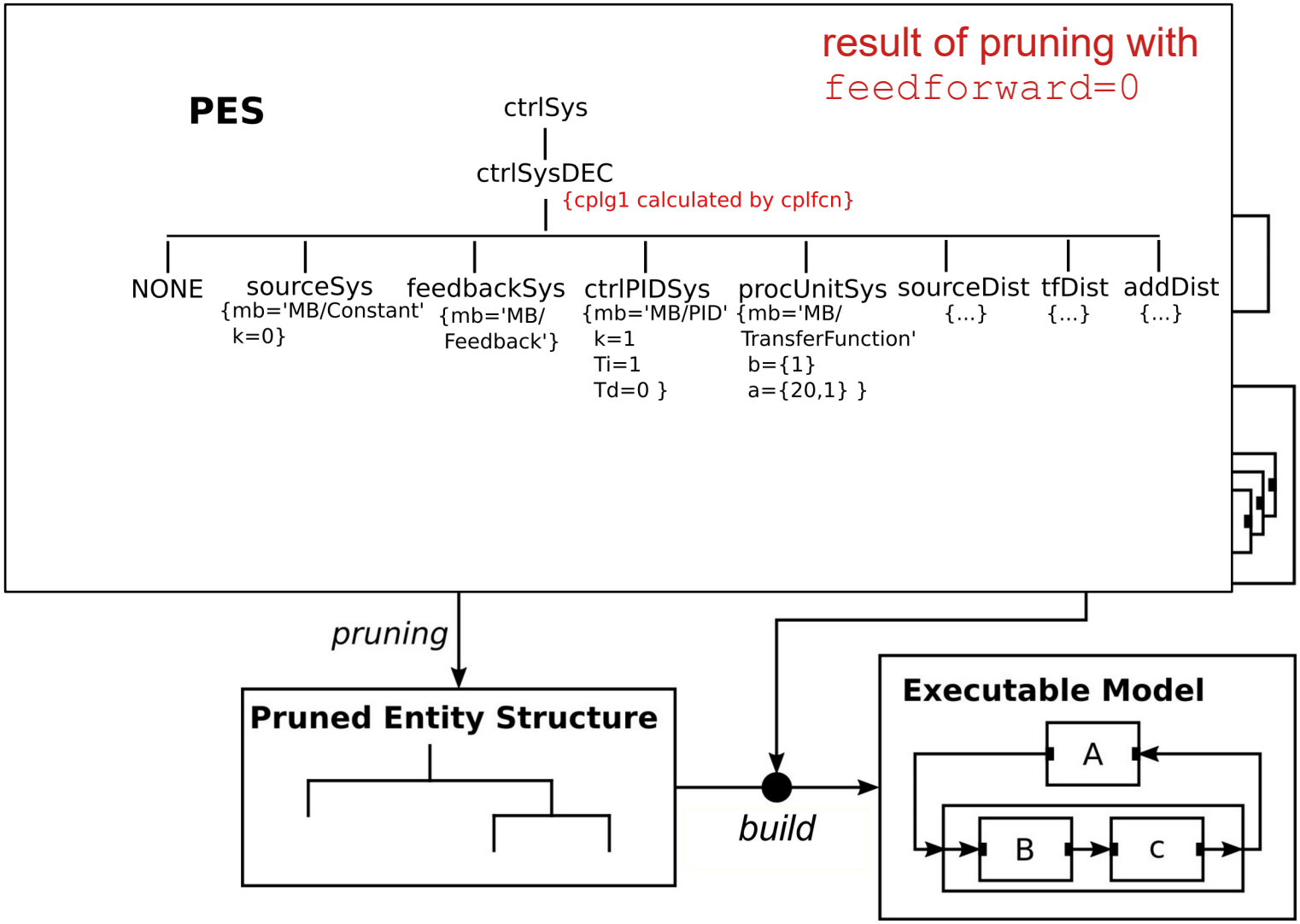
Model Selection and Generation of Variant #1



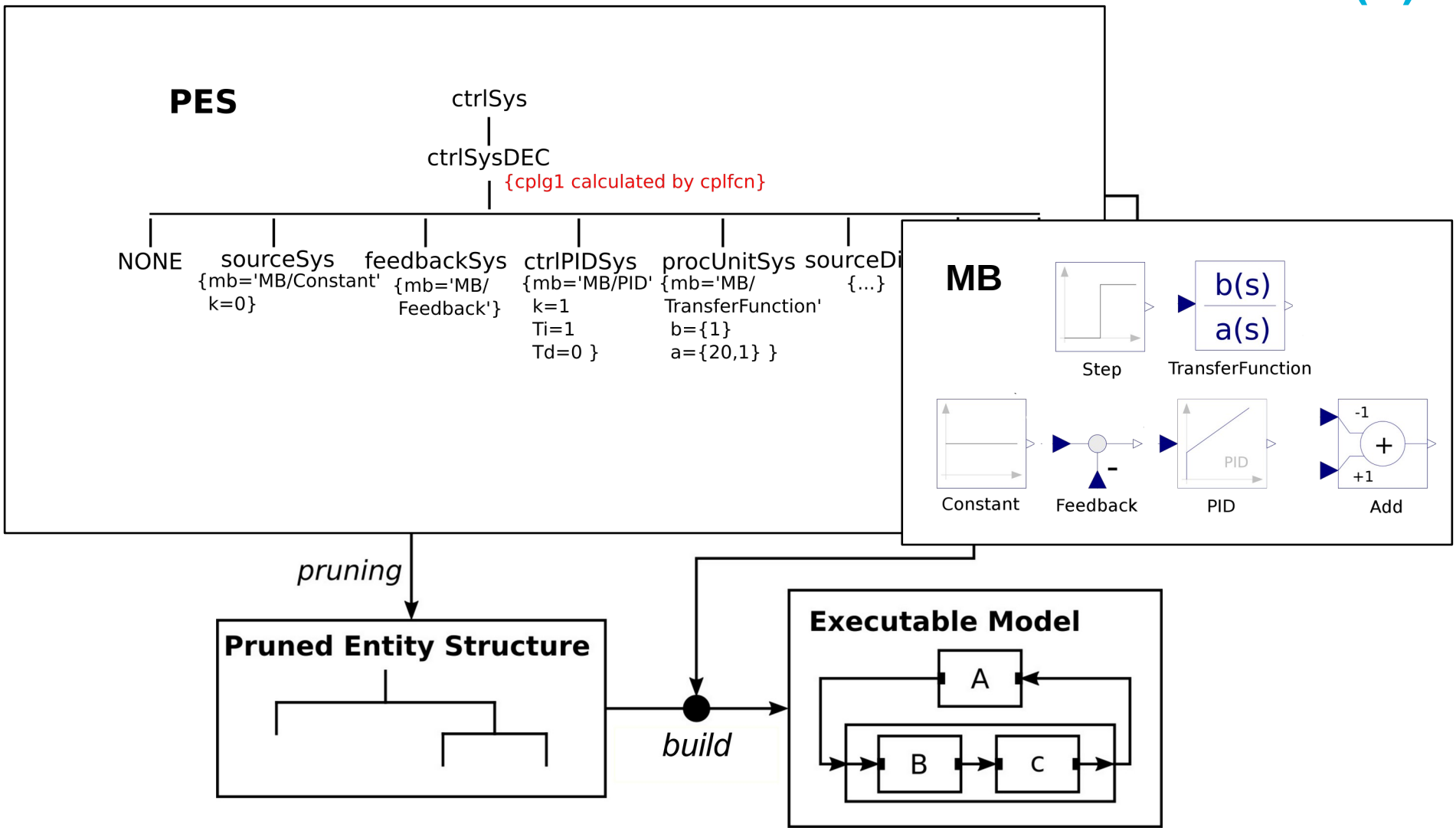
Model Selection and Generation of Variant #1



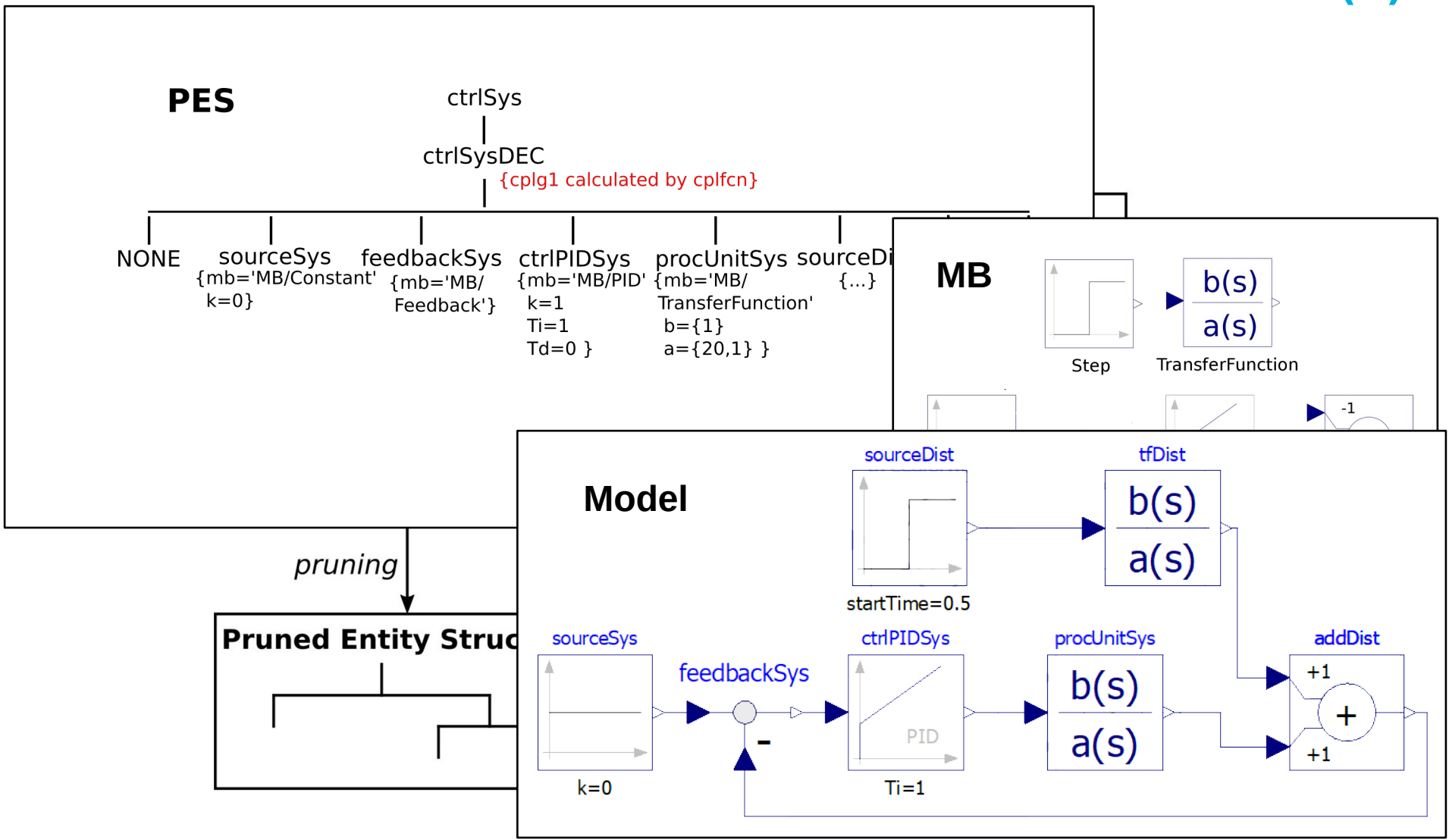
Model Selection and Generation of Variant #1 (2)



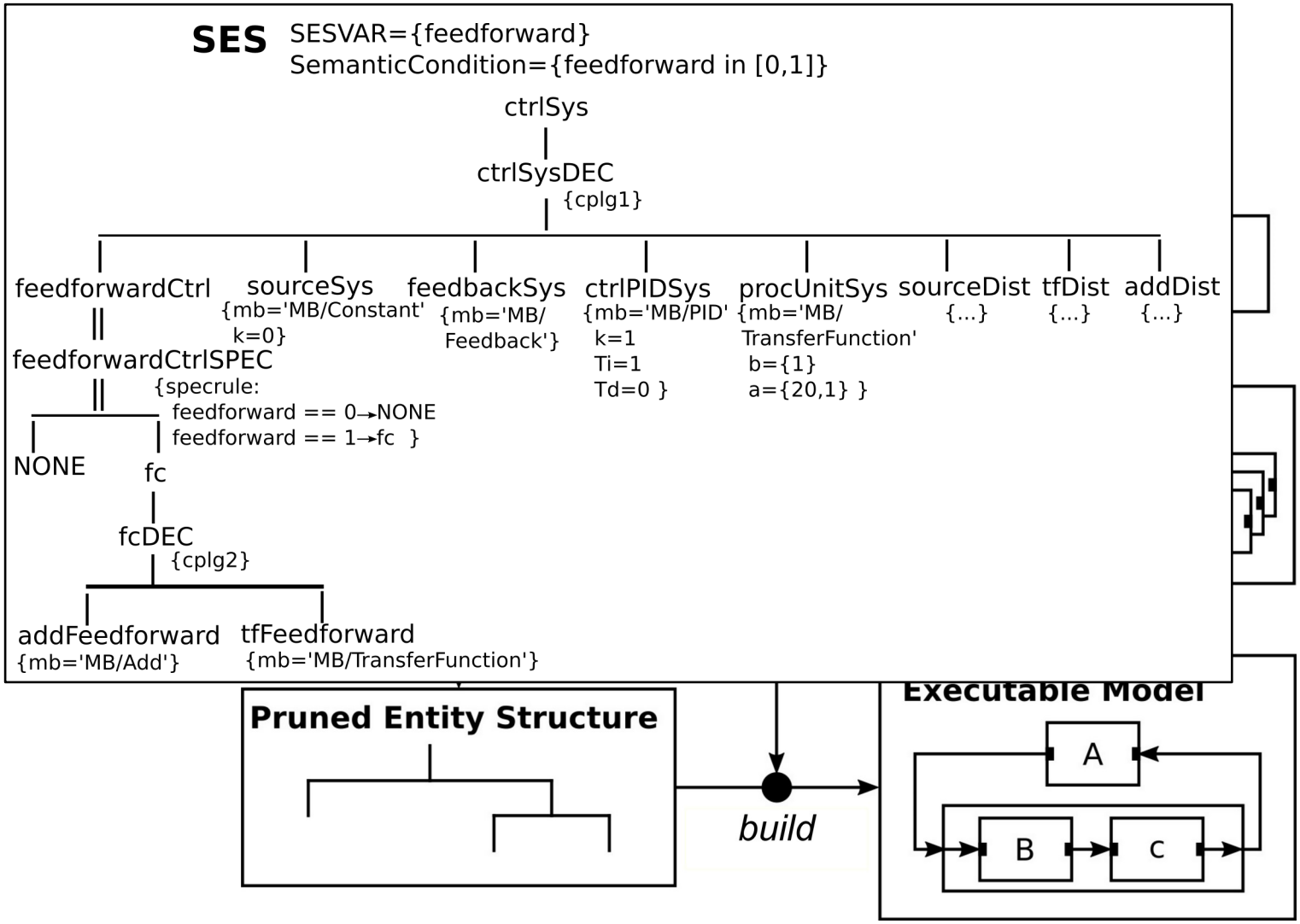
Model Selection and Generation of Variant #1 (3)



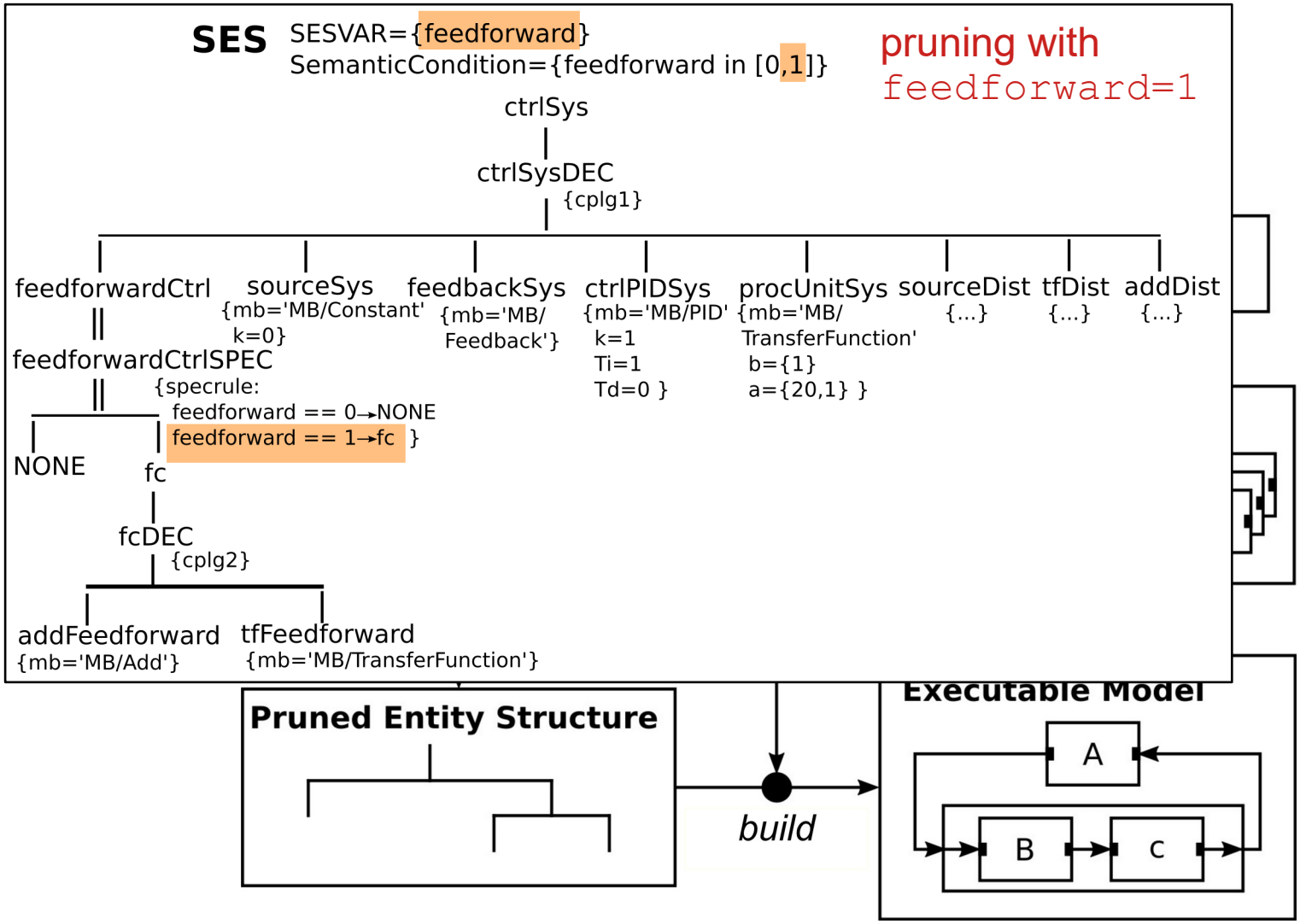
Model Selection and Generation of Variant #1 (4)



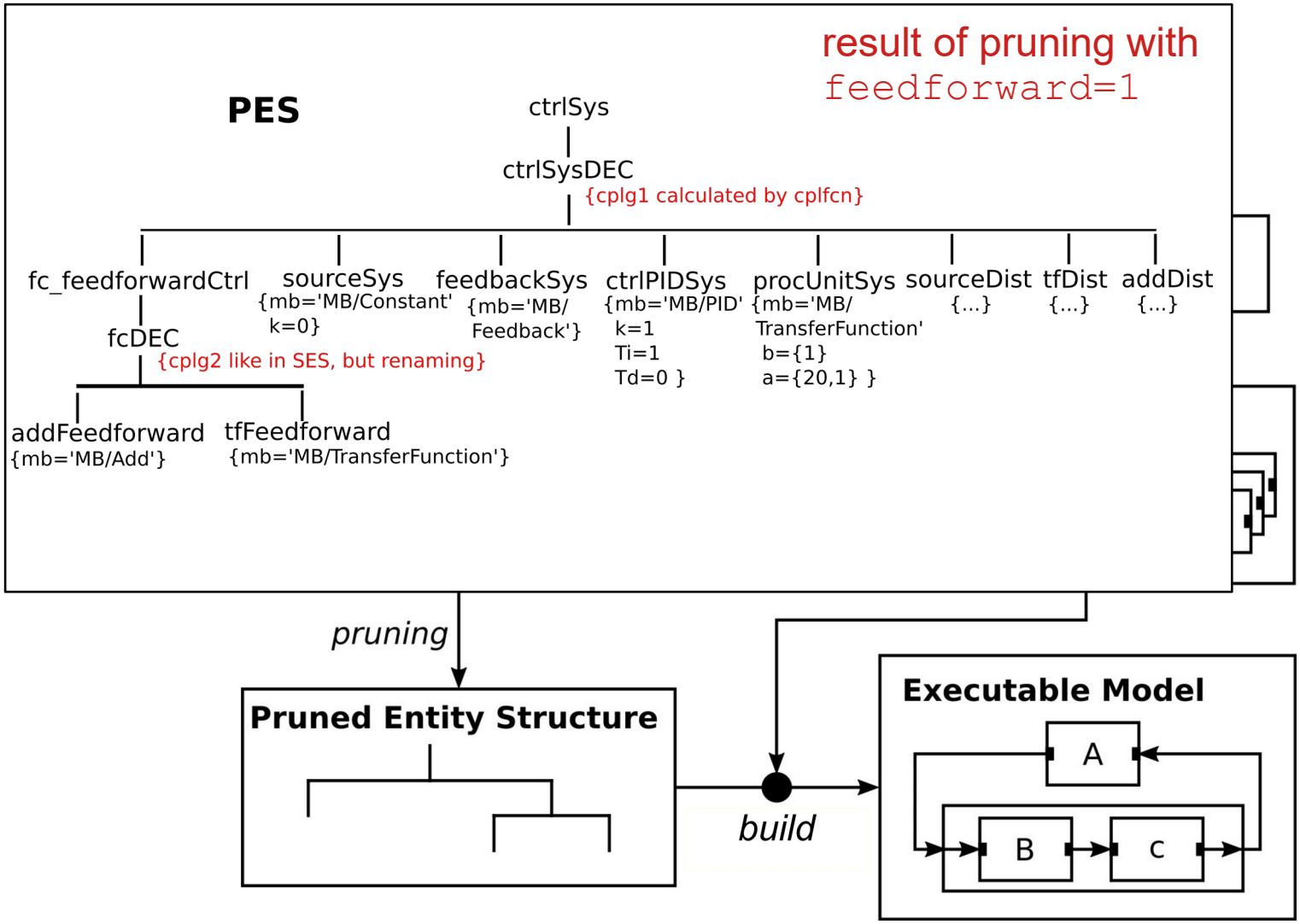
Model Selection and Generation of Variant #2



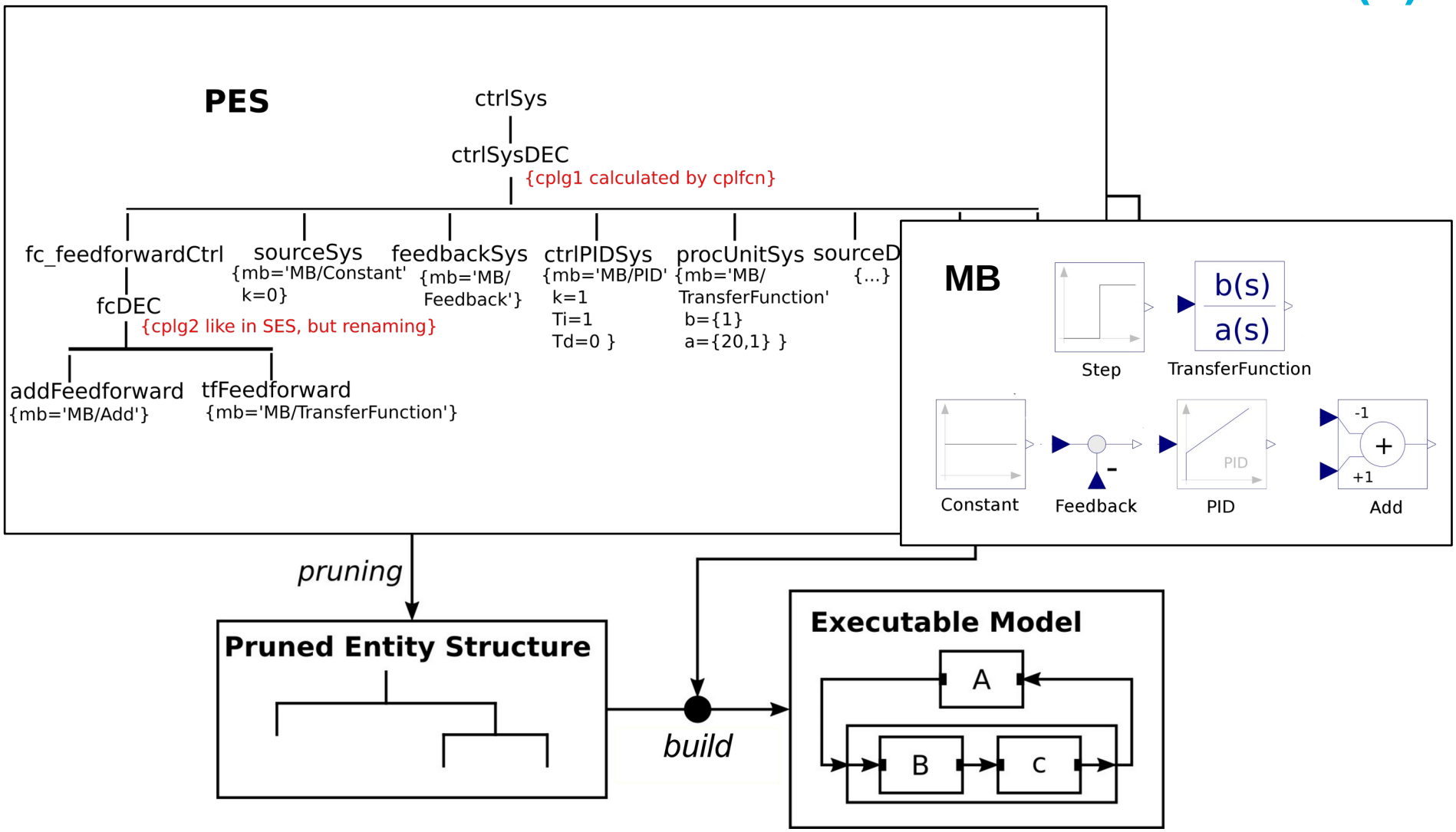
Model Selection and Generation of Variant #2



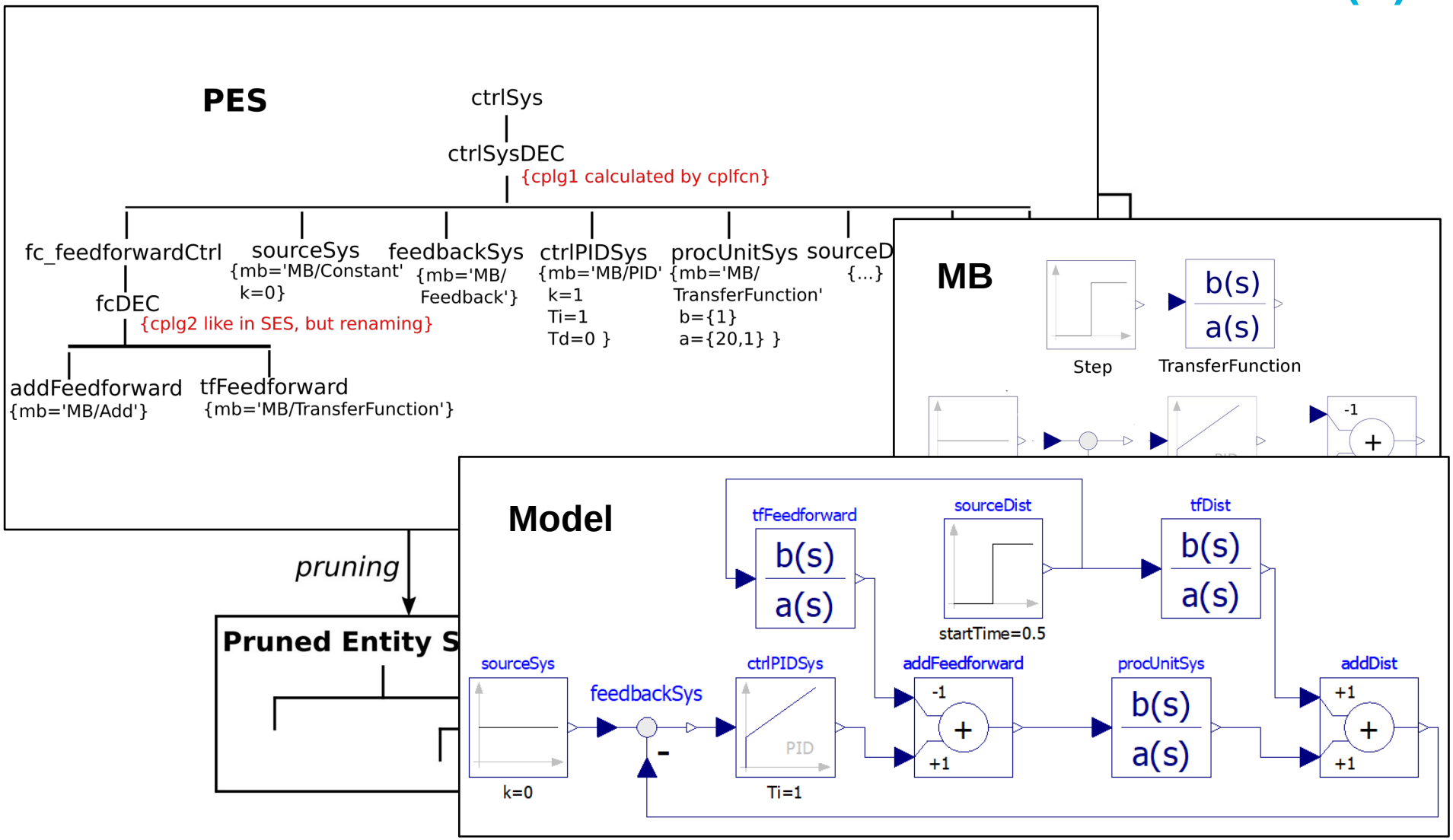
Model Selection and Generation of Variant #2 (2)



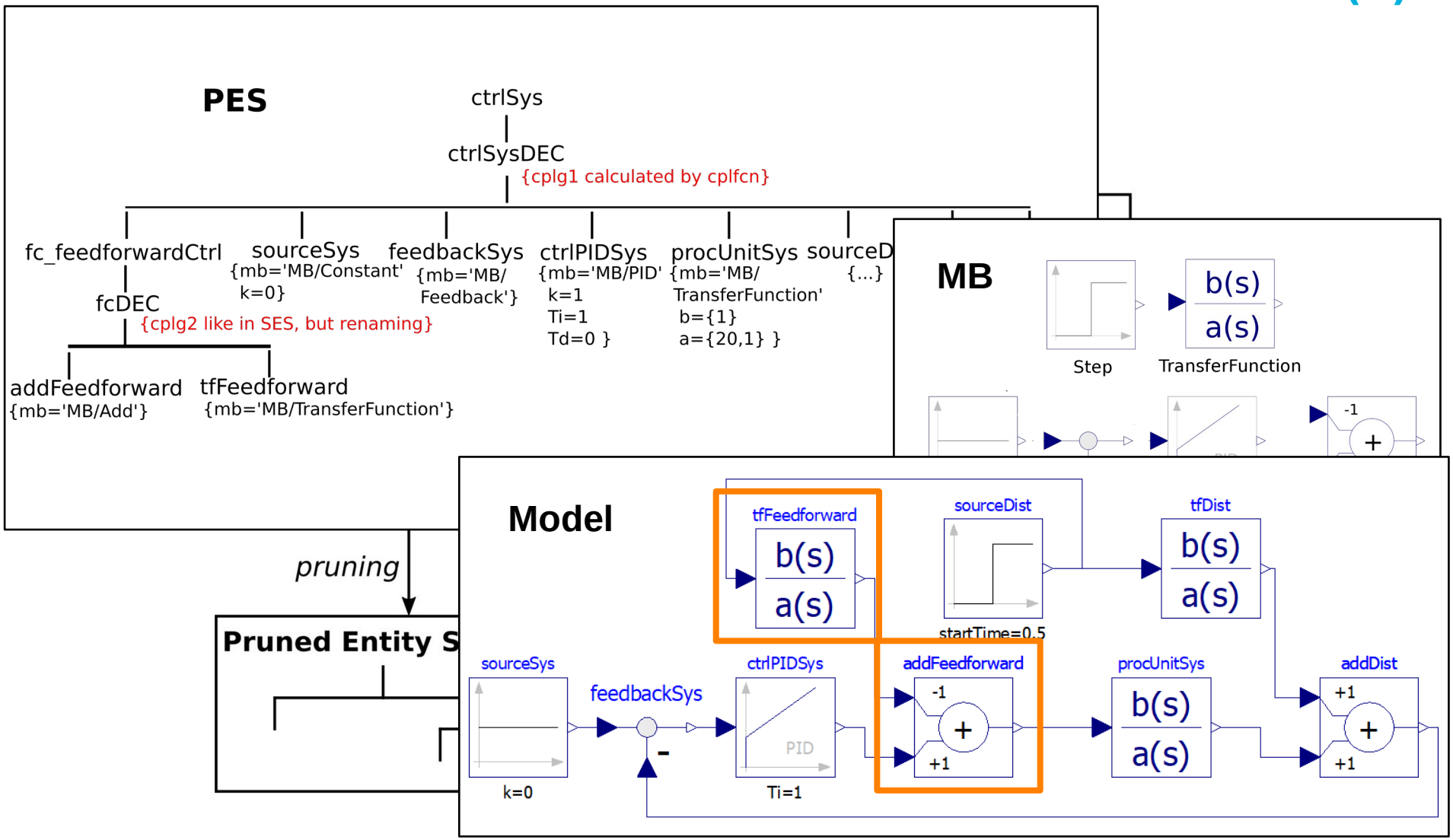
Model Selection and Generation of Variant #2 (3)



Model Selection and Generation of Variant #2 (4)



Model Selection and Generation of Variant #2 (4)



Python Toolset

- Available: `https://github.com/hendrikfolkerts`
- Tools
  - SESToPy → SES editor and IDE
  - SESViewEl → SES tree viewer
  - **SESMoPy** → Model builder → **OpenModelica**, **Dymola**, Simulink
  - SESEuPy
  - SESEcPy

Demonstration of SESMoPy (case study)

- Show provisional Experimental Frame from SESMoPy examples → `Template_for_SESMoPy`
- Show that different simulators can be set → here `OpenModelica`
- Show that two interfaces can be set → here `native`
- Merge Feedback SES from SESToPy examples to simModel → rename `simModel` to `ctrlSys` for merging.
- Show that configurations can be set in `expMethod`
- Prune for `feedforward=0` → dynamic couplings to static couplings
- Flatten for `feedforward=0` and save the FPES as file → explanation flattening: remove inner, coupled components → root node and leaves stay in tree → couplings recalculated
- (Prune for `feedforward=1` to show)
- Show the OpenModelica MB `MB.mo` and copy it in the same directory to the FPES file
- Open SESMoPy GUI → set FPES → create model → models for both configurations created
- Open one created model in OpenModelica and load MB file
- In OpenModelica open the model by double clicking
- Execute simulation → set simulation time to 50 seconds
  → Signals of interest: `sourceSys.y sourceDist.y addDist.y`
  → If the signals do not show up in plot: Click `Auto Scale` and `Fit in View` in plot

- If design objectives are not met with this structure and parameterization → later how to simulate automatically to find fitting structure and parameterization

Outline

1. Introduction
2. The case study
3. Basics of SES/MB based modeling
4. Practical modeling: implementation of an SES
5. Model selection and model generation
6. **The free Matlab SES toolbox** (C. Deatcu)
7. Organization of a simulator-independent MB
8. Full automation of simulation experiments
9. Conclusion

The Free Matlab SES Toolbox

- Available: `https://github.com/cea-wismar`

- **SES Tbx Matlab (available as App)**
  - → SES editor
  - → Function suite to work on SES (merge, check, prune, flatten,…)
  - → Model builder

- **SESViewEl** → SES tree viewer can be connected

The Free Matlab SES Toolbox – Model Builder

- To build & simulate models

  → is rather a build AND execution unit.

- Works with: - **Simulink** (internal calls or via script),
  - OpenModelica,
  - Dymola,
  - (soon MatlabDEVS).

- No graphical interface.

Demonstration of the SES Tbx Matlab with SESViewEl (case study)

- Basic functionality like in SESToPy.
- Two main differences compared to SESToPy:
  - SES wide settings right, node specific settings left side
  - Every change needs to be confirmed. ✓
- Load `Feedback.mat` from `SES-Tbx-root/more_examples/FeedbackControl`
- Show SES in SESViewEl.
- Pruning for `feedforward=0` and `feedforward=1`.
- Model building would work, but there is no accessible output.

Demonstration of the SES Tbx Matlab Model Generation (case study)

- For Simulink `Out` nodes are necessary:
  - add manually after model generation or
  - **add to the SES** before.
- Load adapted SES `Feedback_with_outputs.mat`
- Show the Simulink MB.
- Demonstration for Simulink with a function
  - Write a script or function where options for model and simulation are set (`build_simulate_feedback_with_outputs.m`).
  - File defines options for and then calls the model builder (`modelBuilder.m`).
  - Execute with `feedforward=1`, show model and output.

One of the challenges to keep the SES independent from the simulator!

The free Matlab SES Tbx – Conclusion

- Provides comprehensive functionality to design and manipulate SES.
- Model building works for Simulink, OpenModelica, Dymola, and (soon) MatlabDEVS.
- Model builder builds and executes model.
- Some challenges to keep the SES independent from the simulator.
- Automation of simulation experiments is possible!

Outline

1. Introduction
2. The case study
3. Basics of SES/MB based modeling
4. Practical modeling: implementation of an SES
5. Model selection and model generation
6. The free Matlab SES toolbox
7. **Organization of a simulator-independent MB**
   (H. Folkerts)
8. Full automation of simulation experiments
9. Conclusion

Model Building – Support Different Simulators Facts / Challenges

Model Building – Support Different Simulators Facts / Challenges

- System models should be tested with different simulators
  - Simulators are domain specific
  - Simulator correctness

Model Building – Support Different Simulators Facts / Challenges

- System models should be tested with different simulators
  - Simulators are domain specific
  - Simulator correctness
- **SES is independent of simulator**

Model Building – Support Different Simulators Facts / Challenges

- System models should be tested with different simulators
  - Simulators are domain specific
  - Simulator correctness
- **SES is independent of simulator**
- Native model building needs **one** MB for **each** simulator
  → Error prone and costly to maintain

Model Building – Support Different Simulators Facts / Challenges

- System models should be tested with different simulators
  - Simulators are domain specific
  - Simulator correctness
- **SES is independent of simulator**
- Native model building needs **one** MB for **each** simulator
  → Error prone and costly to maintain
- Native model building needs **specific model builders**, because simulators use:
  - Different portnames / porttypes
  - Different (block) parameters / parameter names
  - Different parameter meaning
  - Different syntax

Model Building – Support Different Simulators Facts / Challenges

- System models should be tested with different simulators
  - Simulators are domain specific
  - Simulator correctness
- **SES is independent of simulator**
- Native model building needs **one** MB for **each** simulator
  → Error prone and costly to maintain
- Native model building needs **specific model builders**, because simulators use:
  - Different portnames / porttypes
  - Different (block) parameters / parameter names
  - Different parameter meaning
  - Different syntax
- **Goal: One** (simple) MB and model builder for **all** simulators

Functional Mock-up Interface (FMI) [1,2]

[1]Blochwitz et al. (2011) „The Functional Mockup Interface for Tool independent Exchange of Simulation Models“. Proc. of the 8th Modelica Conference, Dresden.

[2]Blochwitz et al. (2012) „Functional Mockup Interface 2.0: The Standard for Tool independent Exchange of Simulation Models“. Proc. of the 9th Modelica Conference, Munich.

Functional Mock-up Interface (FMI) [1,2]

- FMI defines a standardized interface of components (models, blocks)

[1]Blochwitz et al. (2011) „The Functional Mockup Interface for Tool independent Exchange of Simulation Models“. Proc. of the 8th Modelica Conference, Dresden.

[2]Blochwitz et al. (2012) „Functional Mockup Interface 2.0: The Standard for Tool independent Exchange of Simulation Models“. Proc. of the 9th Modelica Conference, Munich.

Functional Mock-up Interface (FMI) [1,2]

- FMI defines a standardized interface of components (models, blocks)
  - Interface is based on C code or binaries (know-how protection)

[1]Blochwitz et al. (2011) „The Functional Mockup Interface for Tool independent Exchange of Simulation Models“. Proc. of the 8th Modelica Conference, Dresden.

[2]Blochwitz et al. (2012) „Functional Mockup Interface 2.0: The Standard for Tool independent Exchange of Simulation Models“. Proc. of the 9th Modelica Conference, Munich.

Functional Mock-up Interface (FMI) [1,2]

- FMI defines a standardized interface of components (models, blocks)
  - Interface is based on C code or binaries (know-how protection)
  - Many simulators support FMI

[1]Blochwitz et al. (2011) „The Functional Mockup Interface for Tool independent Exchange of Simulation Models“. Proc. of the 8th Modelica Conference, Dresden.

[2]Blochwitz et al. (2012) „Functional Mockup Interface 2.0: The Standard for Tool independent Exchange of Simulation Models“. Proc. of the 9th Modelica Conference, Munich.

Functional Mock-up Interface (FMI) [1,2]

- FMI defines a standardized interface of components (models, blocks)
  - Interface is based on C code or binaries (know-how protection)
  - Many simulators support FMI
  - Reuse of components
    - (i) For **model exchange**
    - (ii) For co-simulation

[1]Blochwitz et al. (2011) „The Functional Mockup Interface for Tool independent Exchange of Simulation Models“. Proc. of the 8th Modelica Conference, Dresden.

[2]Blochwitz et al. (2012) „Functional Mockup Interface 2.0: The Standard for Tool independent Exchange of Simulation Models“. Proc. of the 9th Modelica Conference, Munich.

Functional Mock-up Interface (FMI) [1,2]

- FMI defines a standardized interface of components (models, blocks)
  - Interface is based on C code or binaries (know-how protection)
  - Many simulators support FMI
  - Reuse of components
    - (i) For **model exchange**
    - (ii) For co-simulation
- **FMI for model exchange** defines an interface for the simulation environment to generate C code of a dynamic system model (still problems for discrete event models)

[1]Blochwitz et al. (2011) „The Functional Mockup Interface for Tool independent Exchange of Simulation Models“. Proc. of the 8th Modelica Conference, Dresden.

[2]Blochwitz et al. (2012) „Functional Mockup Interface 2.0: The Standard for Tool independent Exchange of Simulation Models“. Proc. of the 9th Modelica Conference, Munich.

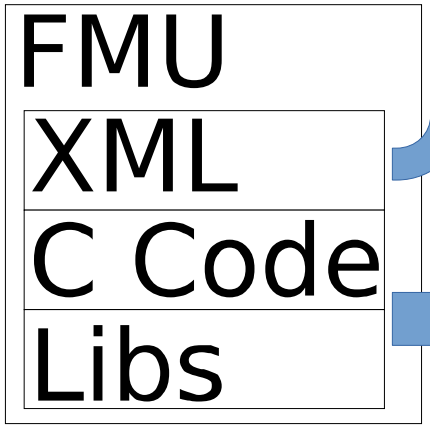
Functional Mock-up Unit (FMU)

Functional Mock-up Unit (FMU)

- Block = Functional Mock-up Unit (FMU) → zipped file, fileextension .fmu
- FMU implements the FMI

Functional Mock-up Unit (FMU)

- Block = Functional Mock-up Unit (FMU) → zipped file, fileextension .fmu
- FMU implements the FMI

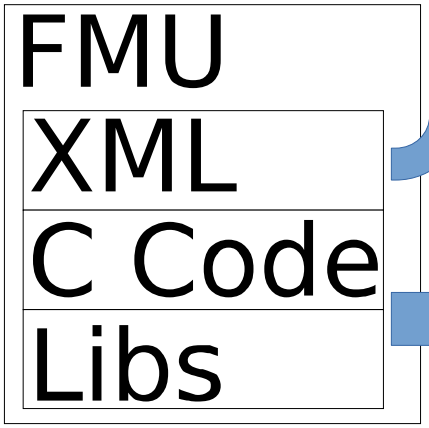


Description of interface data → definition of all exposed variables, static information

Functionality in C source or binary form (for different platforms), C interface to be used by the simulation environment

Functional Mock-up Unit (FMU)

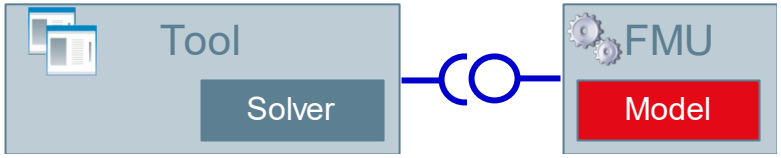
- Block = Functional Mock-up Unit (FMU) → zipped file, fileextension .fmu
- FMU implements the FMI



Description of interface data → definition of all exposed variables, static information

Functionality in C source or binary form (for different platforms), C interface to be used by the simulation environment

Model Exchange



Some information taken from the FMI presentation on the website www.fmi-standard.org.

Model Building – Support Different Simulators

Model Building – Support Different Simulators

- **Using OpenModelica and FMI for model exchange (export whole model as FMU)**

Model Building – Support Different Simulators

- **Using OpenModelica and FMI for model exchange (export whole model as FMU)**
  - Define an MB for OpenModelica and/or export FMUs from any simulator, build and configure OpenModelica model, finally **export the whole model as an FMU**

Model Building – Support Different Simulators

- **Using OpenModelica and FMI for model exchange (export whole model as FMU)**
  - Define an MB for OpenModelica and/or export FMUs from any simulator, build and configure OpenModelica model, finally **export the whole model as an FMU**
  - Import model FMU in the target simulator

Model Building – Support Different Simulators

- **Using OpenModelica and FMI for model exchange (export whole model as FMU)**
  - Define an MB for OpenModelica and/or export FMUs from any simulator, build and configure OpenModelica model, finally **export the whole model as an FMU**
  - Import model FMU in the target simulator

→ **One MB for all simulators**

Model Building – Support Different Simulators

- **Using OpenModelica and FMI for model exchange (export whole model as FMU)**
  - Define an MB for OpenModelica and/or export FMUs from any simulator, build and configure OpenModelica model, finally **export the whole model as an FMU**
  - Import model FMU in the target simulator

→ **One MB for all simulators**

→ No possibility to change structure of model built as FMU, only parameters

Model Building – Support Different Simulators

- **Using OpenModelica and FMI for model exchange (export whole model as FMU)**
  - Define an MB for OpenModelica and/or export FMUs from any simulator, build and configure OpenModelica model, finally **export the whole model as an FMU**
  - Import model FMU in the target simulator

→ **One MB for all simulators**

→ No possibility to change structure of model built as FMU, only parameters

Current development state

Model Building – Support Different Simulators

- **Using OpenModelica and FMI for model exchange (export whole model as FMU)**
  - Define an MB for OpenModelica and/or export FMUs from any simulator, build and configure OpenModelica model, finally **export the whole model as an FMU**
  - Import model FMU in the target simulator

→ **One MB for all simulators**

→ No possibility to change structure of model built as FMU, only parameters

Current development state

- **Using OpenModelica and FMI for model exchange (export block FMUs)**

Model Building – Support Different Simulators

- **Using OpenModelica and FMI for model exchange (export whole model as FMU)**
  - Define an MB for OpenModelica and/or export FMUs from any simulator, build and configure OpenModelica model, finally **export the whole model as an FMU**
  - Import model FMU in the target simulator

→ **One MB for all simulators**

→ No possibility to change structure of model built as FMU, only parameters

Current development state

- **Using OpenModelica and FMI for model exchange (export block FMUs)**
  - Define an MB for OpenModelica and/or export FMUs from any simulator, import and configure blocks in OpenModelica, **export each configured block as FMU**

Model Building – Support Different Simulators

- **Using OpenModelica and FMI for model exchange (export whole model as FMU)**
  - Define an MB for OpenModelica and/or export FMUs from any simulator, build and configure OpenModelica model, finally **export the whole model as an FMU**
  - Import model FMU in the target simulator

→ **One MB for all simulators**

→ No possibility to change structure of model built as FMU, only parameters

Current development state

- **Using OpenModelica and FMI for model exchange (export block FMUs)**
  - Define an MB for OpenModelica and/or export FMUs from any simulator, import and configure blocks in OpenModelica, **export each configured block as FMU**
  - Import and connect preconfigured FMUs in the target simulator

Model Building – Support Different Simulators

- **Using OpenModelica and FMI for model exchange (export whole model as FMU)**
  - Define an MB for OpenModelica and/or export FMUs from any simulator, build and configure OpenModelica model, finally **export the whole model as an FMU**
  - Import model FMU in the target simulator

→ **One MB for all simulators**

→ No possibility to change structure of model built as FMU, only parameters

Current development state

- **Using OpenModelica and FMI for model exchange (export block FMUs)**
  - Define an MB for OpenModelica and/or export FMUs from any simulator, import and configure blocks in OpenModelica, **export each configured block as FMU**
  - Import and connect preconfigured FMUs in the target simulator

→ One MB for all simulators

Model Building – Support Different Simulators

- **Using OpenModelica and FMI for model exchange (export whole model as FMU)**
  - Define an MB for OpenModelica and/or export FMUs from any simulator, build and configure OpenModelica model, finally **export the whole model as an FMU**
  - Import model FMU in the target simulator

→ **One MB for all simulators**

→ No possibility to change structure of model built as FMU, only parameters

Current development state

- **Using OpenModelica and FMI for model exchange (export block FMUs)**
  - Define an MB for OpenModelica and/or export FMUs from any simulator, import and configure blocks in OpenModelica, **export each configured block as FMU**
  - Import and connect preconfigured FMUs in the target simulator

→ One MB for all simulators

→ Possibilty to change structure of the model

Model Building – Support Different Simulators

- **Using OpenModelica and FMI for model exchange (export whole model as FMU)**
  - Define an MB for OpenModelica and/or export FMUs from any simulator, build and configure OpenModelica model, finally **export the whole model as an FMU**
  - Import model FMU in the target simulator

→ **One MB for all simulators**

→ No possibility to change structure of model built as FMU, only parameters

Current development state

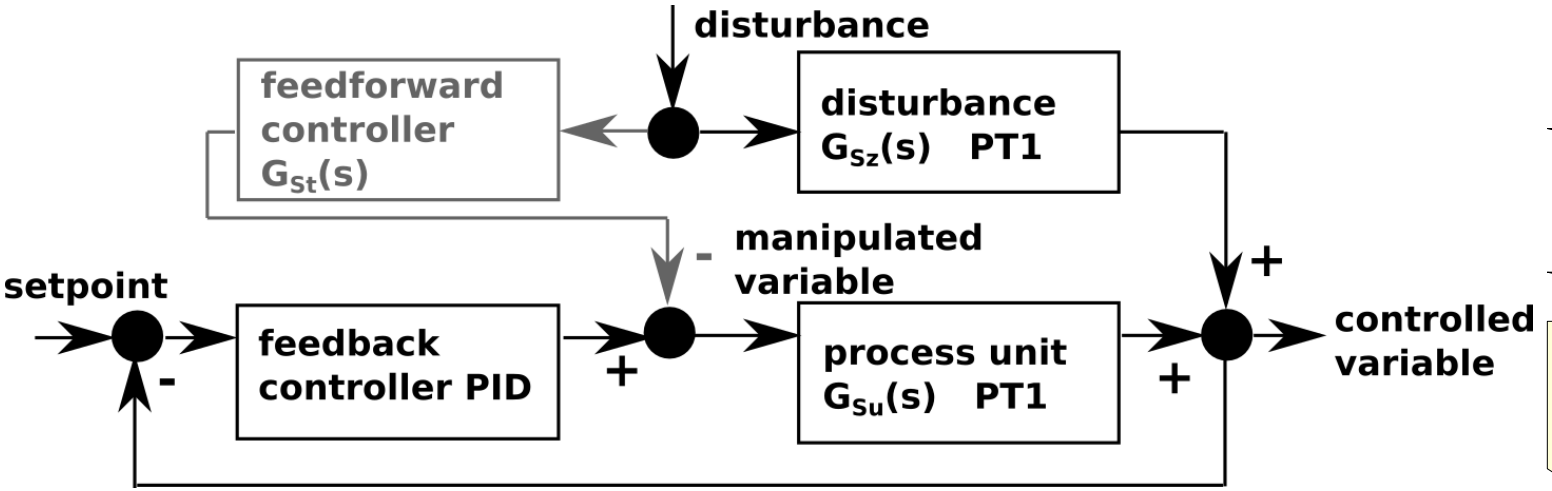
- **Using OpenModelica and FMI for model exchange (export block FMUs)**
  - Define an MB for OpenModelica and/or export FMUs from any simulator, import and configure blocks in OpenModelica, **export each configured block as FMU**
  - Import and connect preconfigured FMUs in the target simulator

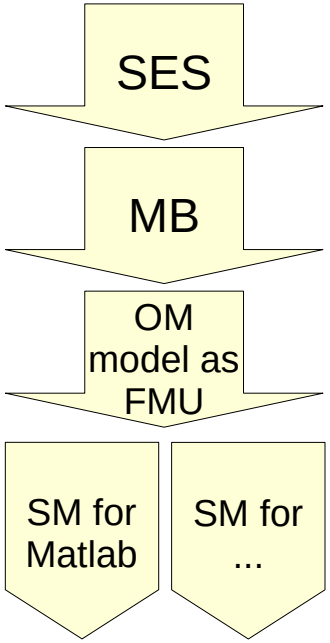
→ One MB for all simulators

→ Possibilty to change structure of the model

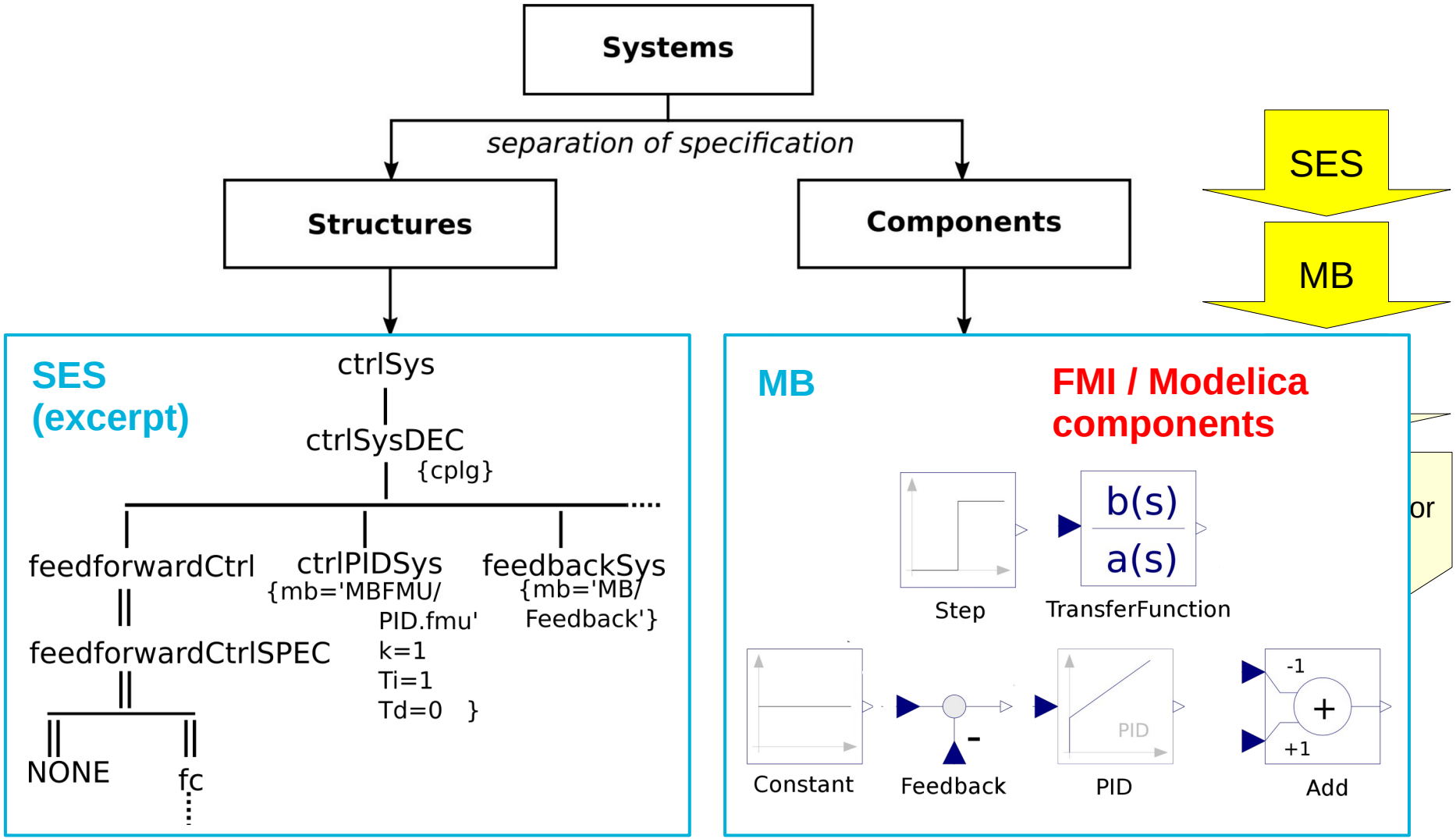
→ Model building VERY slow → parallel processing?

Case Study with FMI

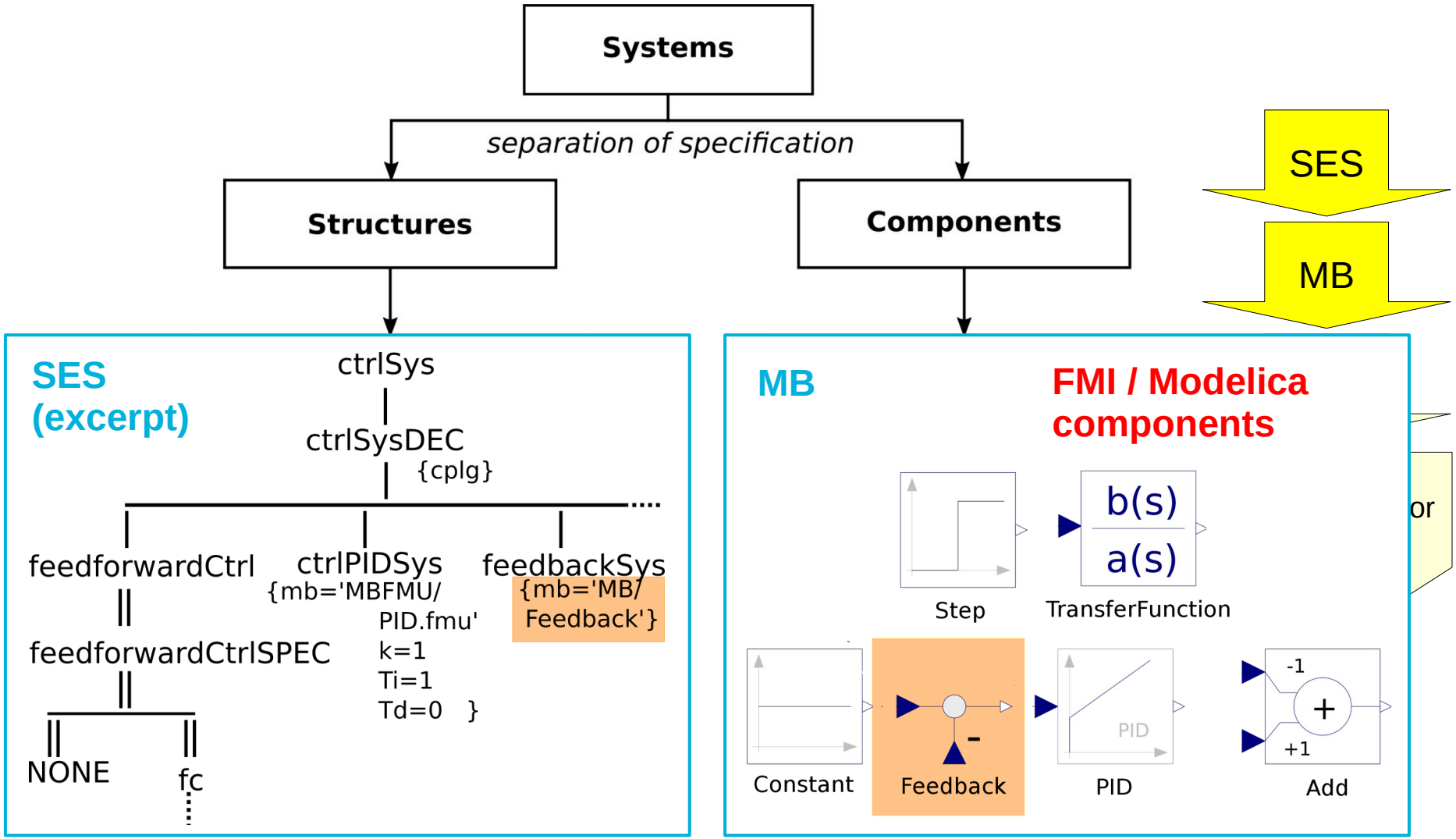




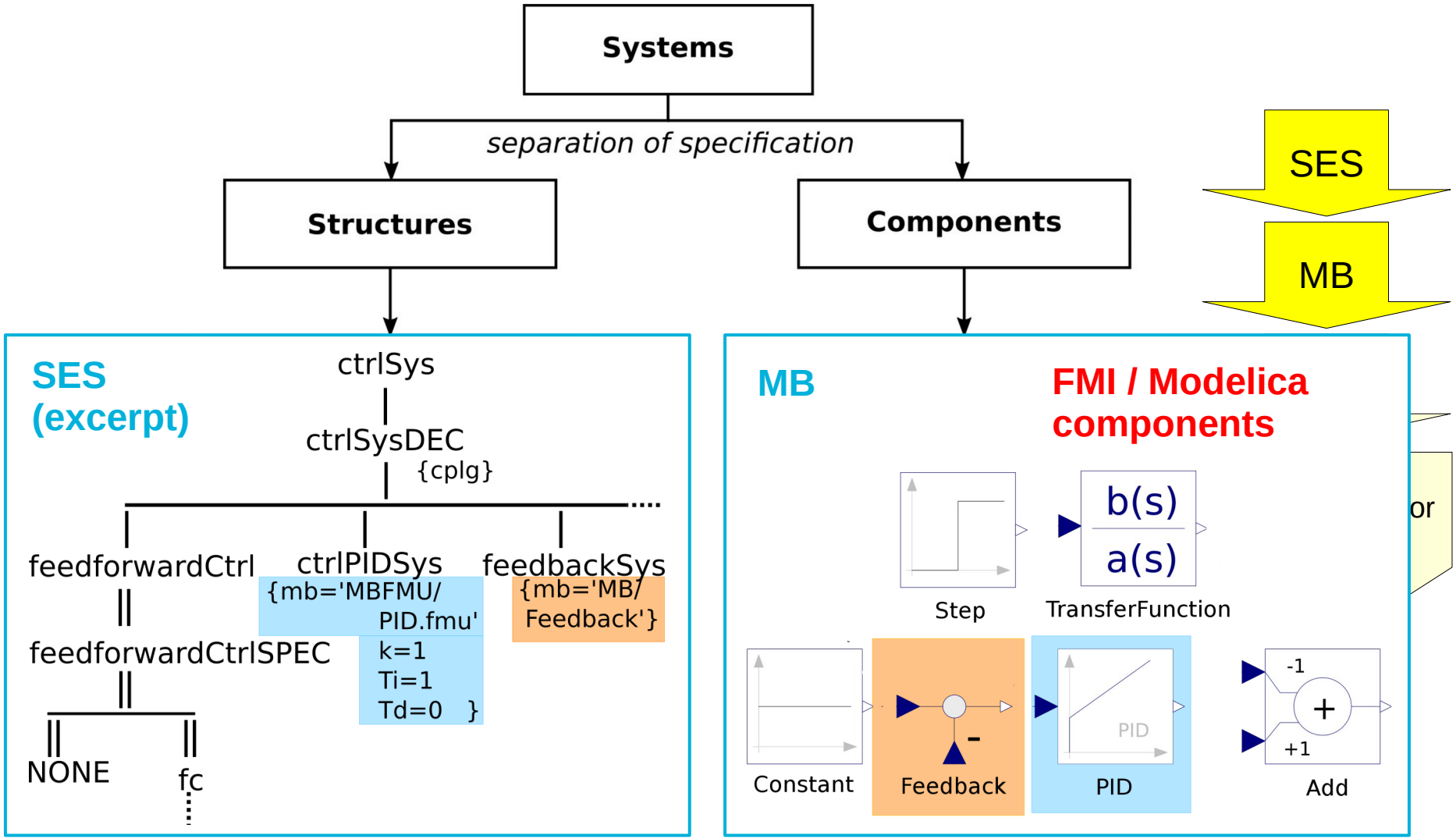
SES/MB Approach with FMI Components



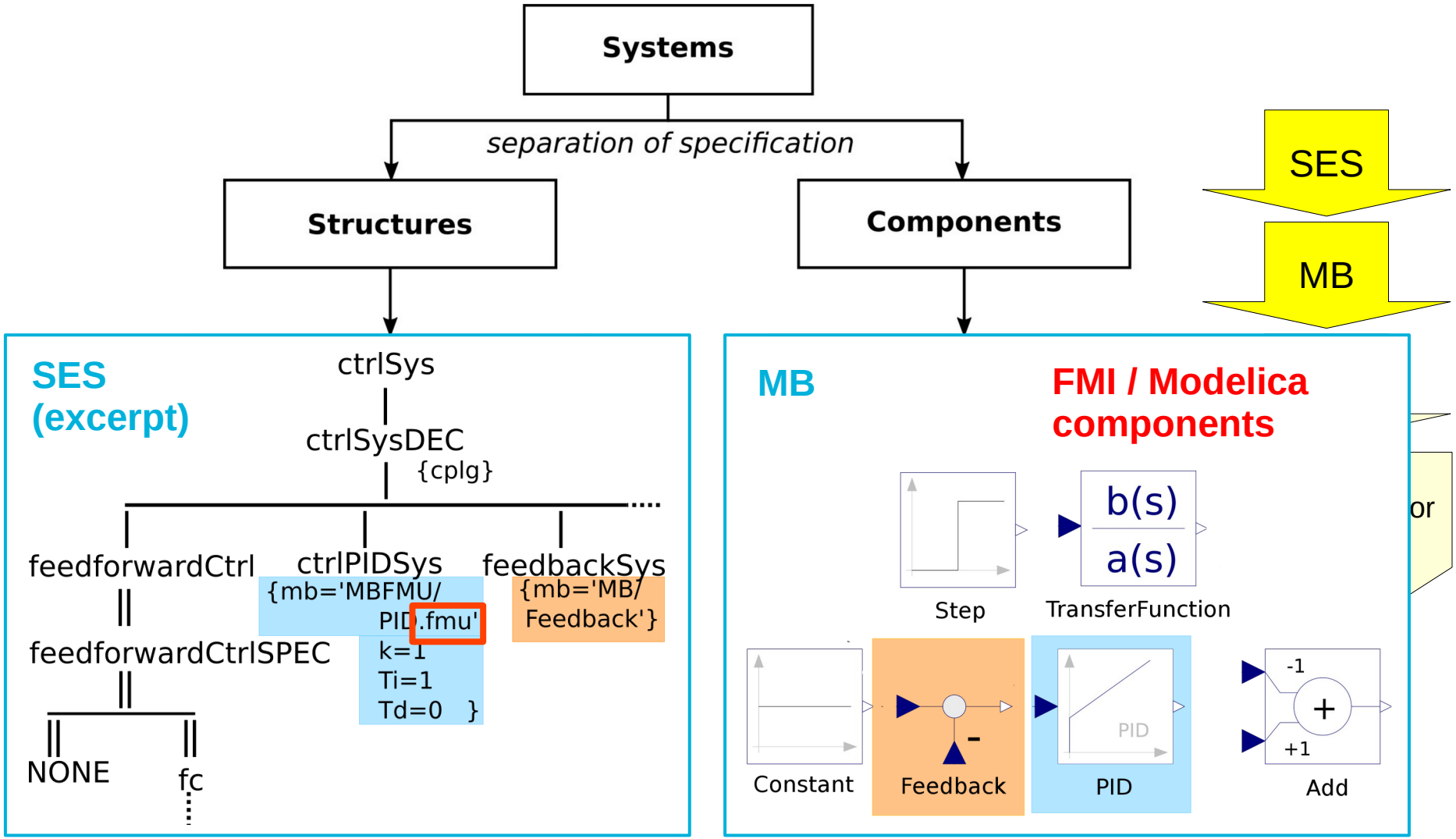
SES/MB Approach with FMI Components



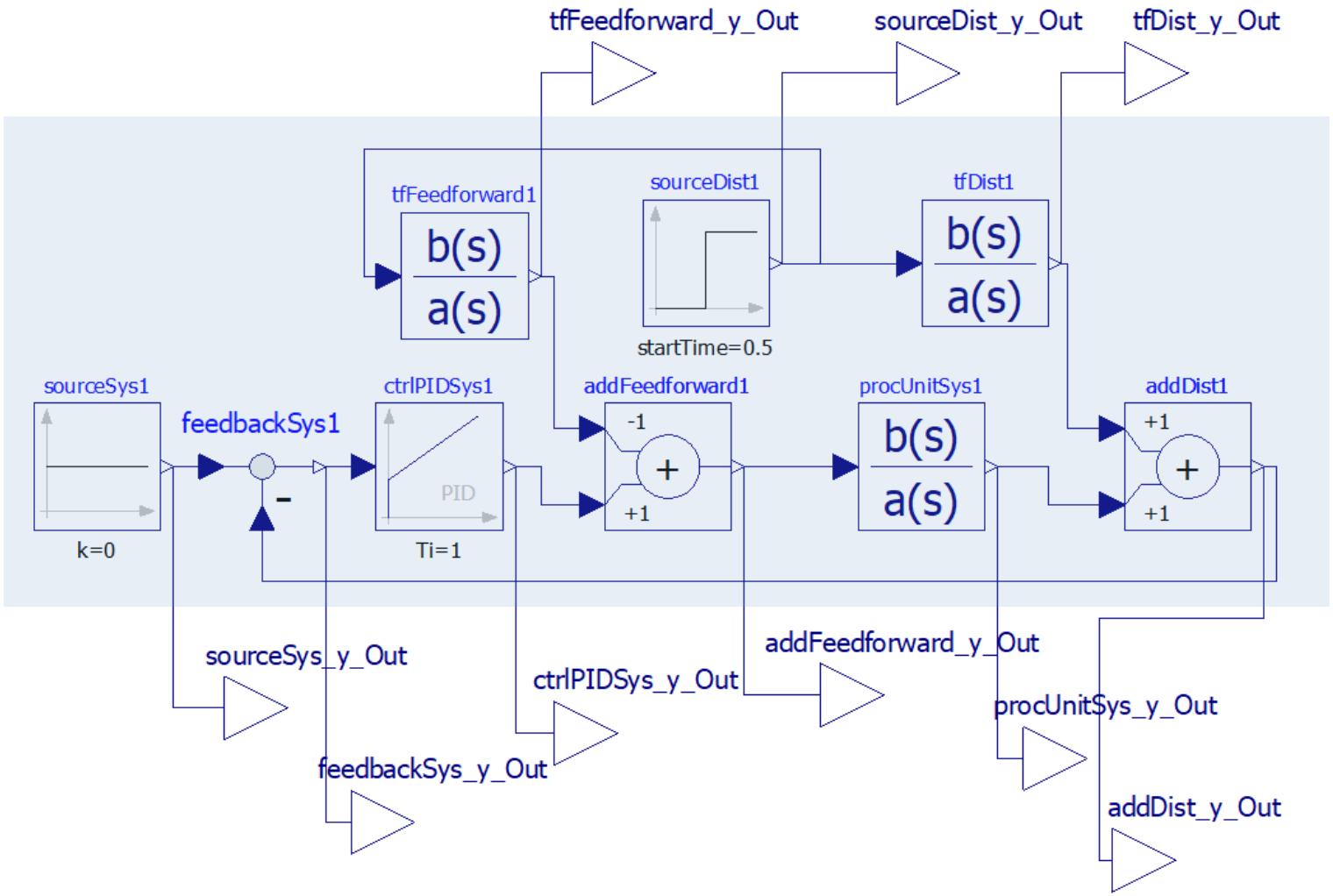
SES/MB Approach with FMI Components

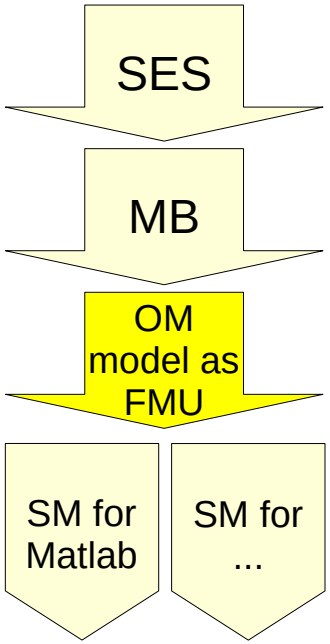


SES/MB Approach with FMI Components

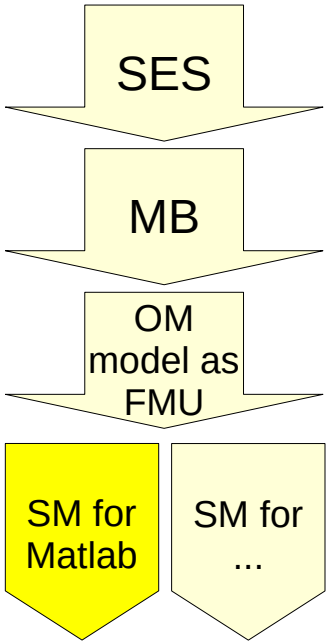


Generated & Configured OpenModelica Model / FMU



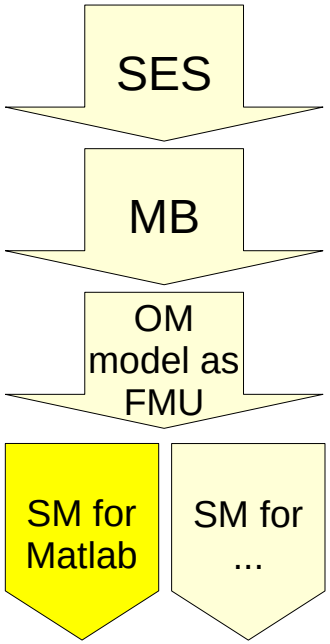


Case Study: FMU Imported in Matlab/Simulink



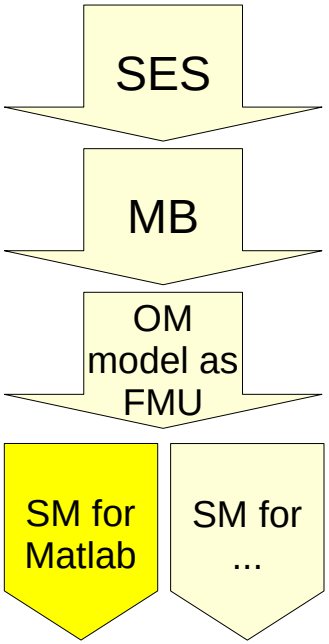
Case Study: FMU Imported in Matlab/Simulink

- **Why do we need ports in the model FMU?** → Simulink only returns variables connected to an Simulink “Out” block → “Out” blocks added in Modelica before exporting the FMU, which in turn can be connected to Simulink “Out” blocks



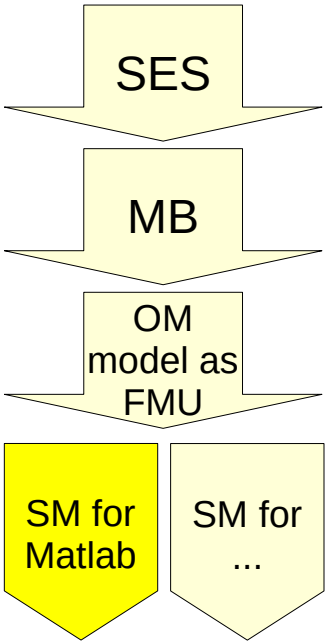
Case Study: FMU Imported in Matlab/Simulink

- **Why do we need ports in the model FMU?** → Simulink only returns variables connected to an Simulink “Out” block → “Out” blocks added in Modelica before exporting the FMU, which in turn can be connected to Simulink “Out” blocks
- A .m Matlab script is created with instructions on:



Case Study: FMU Imported in Matlab/Simulink

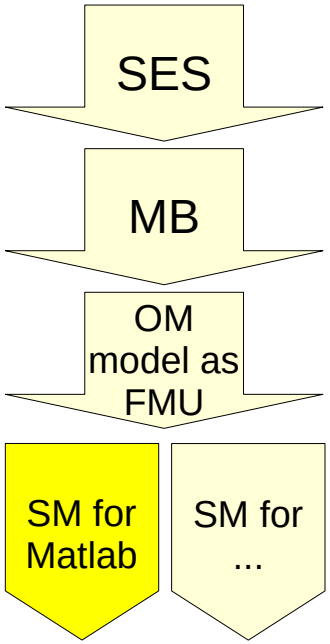
- **Why do we need ports in the model FMU?** → Simulink only returns variables connected to an Simulink “Out” block → “Out” blocks added in Modelica before exporting the FMU, which in turn can be connected to Simulink “Out” blocks
- A .m Matlab script is created with instructions on:
  - Create Simulink model



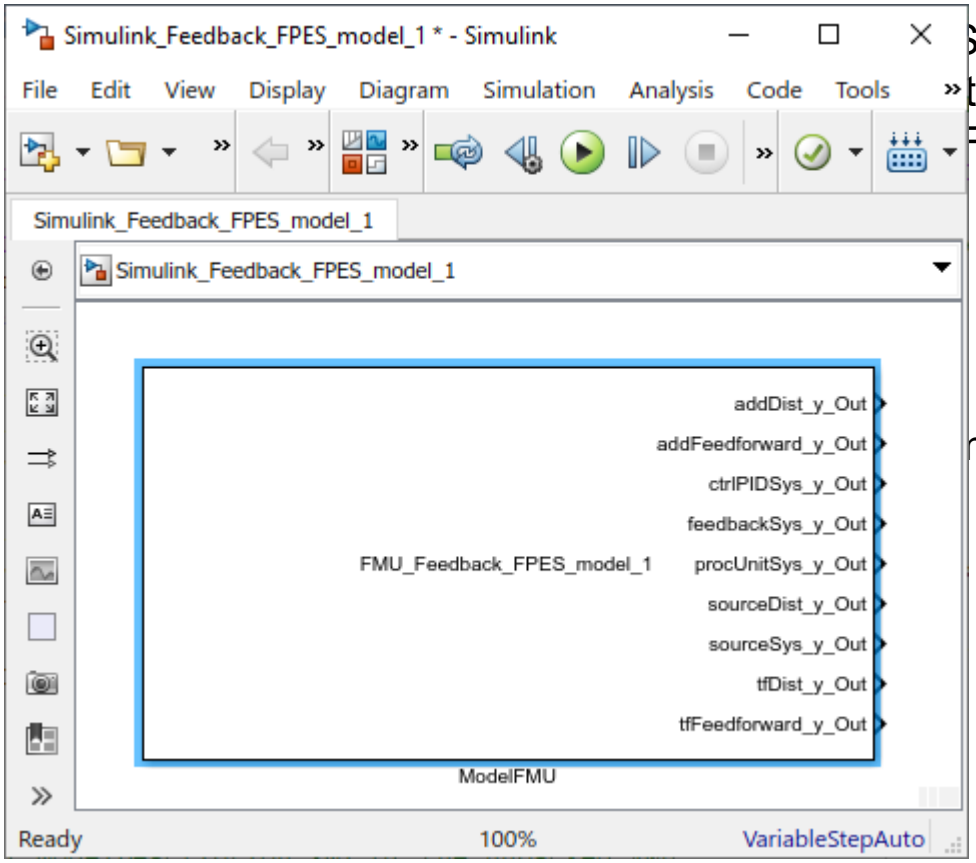
Case Study: FMU Imported in Matlab/Simulink

- **Why do we need ports in the model FMU?** → Simulink only returns variables connected to an Simulink “Out” block → “Out” blocks added in Modelica before exporting the FMU, which in turn can be connected to Simulink “Out” blocks
- A .m Matlab script is created with instructions on:
  - Create Simulink model
  - Add Simulink FMU block, configure with the model FMU

    → FMU is extracted

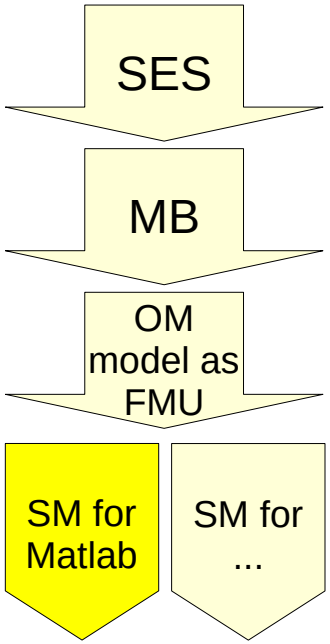


Case Study: FMU Imported in Matlab/Simulink



imulink only
t" block → "Out"
FMU, which in

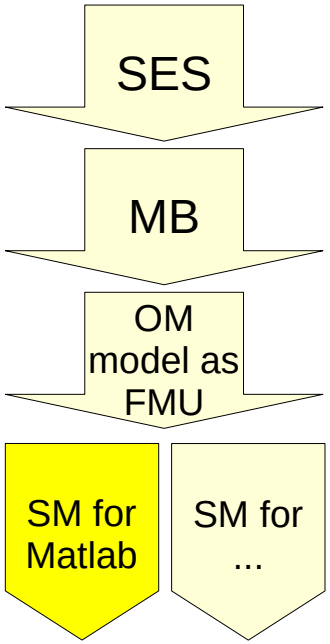
model FMU



Case Study: FMU Imported in Matlab/Simulink

- **Why do we need ports in the model FMU?** → Simulink only returns variables connected to an Simulink “Out” block → “Out” blocks added in Modelica before exporting the FMU, which in turn can be connected to Simulink “Out” blocks
- A .m Matlab script is created with instructions on:
  - Create Simulink model
  - Add Simulink FMU block, configure with the model FMU

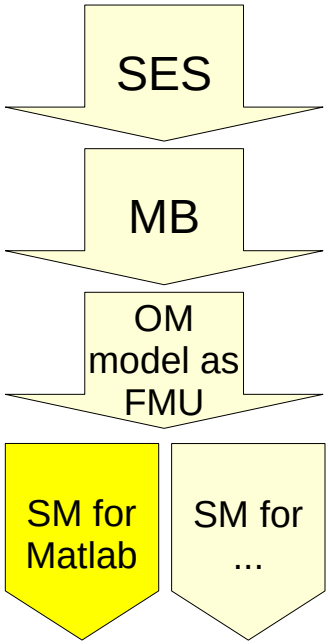
    → FMU is extracted



Case Study: FMU Imported in Matlab/Simulink

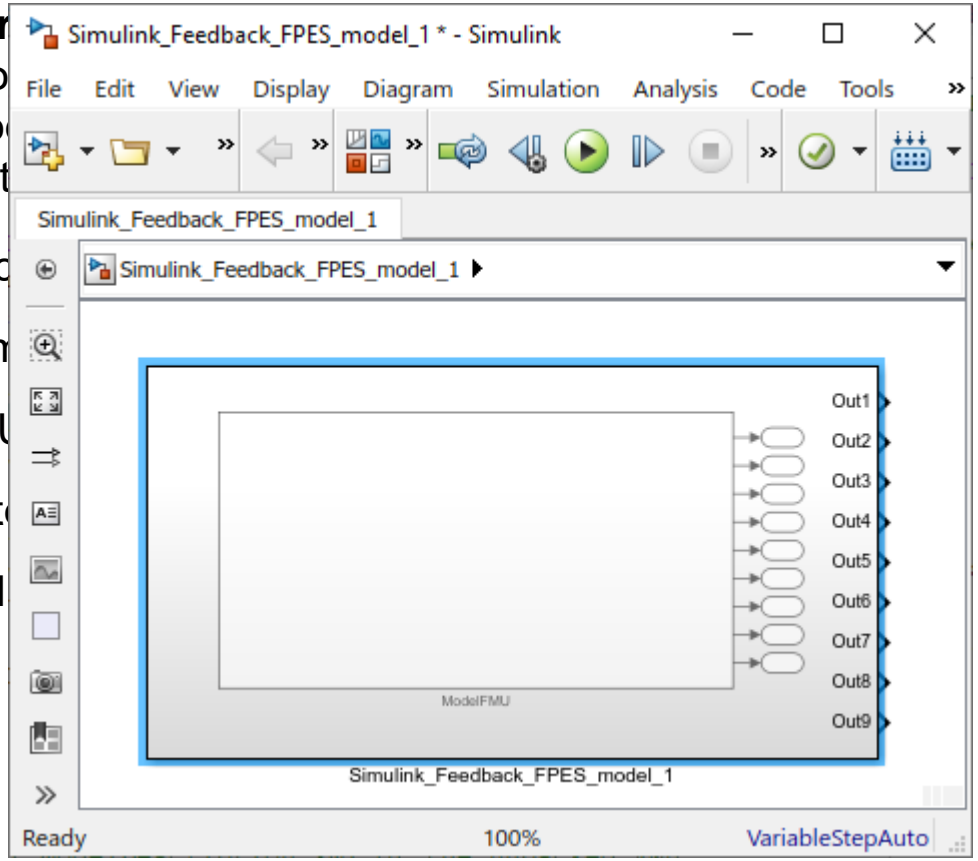
- **Why do we need ports in the model FMU?** → Simulink only returns variables connected to an Simulink “Out” block → “Out” blocks added in Modelica before exporting the FMU, which in turn can be connected to Simulink “Out” blocks
- A .m Matlab script is created with instructions on:
  - Create Simulink model
  - Add Simulink FMU block, configure with the model FMU

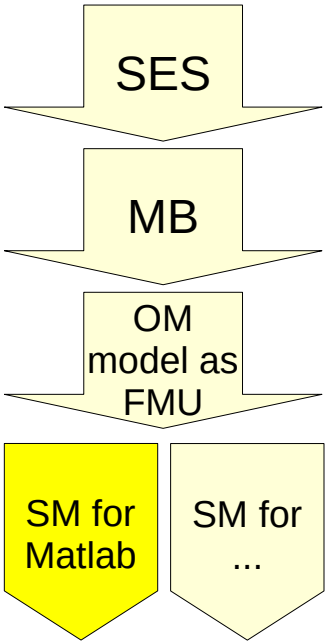
    → FMU is extracted
  - Create Submodel → Ports get names that can be searched for



Case Study: FMU Imported in Matlab/Simulink

- **Why do we need por**
  returns variables co
  blocks added in Mo
  turn can be connect
- A .m Matlab script is c
  - Create Simulink m
  - Add Simulink FMU
    → FMU is extract
  - Create Submodel
    for

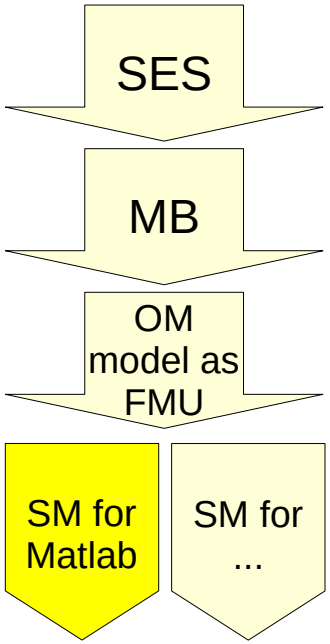




Case Study: FMU Imported in Matlab/Simulink

- **Why do we need ports in the model FMU?** → Simulink only returns variables connected to an Simulink “Out” block → “Out” blocks added in Modelica before exporting the FMU, which in turn can be connected to Simulink “Out” blocks
- A .m Matlab script is created with instructions on:
  - Create Simulink model
  - Add Simulink FMU block, configure with the model FMU

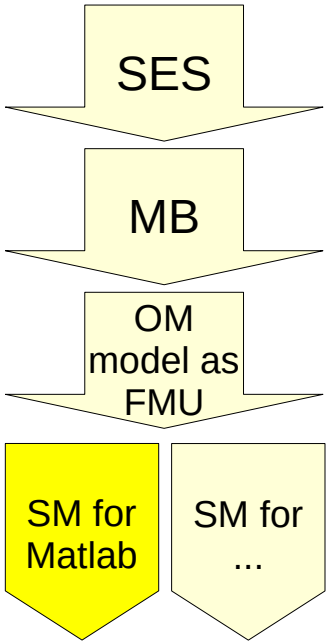
    → FMU is extracted
  - Create Submodel → Ports get names that can be searched for



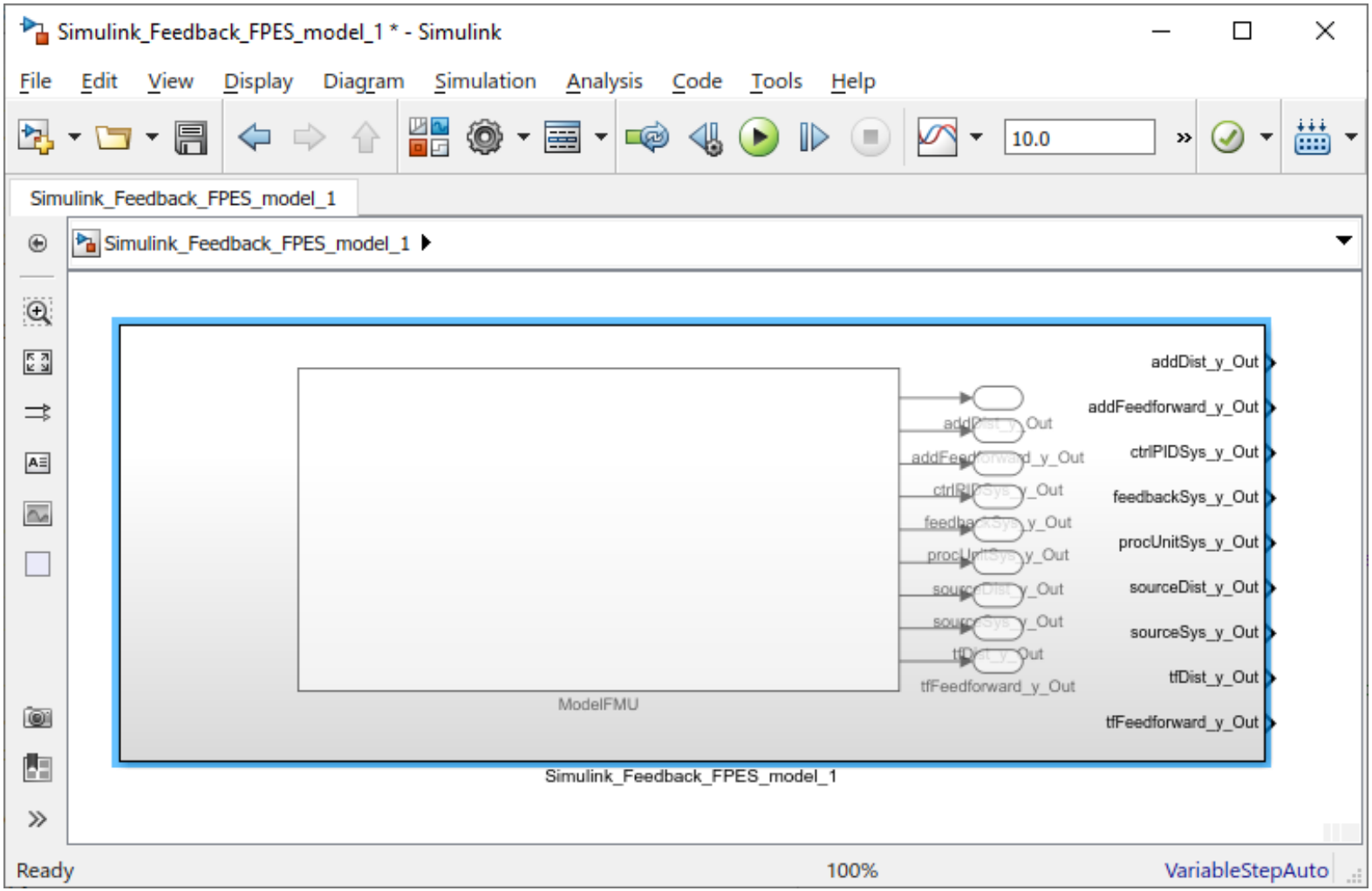
Case Study: FMU Imported in Matlab/Simulink

- **Why do we need ports in the model FMU?** → Simulink only returns variables connected to an Simulink “Out” block → “Out” blocks added in Modelica before exporting the FMU, which in turn can be connected to Simulink “Out” blocks
- A .m Matlab script is created with instructions on:
  - Create Simulink model
  - Add Simulink FMU block, configure with the model FMU

    → FMU is extracted
  - Create Submodel → Ports get names that can be searched for
  - Rename the ports → There is an “outputs” section in the modelDescription.xml of the FMU



Case Study: FMU Imported in Matlab/Simulink



SES

MB

OM model as FMU

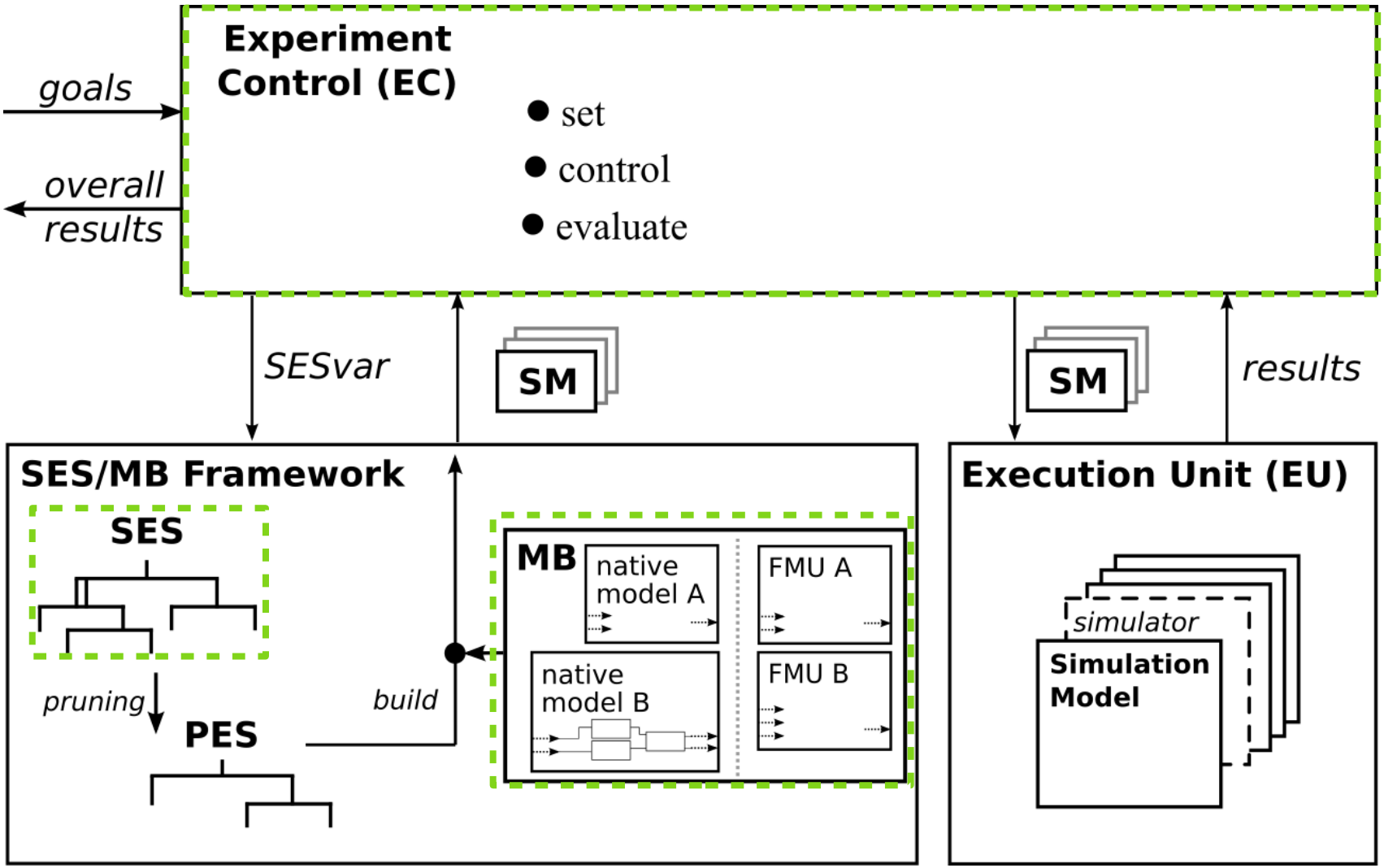
SM for Matlab

SM for ...

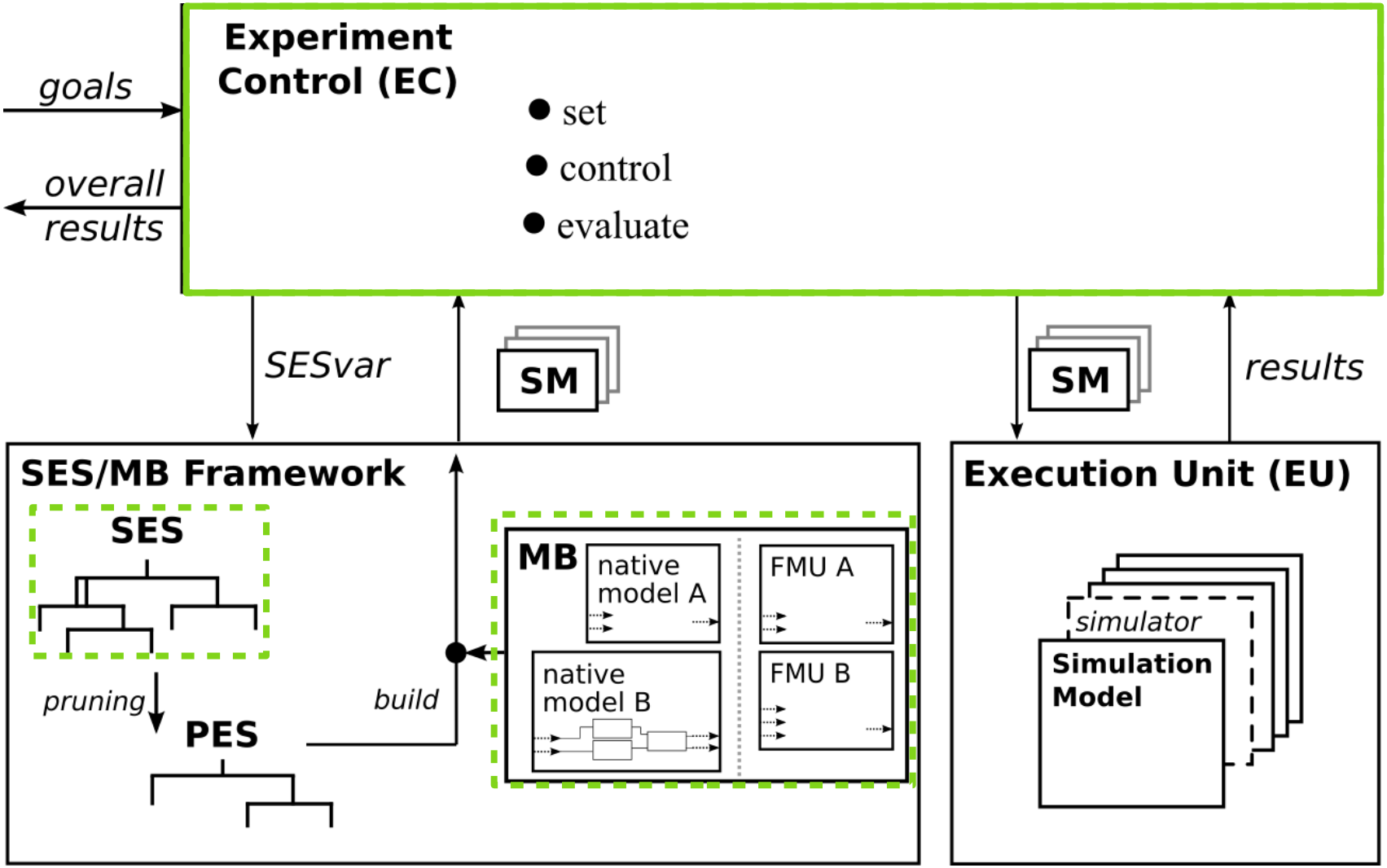
Outline

1. Introduction
2. The case study
3. Basics of SES/MB based modeling
4. Practical modeling: implementation of an SES
5. Model selection and model generation
6. The free Matlab SES toolbox
7. Organization of a simulator-independent MB
8. **Full automation of simulation experiments**
   (H. Folkerts)
9. Conclusion

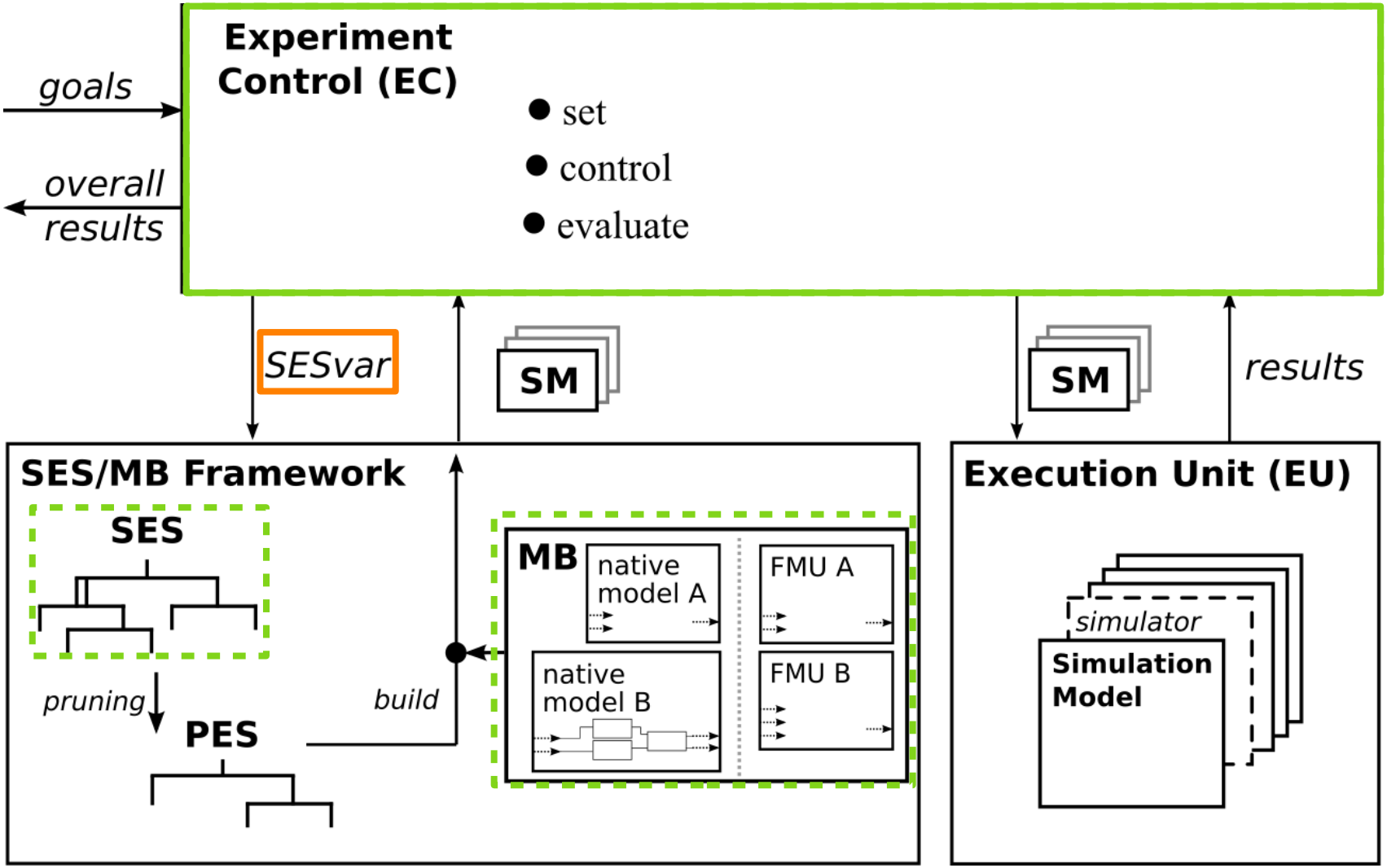
Extended SES/MB Architecture



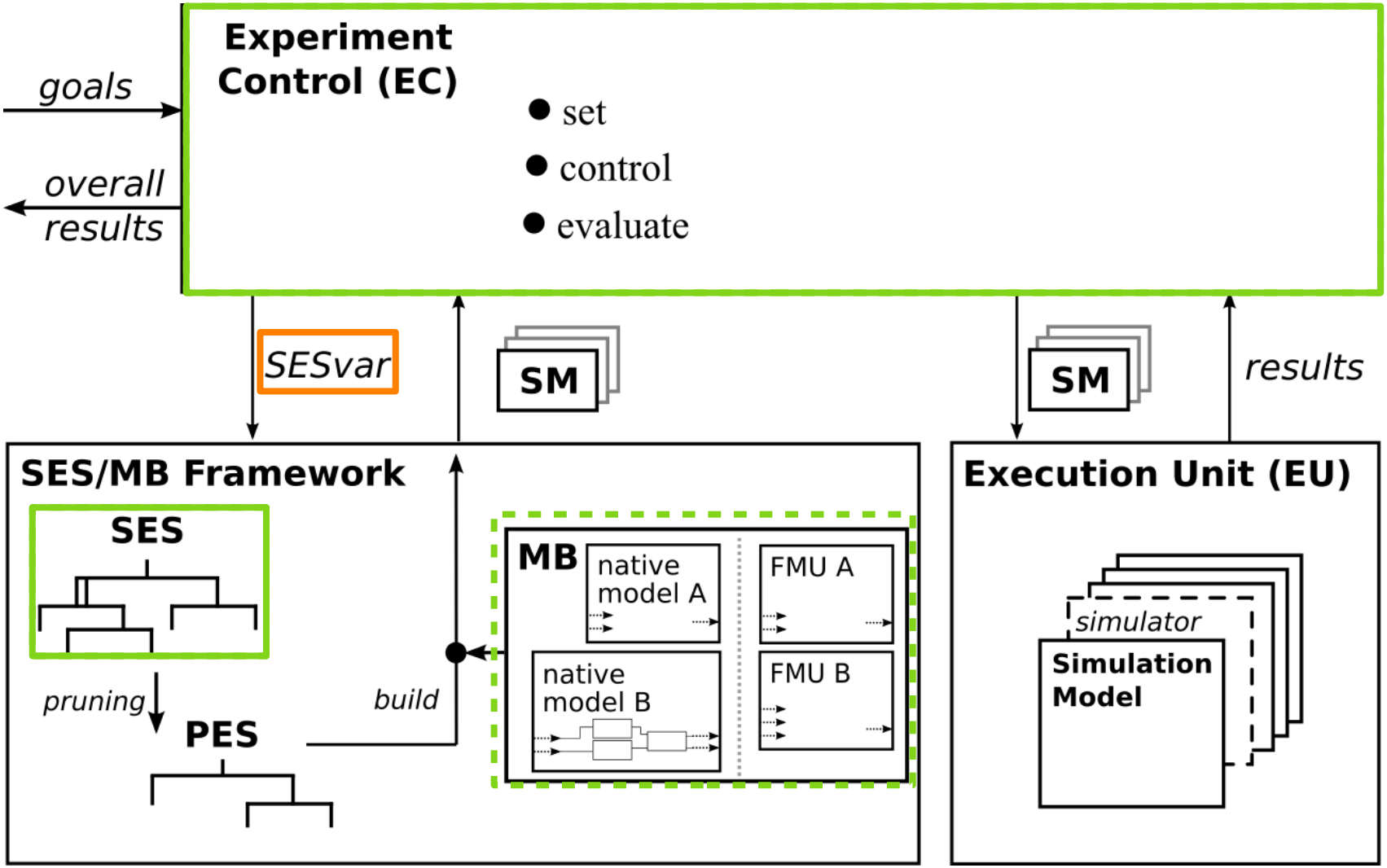
Extended SES/MB Architecture



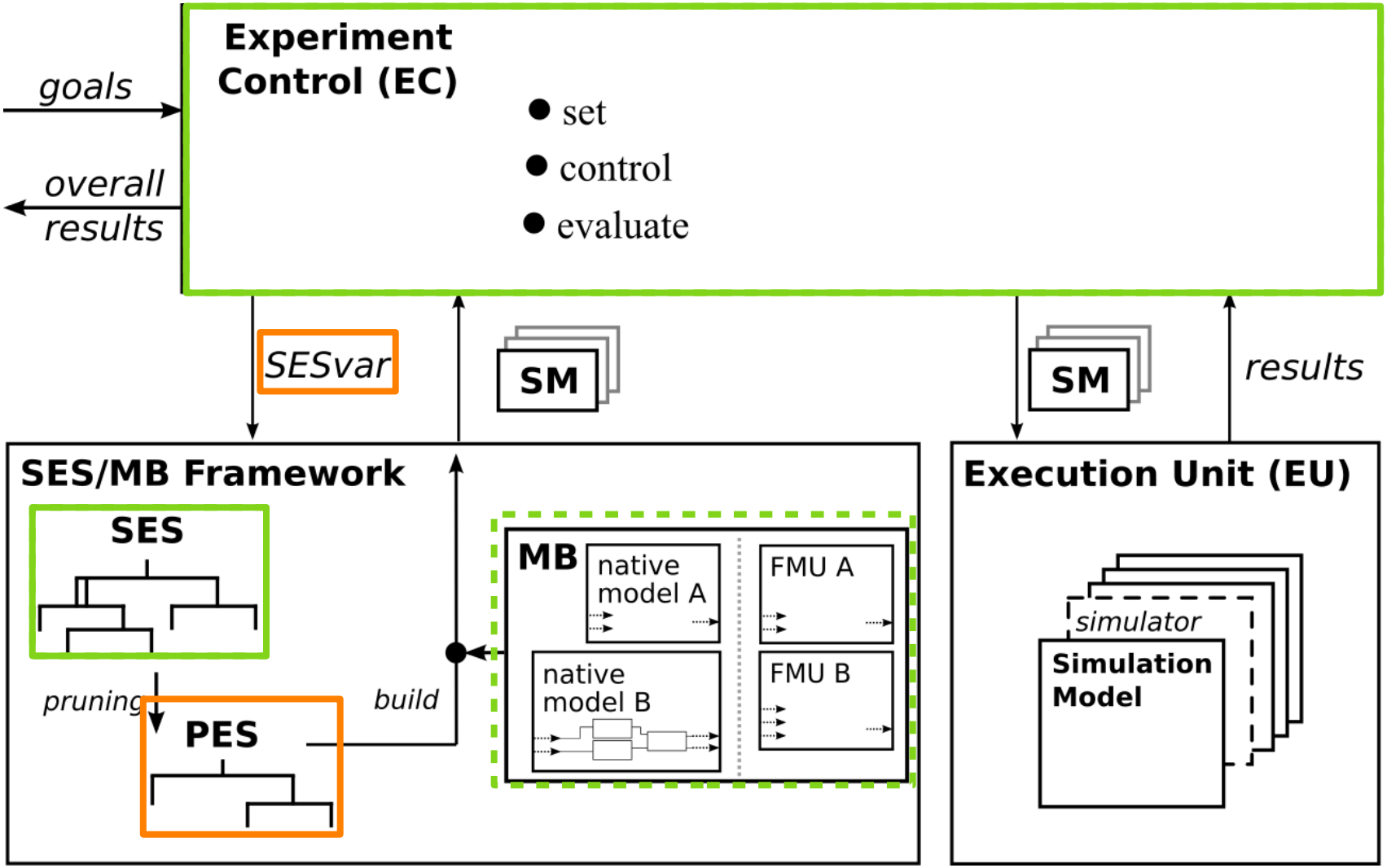
Extended SES/MB Architecture



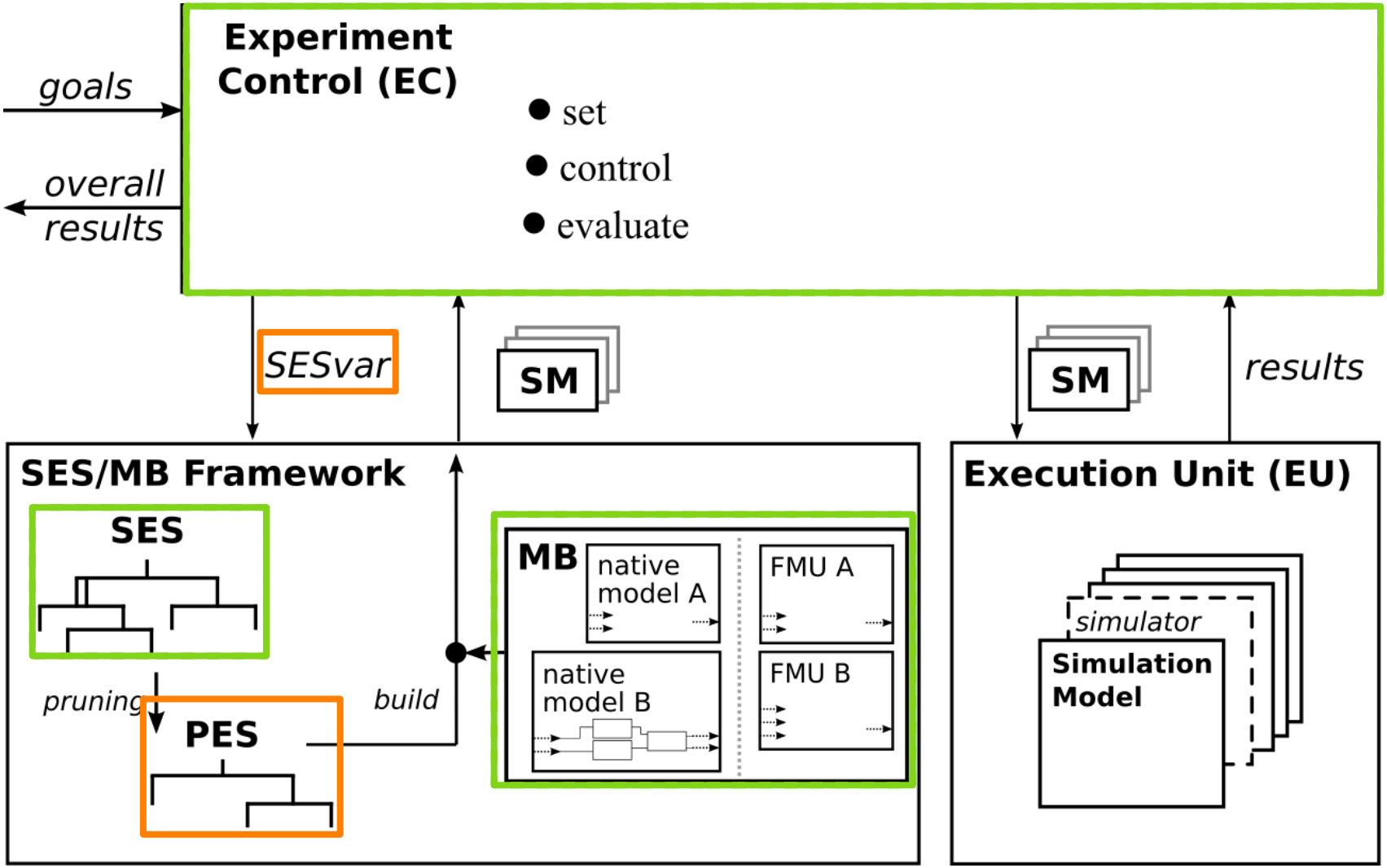
Extended SES/MB Architecture



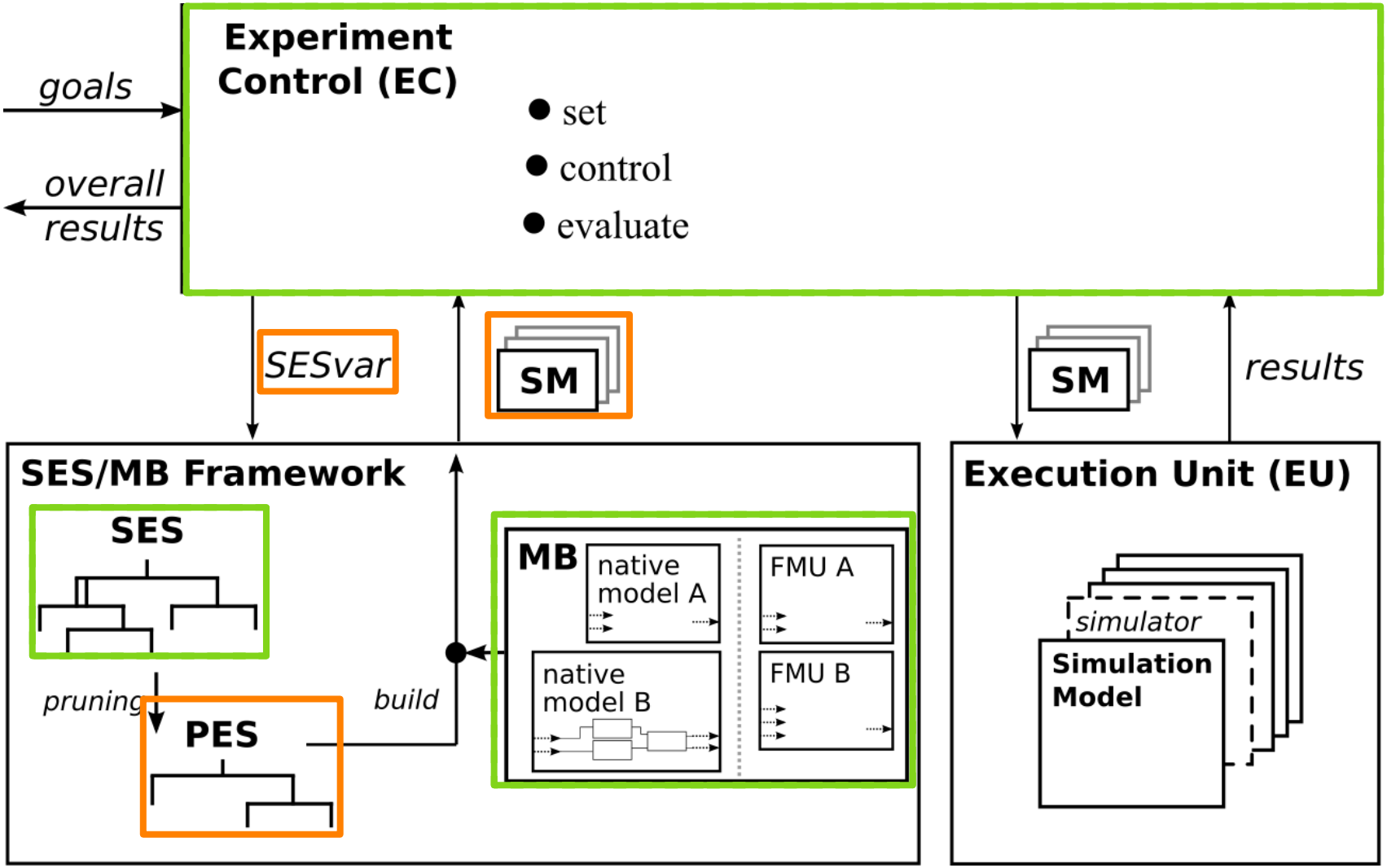
Extended SES/MB Architecture



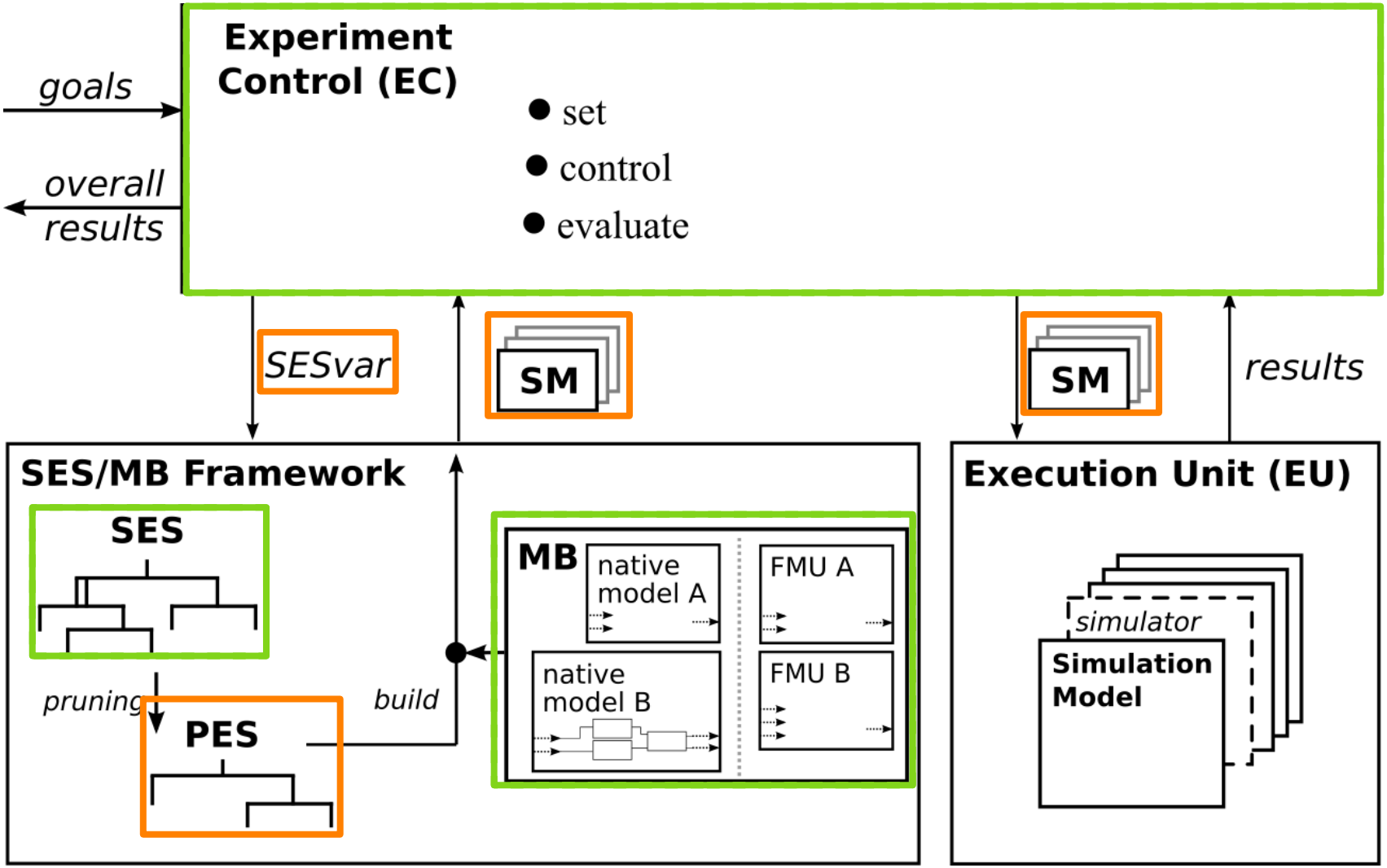
Extended SES/MB Architecture



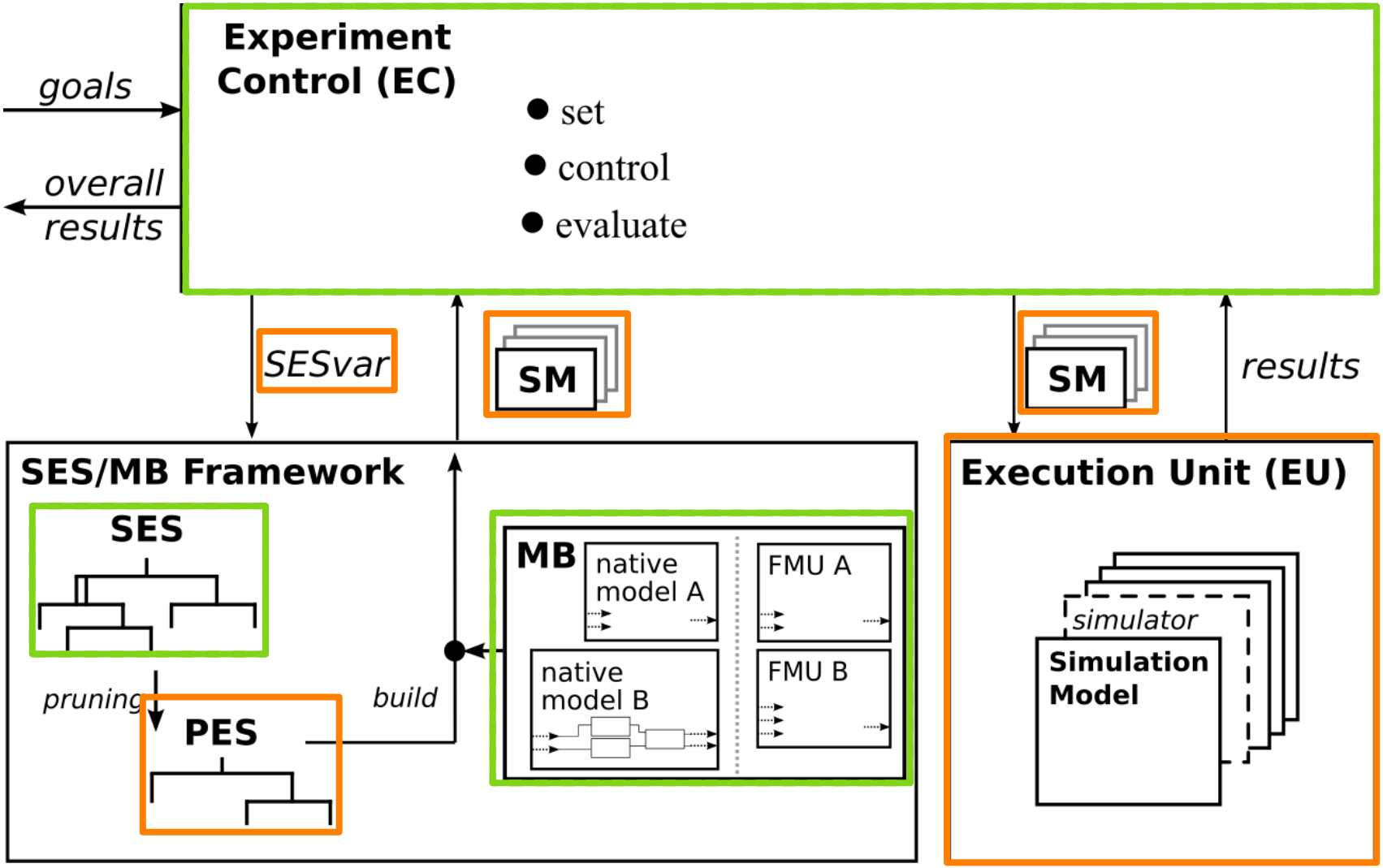
Extended SES/MB Architecture



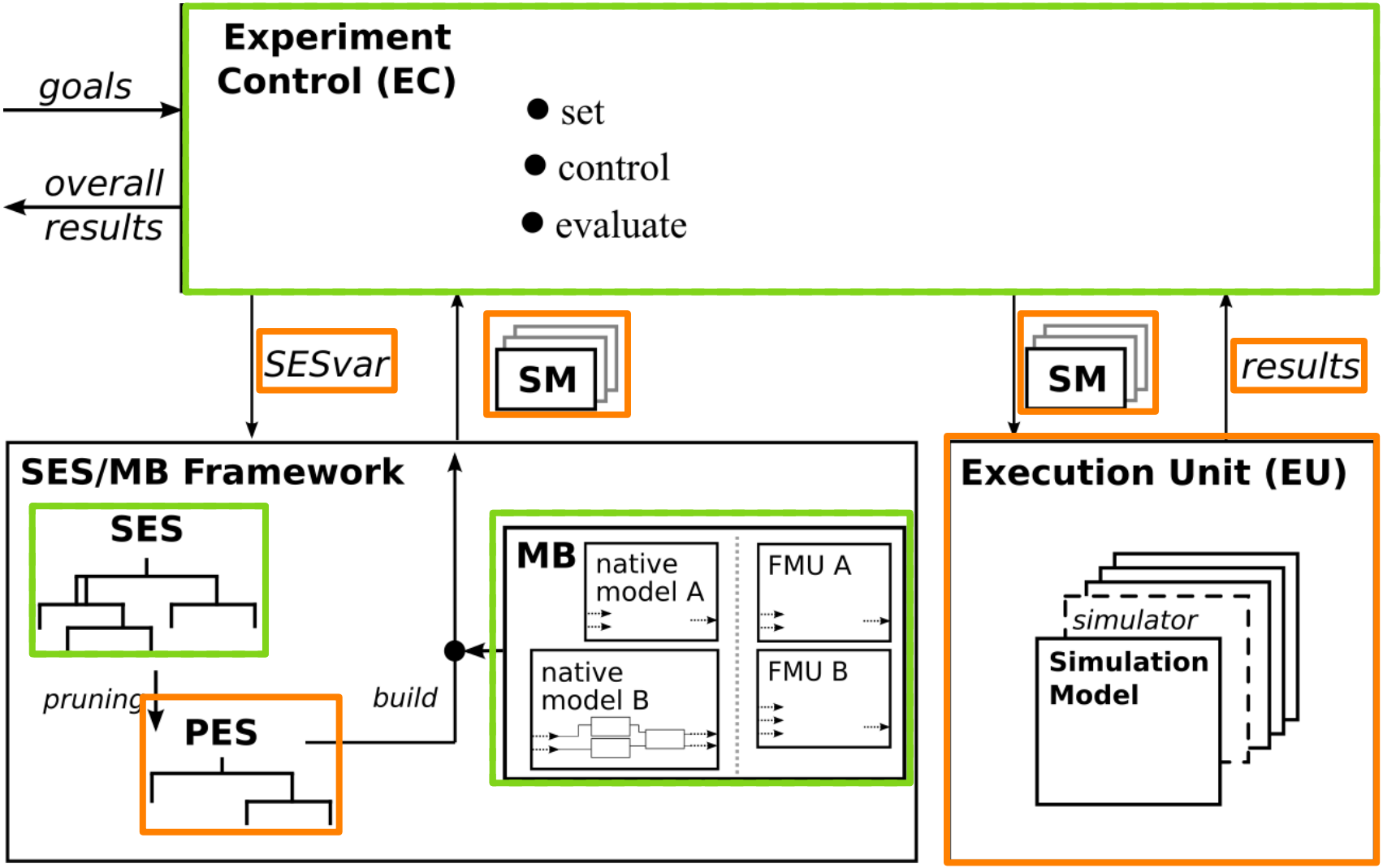
Extended SES/MB Architecture



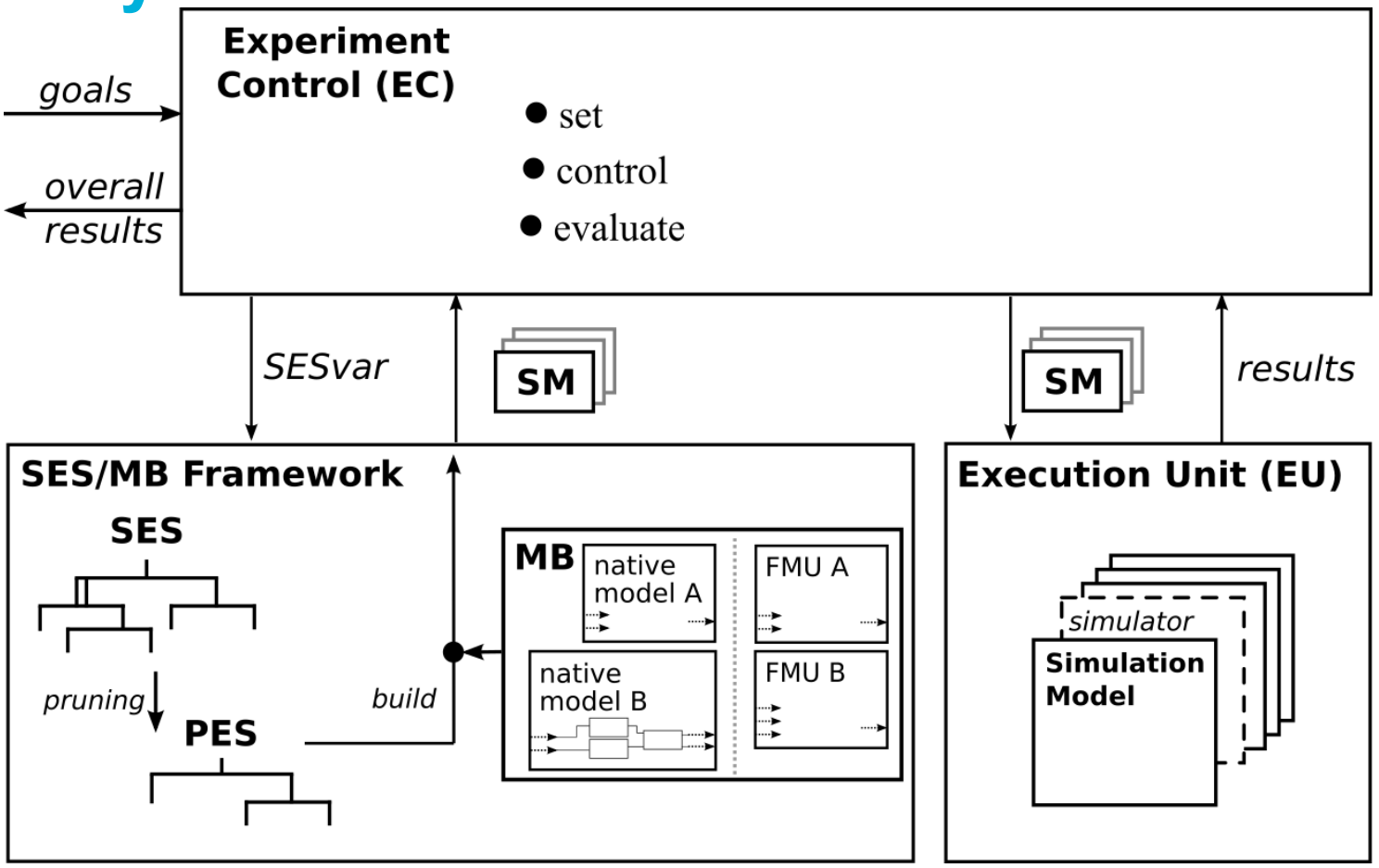
Extended SES/MB Architecture

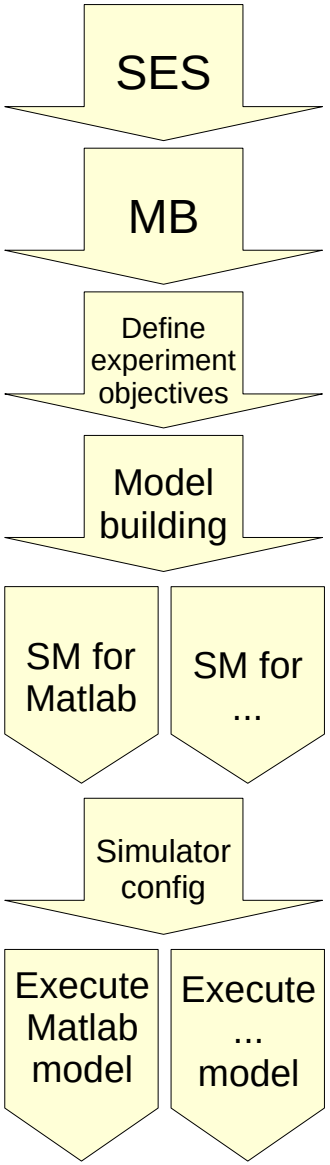


Extended SES/MB Architecture



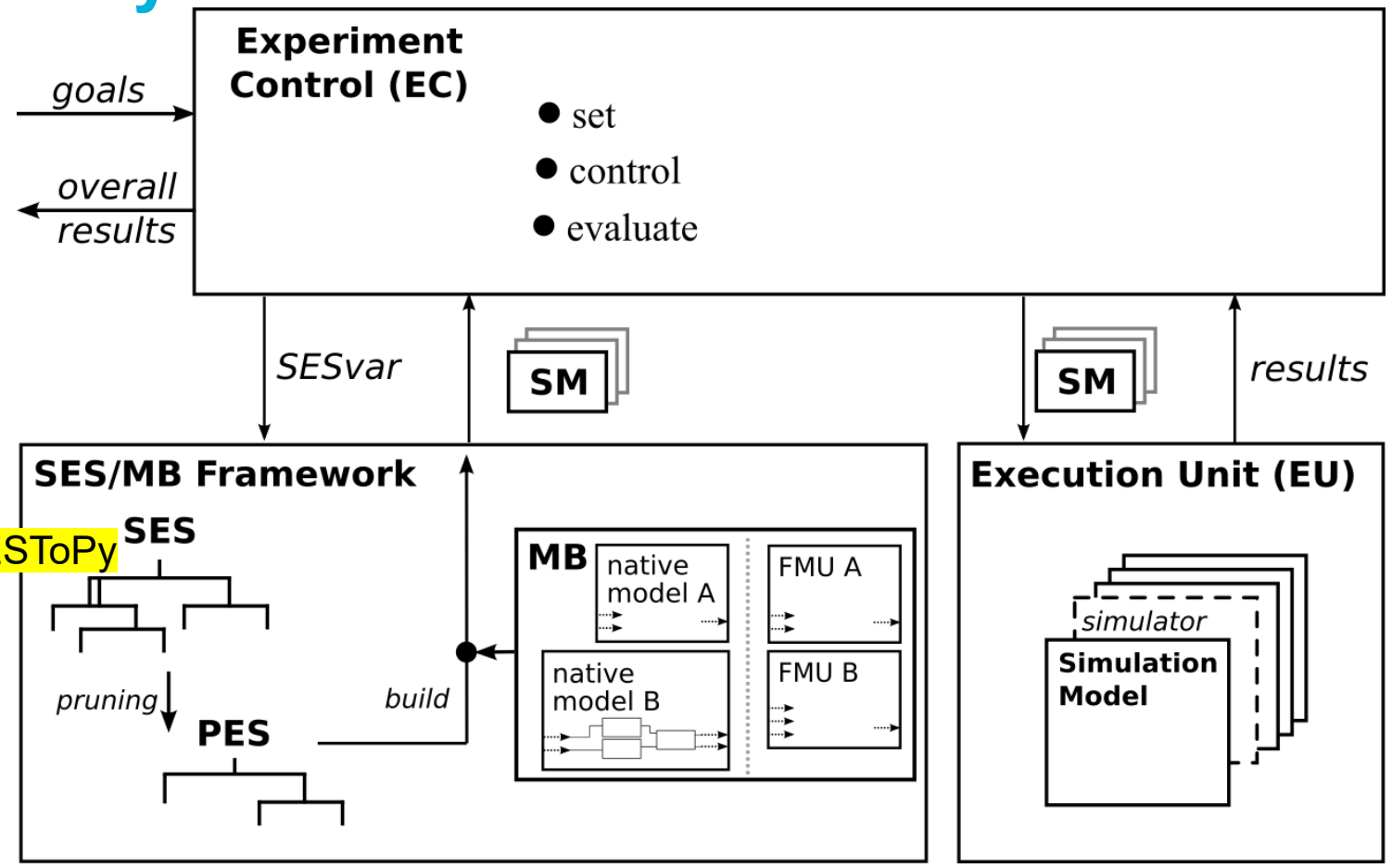
Extended SES/MB Architecture Python Tools





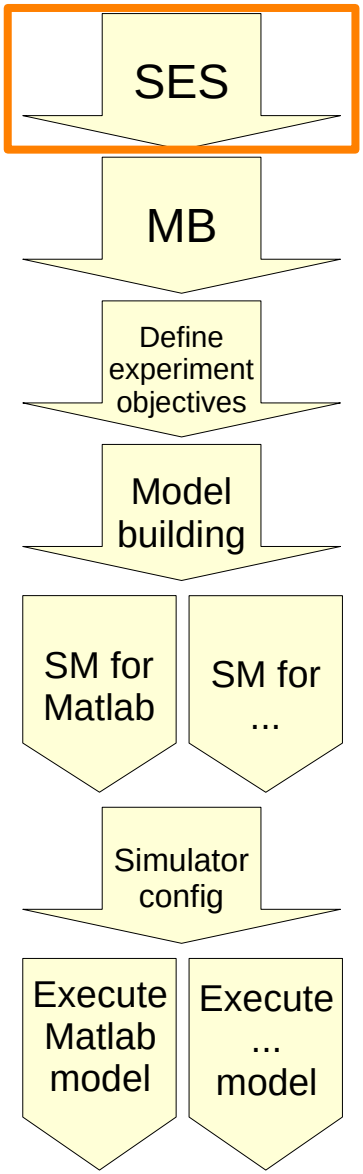
https://github.com/hendrikfolkerts

Extended SES/MB Architecture Python Tools

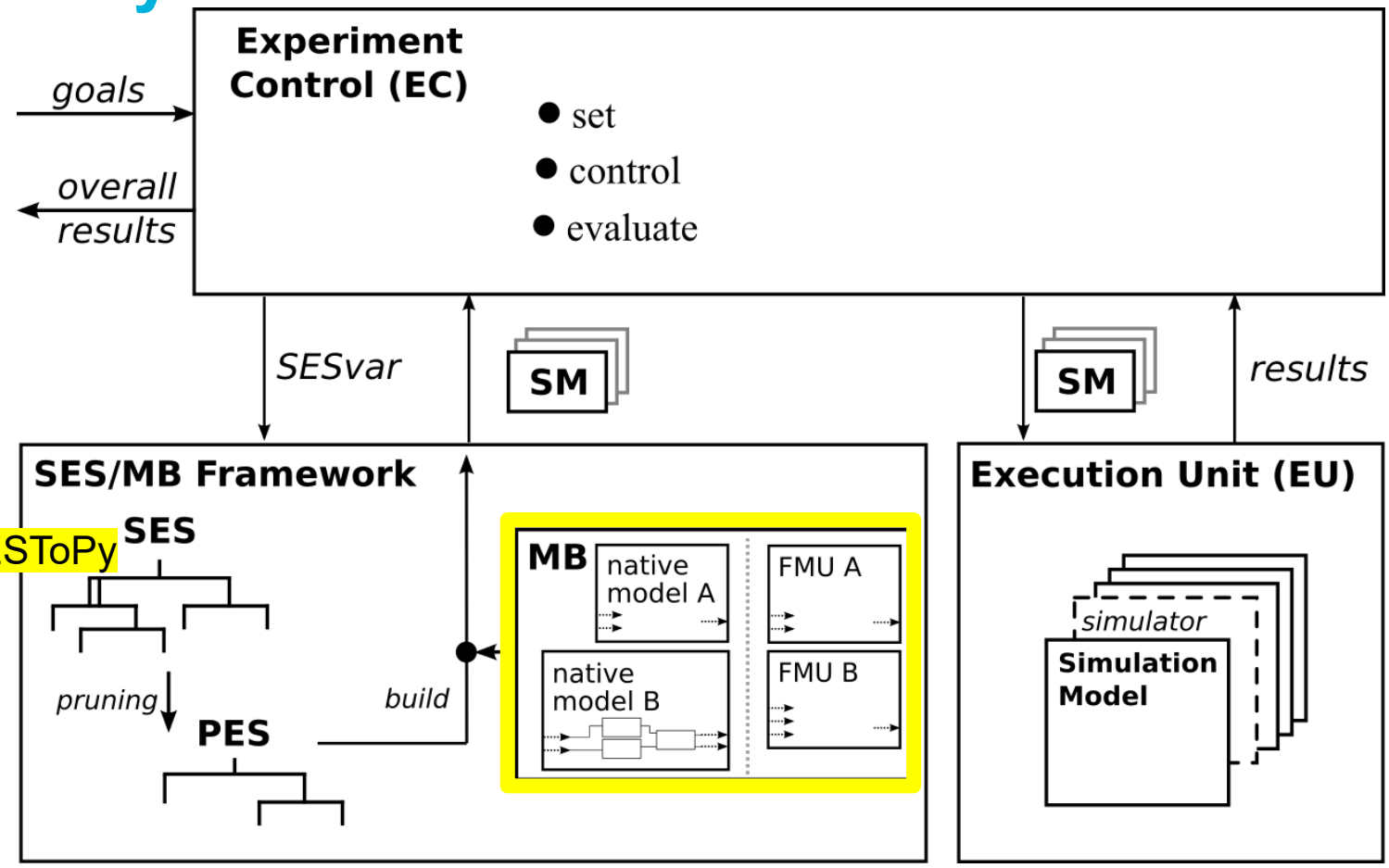


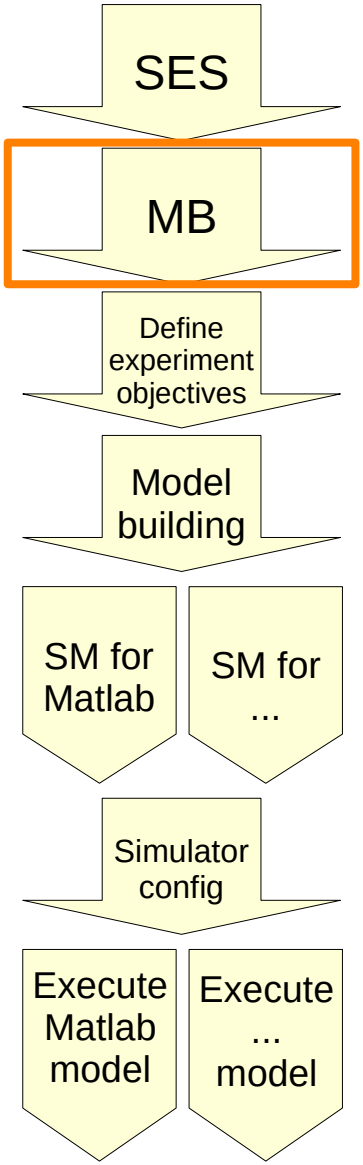
SESToPy

https://github.com/hendrikfolkerts



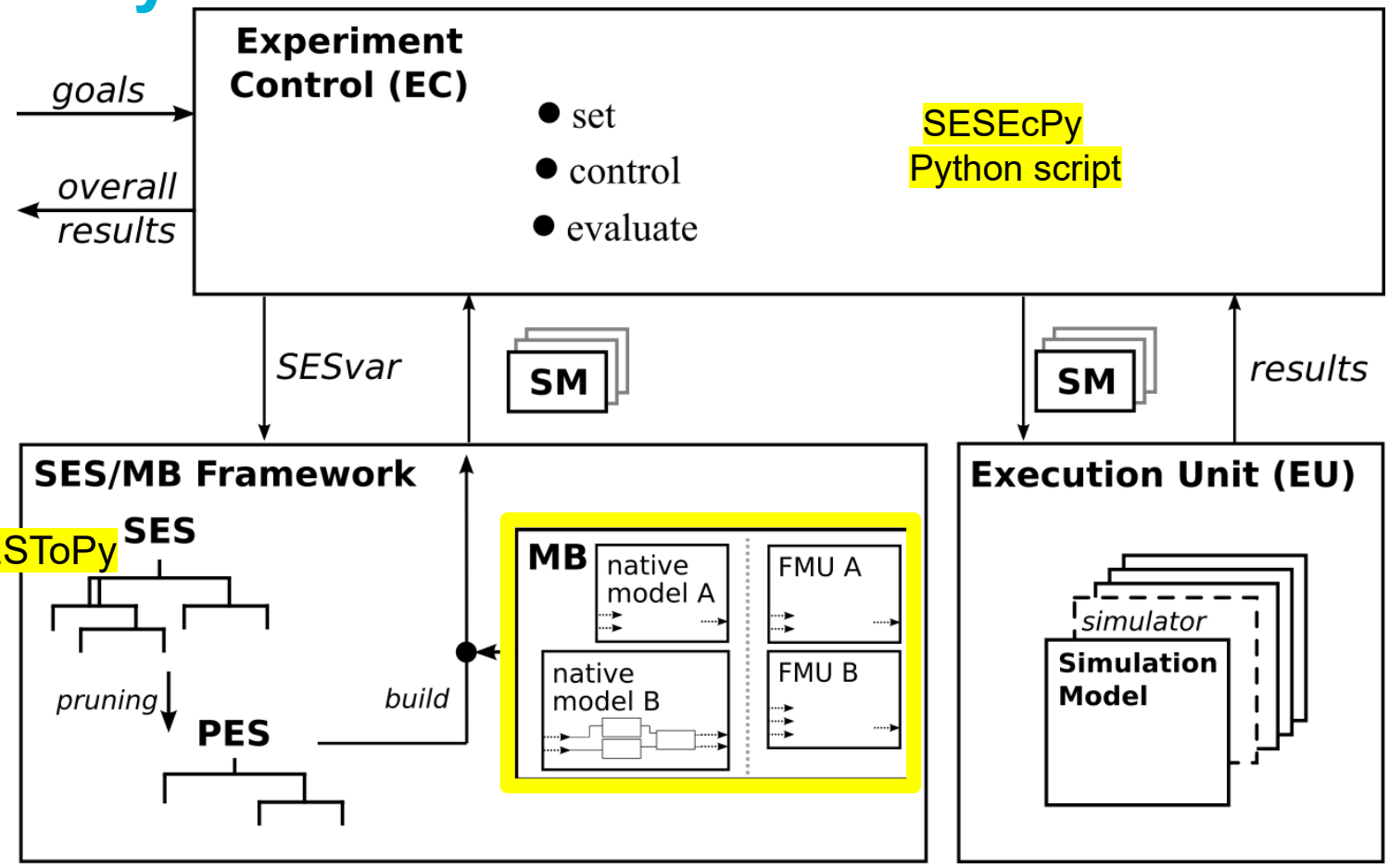
Extended SES/MB Architecture Python Tools

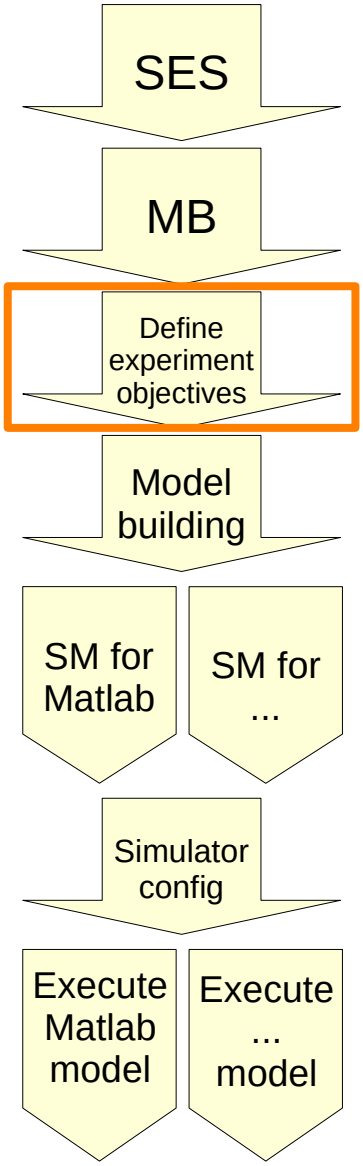




https://github.com/hendrikfolkerts

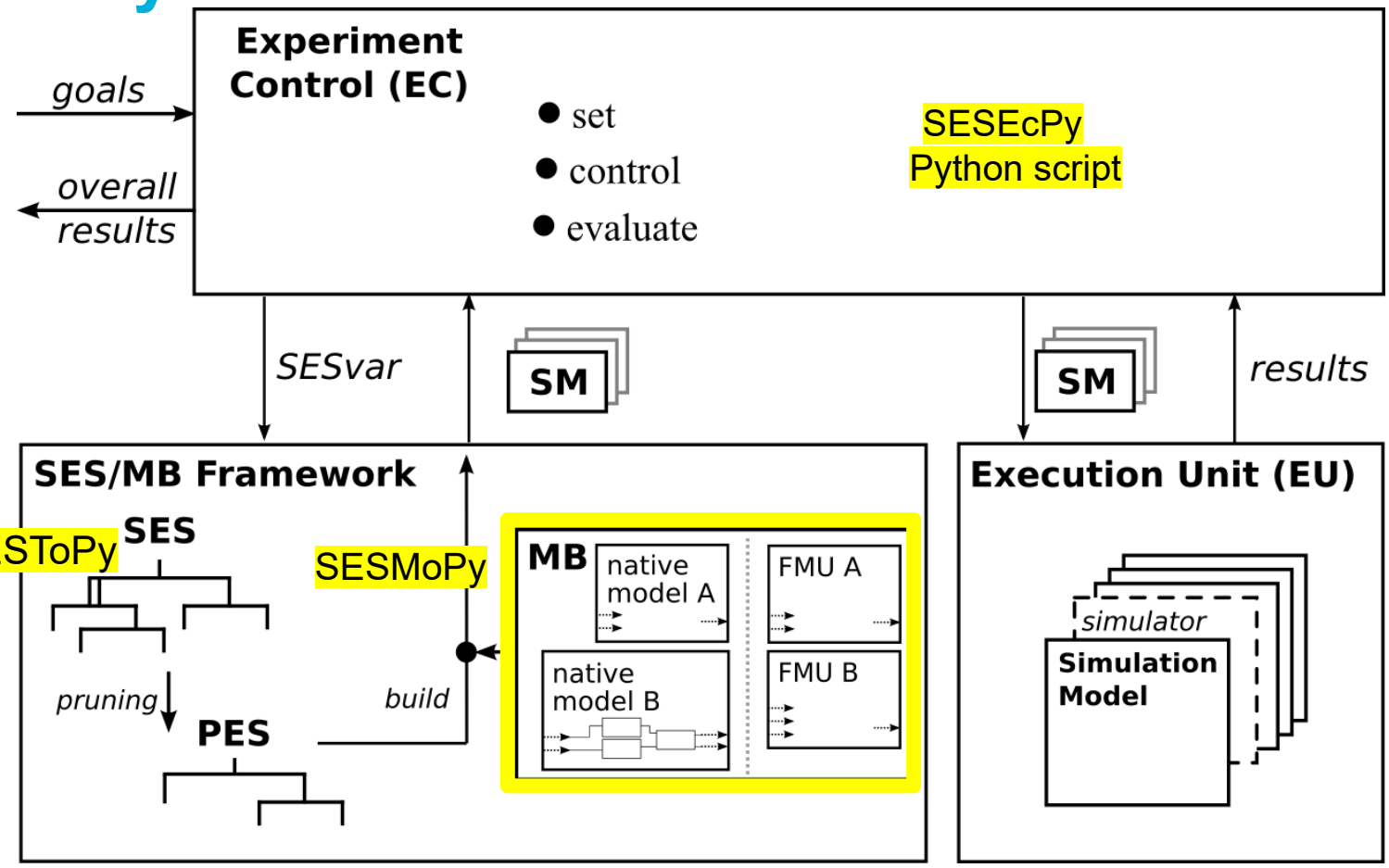
Extended SES/MB Architecture Python Tools

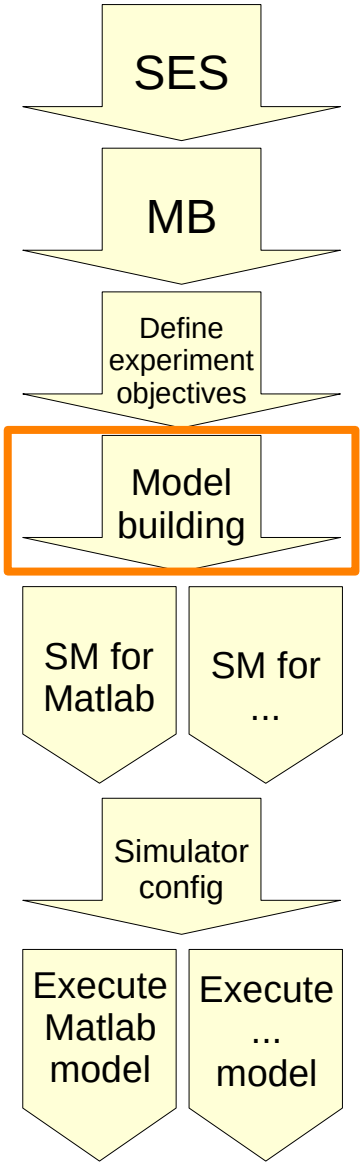




https://github.com/hendrikfolkerts

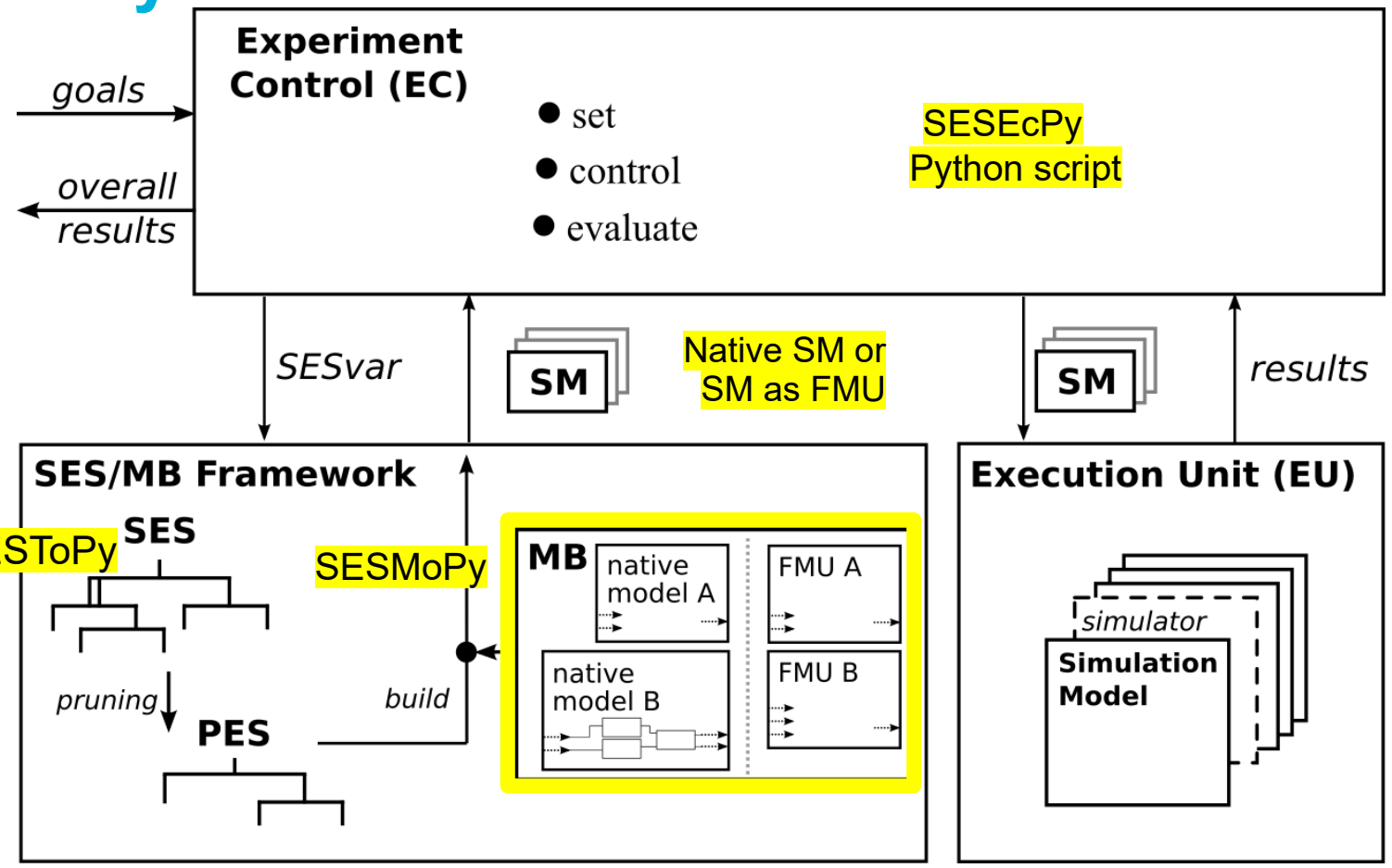
Extended SES/MB Architecture Python Tools

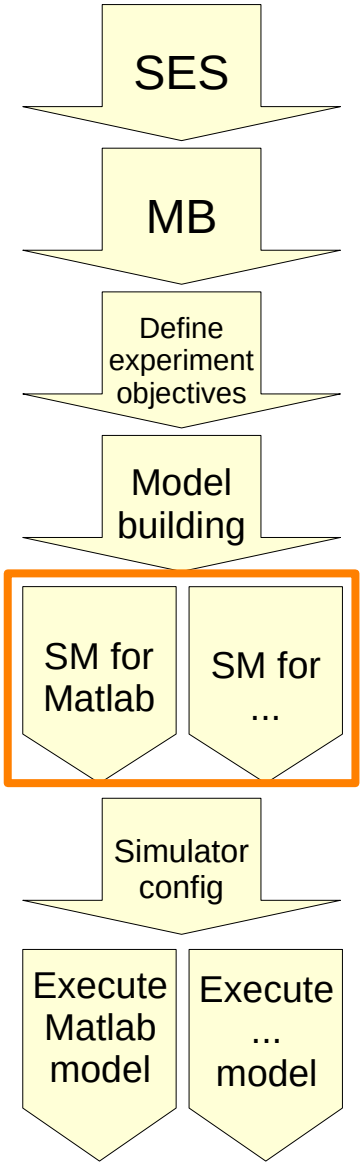




https://github.com/hendrikfolkerts

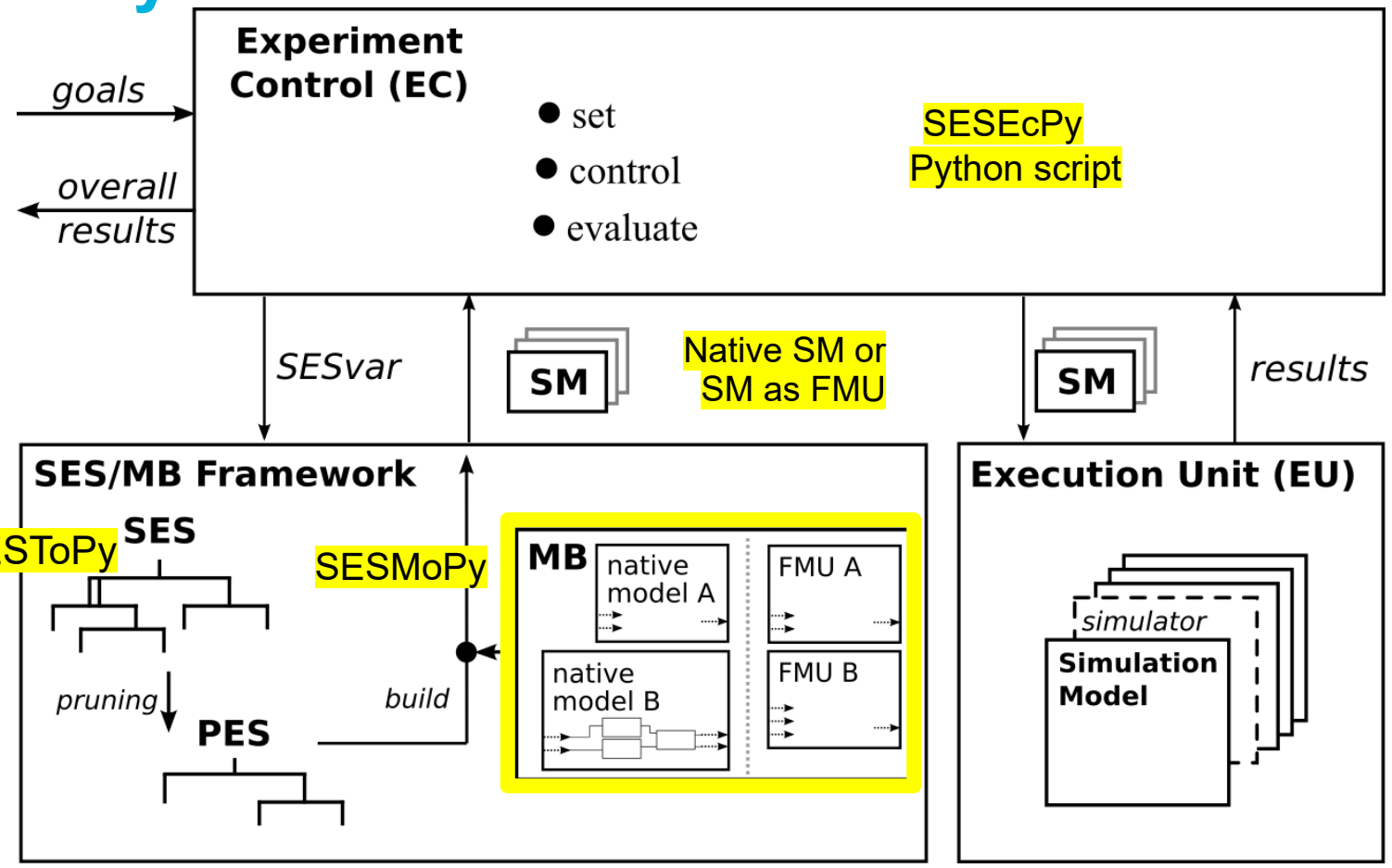
Extended SES/MB Architecture Python Tools

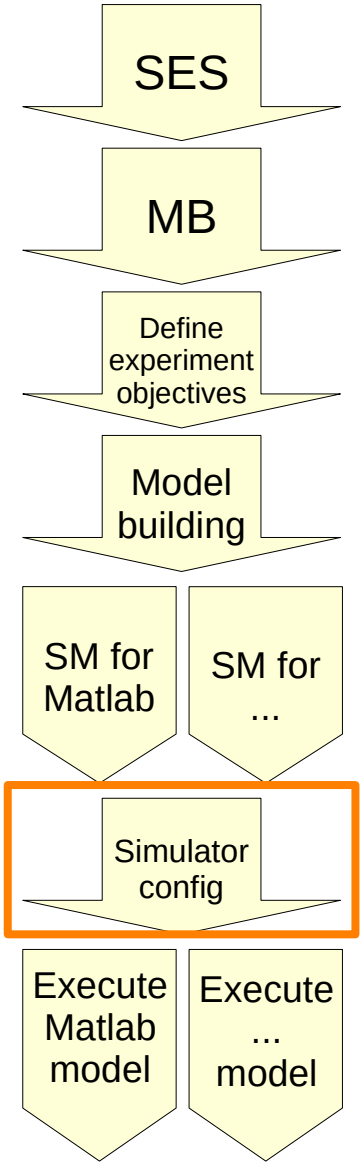




https://github.com/hendrikfolkerts

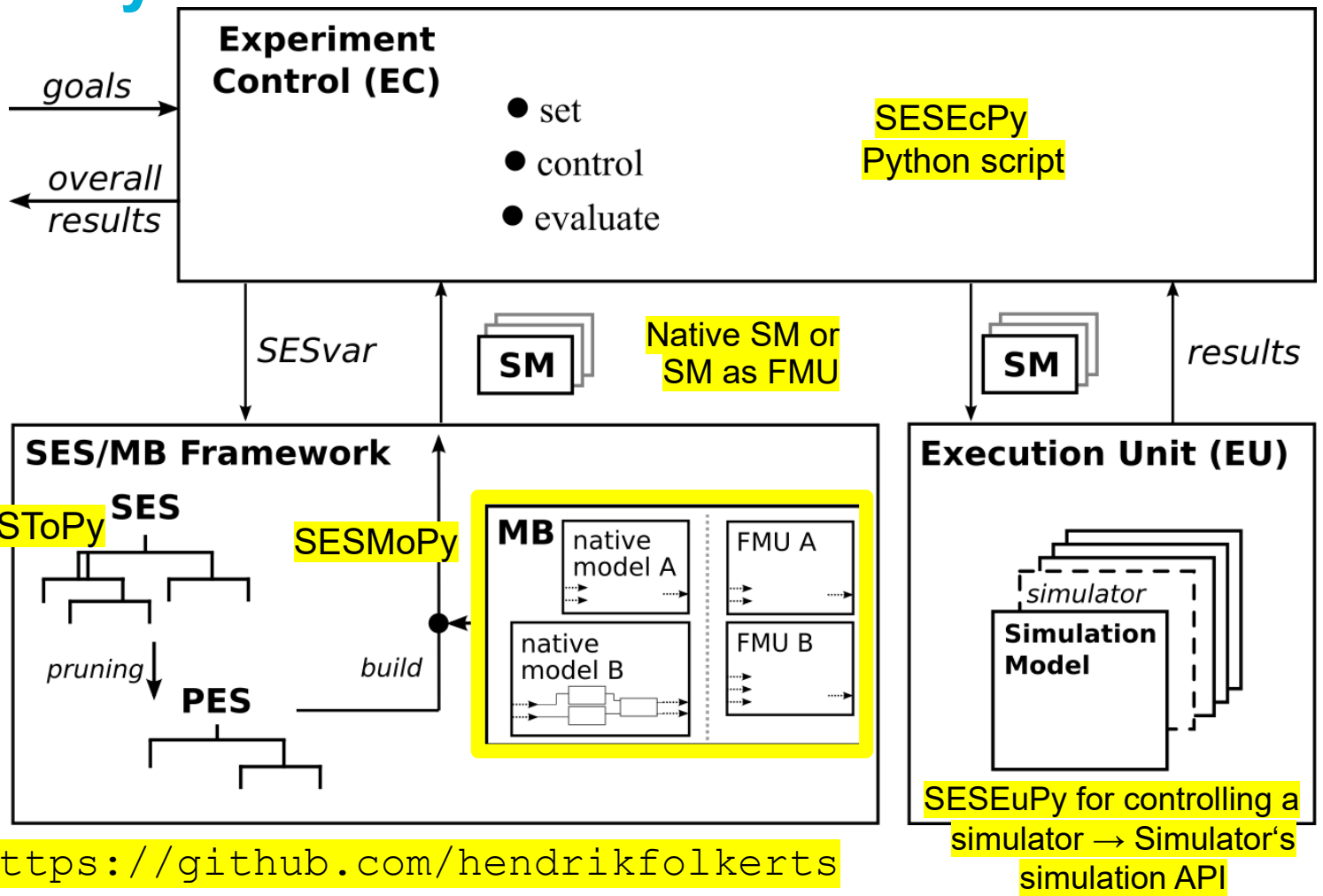
Extended SES/MB Architecture Python Tools

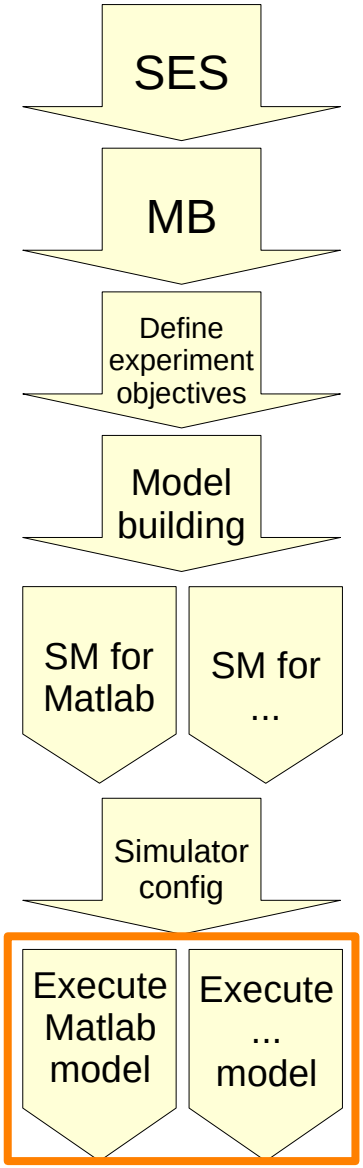




https://github.com/hendrikfolkerts

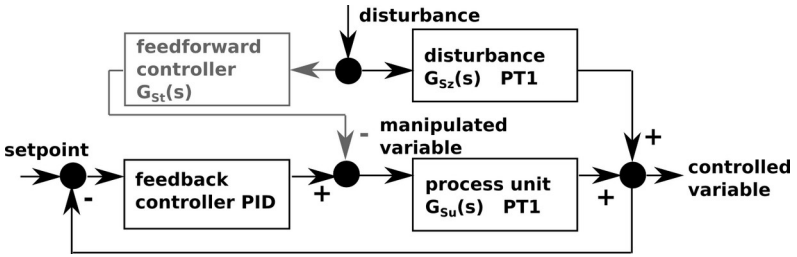
Extended SES/MB Architecture Python Tools

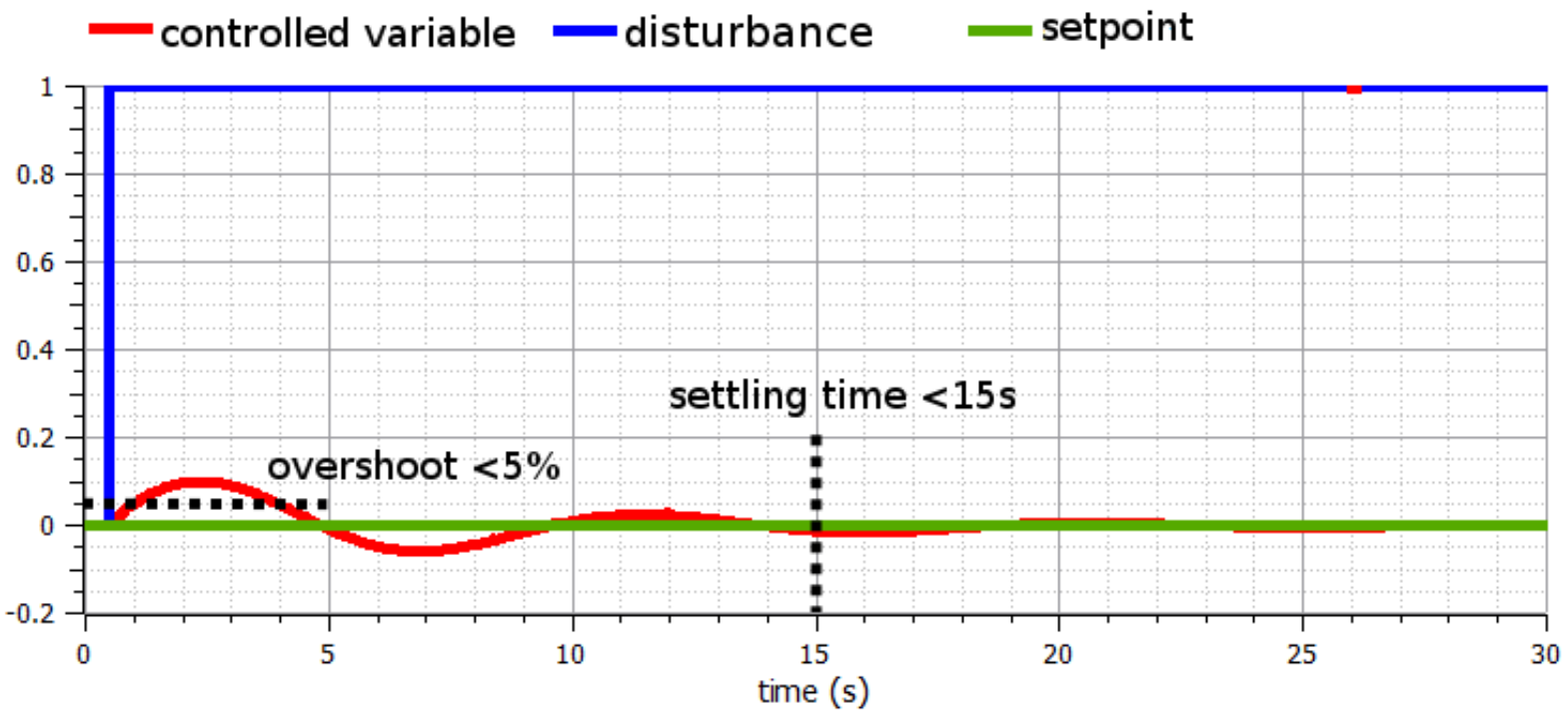




Case Study: Control Goals

- Goal for the control after a disturbance
  - Overshoot < 5%
  - Settling time < 15s





Case Study: Experimentation Steps

Case Study: Experimentation Steps

- → **Code in Experiment Control** ←

Case Study: Experimentation Steps

- → **Code in Experiment Control** ←

- **Try without a feedforward control:**
  - `feedforward=0` simulate with PID: `k=1, Ti=1, Td=0`
  - `feedforward=0` simulate with PID: `k=5, Ti=0.5, Td=0`

Case Study: Experimentation Steps

- → **Code in Experiment Control** ←

- **Try without a feedforward control:**
  - `feedforward=0` simulate with PID: `k=1, Ti=1, Td=0`
  - `feedforward=0` simulate with PID: `k=5, Ti=0.5, Td=0`

- **If the goals are reached with one of these configurations:**
  - Return PID configuration as overall result

Case Study: Experimentation Steps

- → **Code in Experiment Control** ←

- **Try without a feedforward control:**
  - `feedforward=0` simulate with PID: `k=1, Ti=1, Td=0`
  - `feedforward=0` simulate with PID: `k=5, Ti=0.5, Td=0`
- **If the goals are reached with one of these configurations:**
  - Return PID configuration as overall result
- **Else try with a feedforward control:**
  - `feedforward=1` simulate with both PID configurations

Case Study: Experimentation Steps

- → **Code in Experiment Control** ←

- **Try without a feedforward control:**
  - `feedforward=0` simulate with PID: `k=1, Ti=1, Td=0`
  - `feedforward=0` simulate with PID: `k=5, Ti=0.5, Td=0`
- **If the goals are reached with one of these configurations:**
  - Return PID configuration as overall result
- **Else try with a feedforward control:**
  - `feedforward=1` simulate with both PID configurations
- **If the goals are reached with one of these configurations:**
  - Return PID configuration as overall result

Case Study: Experimentation Steps

- → **Code in Experiment Control** ←

- **Try without a feedforward control:**
  - `feedforward=0` simulate with PID: `k=1, Ti=1, Td=0`
  - `feedforward=0` simulate with PID: `k=5, Ti=0.5, Td=0`
- **If the goals are reached with one of these configurations:**
  - Return PID configuration as overall result
- **Else try with a feedforward control:**
  - `feedforward=1` simulate with both PID configurations
- **If the goals are reached with one of these configurations:**
  - Return PID configuration as overall result
- **Else return: Goals cannot be reached with these configurations / parameters**

Case Study: Experimentation Steps

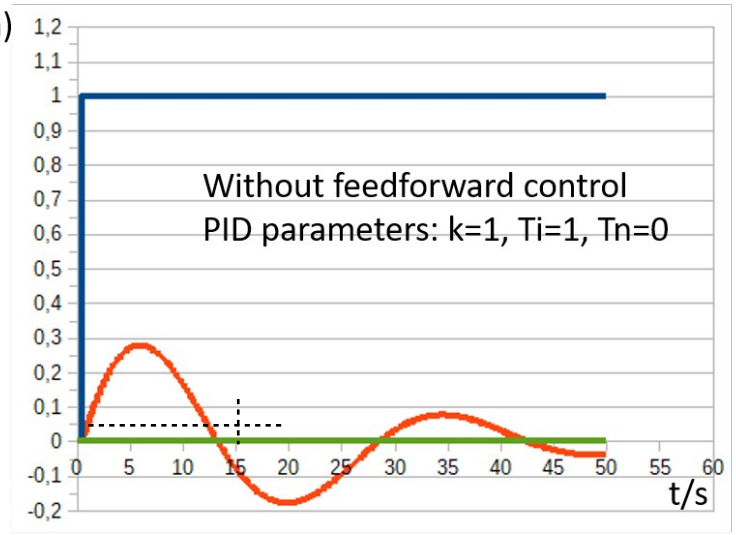
- → **Code in Experiment Control** ←

- **Try without a feedforward control:**
  - `feedforward=0` simulate with PID: `k=1, Ti=1, Td=0`
  - `feedforward=0` simulate with PID: `k=5, Ti=0.5, Td=0`
- **If the goals are reached with one of these configurations:**
  - Return PID configuration as overall result
- **Else try with a feedforward control:**
  - `feedforward=1` simulate with both PID configurations
- **If the goals are reached with one of these configurations:**
  - Return PID configuration as overall result
- **Else return: Goals cannot be reached with these configurations / parameters**

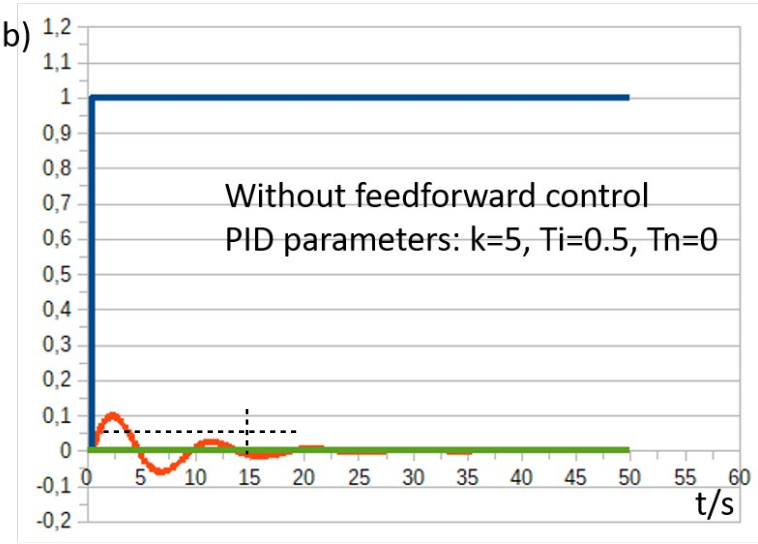
- **Start over with another simulator → model by model validation**

Case Study: Simulation Results

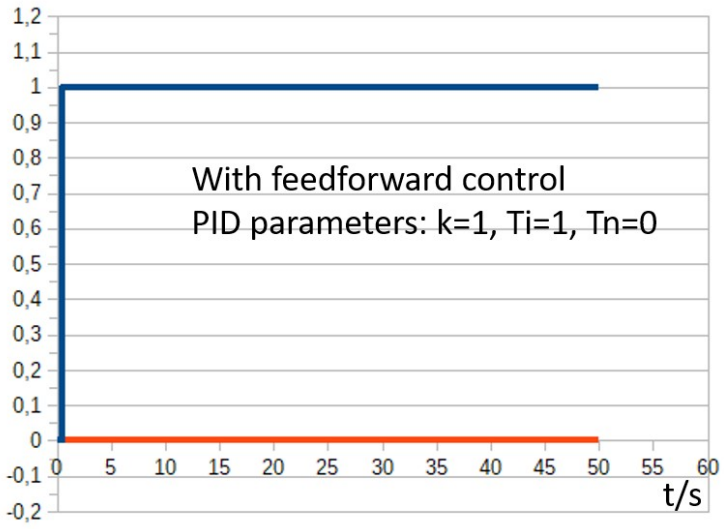
(a)



(b)



(c)

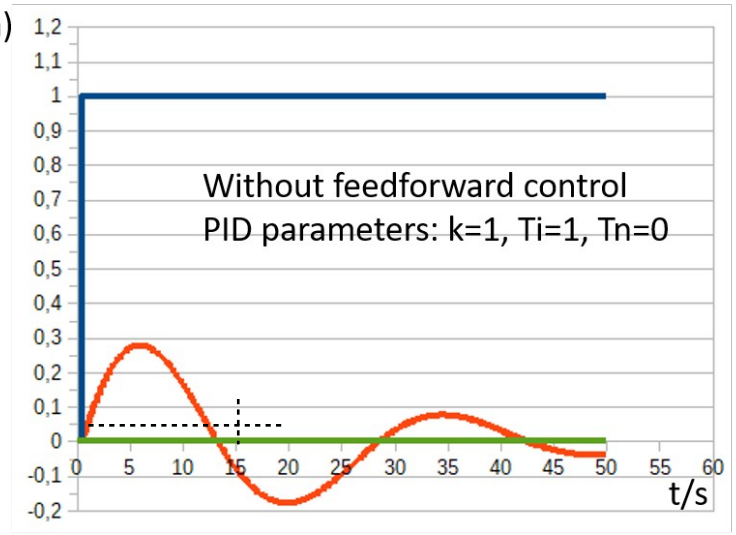


Key:
- setpoint
- disturbance
- controlled variable

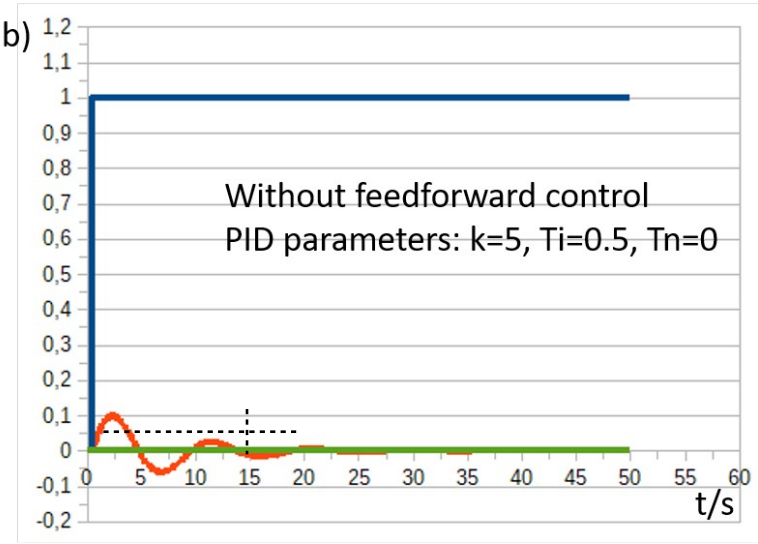
Control goals:
overshoot < 5%
settling time < 15s

Case Study: Simulation Results

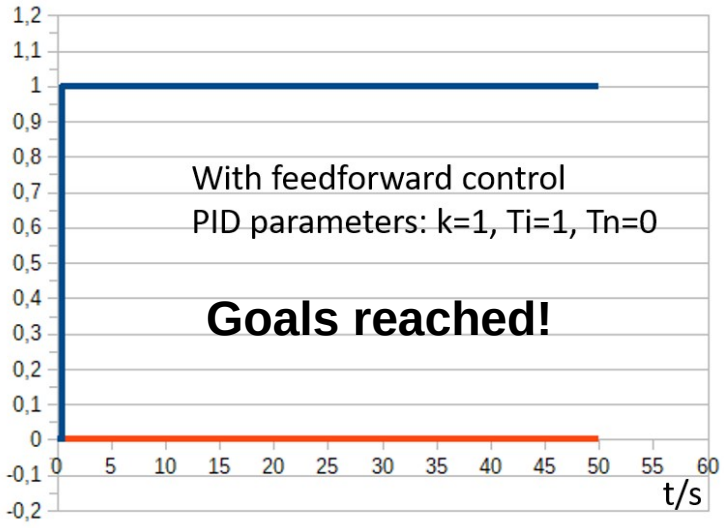
(a)



(b)



(c)



Key:
- setpoint
- disturbance
- controlled variable

Control goals:
overshoot < 5%
settling time < 15s

Outline

1. Introduction
2. The case study
3. Basics of SES/MB based modeling
4. Practical modeling: implementation of an SES
5. Model selection and model generation
6. The free Matlab SES toolbox
7. Organization of a simulator-independent MB
8. Full automation of simulation experiments
9. **Conclusion** (T. Pawletta)

Conclusion

Sources for presented tools:

https://github.com/cea-wismar

https://github.com/hendrikfolkerts

Conclusion

- SES supports simulator-independent modeling of model configurations regarding model structures and parameter settings

Sources for presented tools:

`https://github.com/cea-wismar`

`https://github.com/hendrikfolkerts`

Conclusion

- SES supports simulator-independent modeling of model configurations regarding model structures and parameter settings
- MBs are usually simulator-specific (and system dynamic specific)
  - No problem, if working in only one M&S environment
  - Difficult maintenance, if working with multiple simulators

Sources for presented tools:

`https://github.com/cea-wismar`

`https://github.com/hendrikfolkerts`

Conclusion

- SES supports simulator-independent modeling of model configurations regarding model structures and parameter settings
- MBs are usually simulator-specific (and system dynamic specific)
  - No problem, if working in only one M&S environment
  - Difficult maintenance, if working with multiple simulators
- Using FMI a simulator-independent MB is possible
  - Support for efficient model building for multiple simulators (still problems for discrete event models)

Sources for presented tools:

`https://github.com/cea-wismar`

`https://github.com/hendrikfolkerts`

Conclusion

- SES supports simulator-independent modeling of model configurations regarding model structures and parameter settings
- MBs are usually simulator-specific (and system dynamic specific)
  - No problem, if working in only one M&S environment
  - Difficult maintenance, if working with multiple simulators
- Using FMI a simulator-independent MB is possible
  - Support for efficient model building for multiple simulators (still problems for discrete event models)
- The Extended SES/MB Architecture supports a full experiment automation regarding defined design objectives (using multiple simulators)

Sources for presented tools:

`https://github.com/cea-wismar`

`https://github.com/hendrikfolkerts`

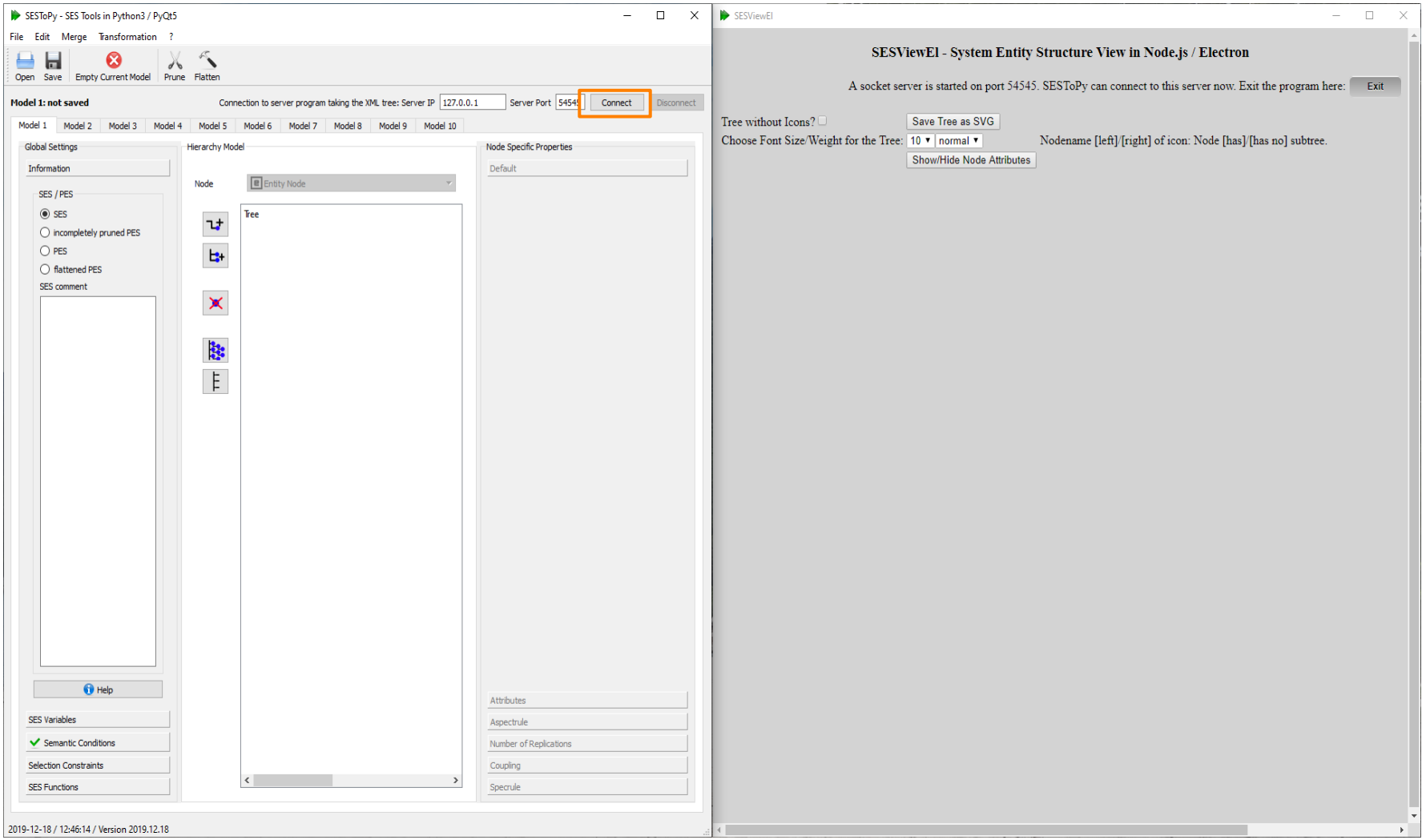
Thank you!

Questions?

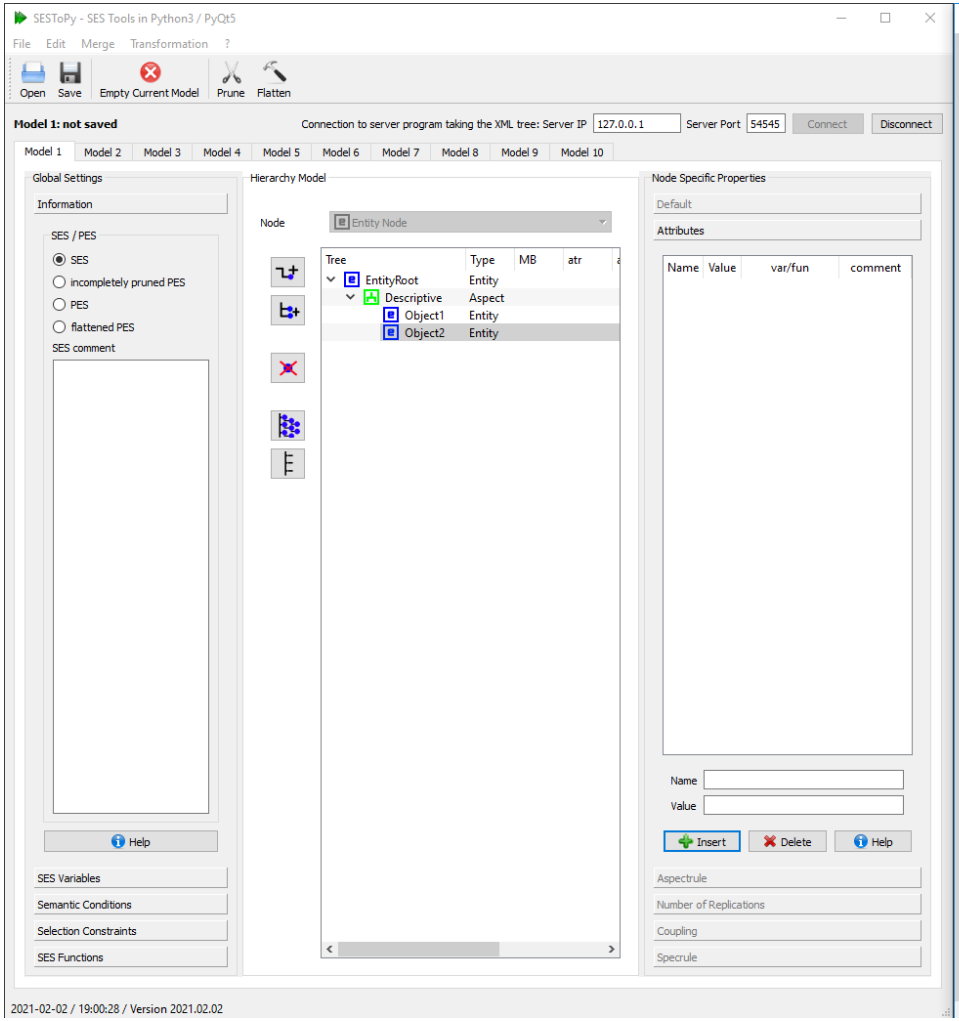
Backup

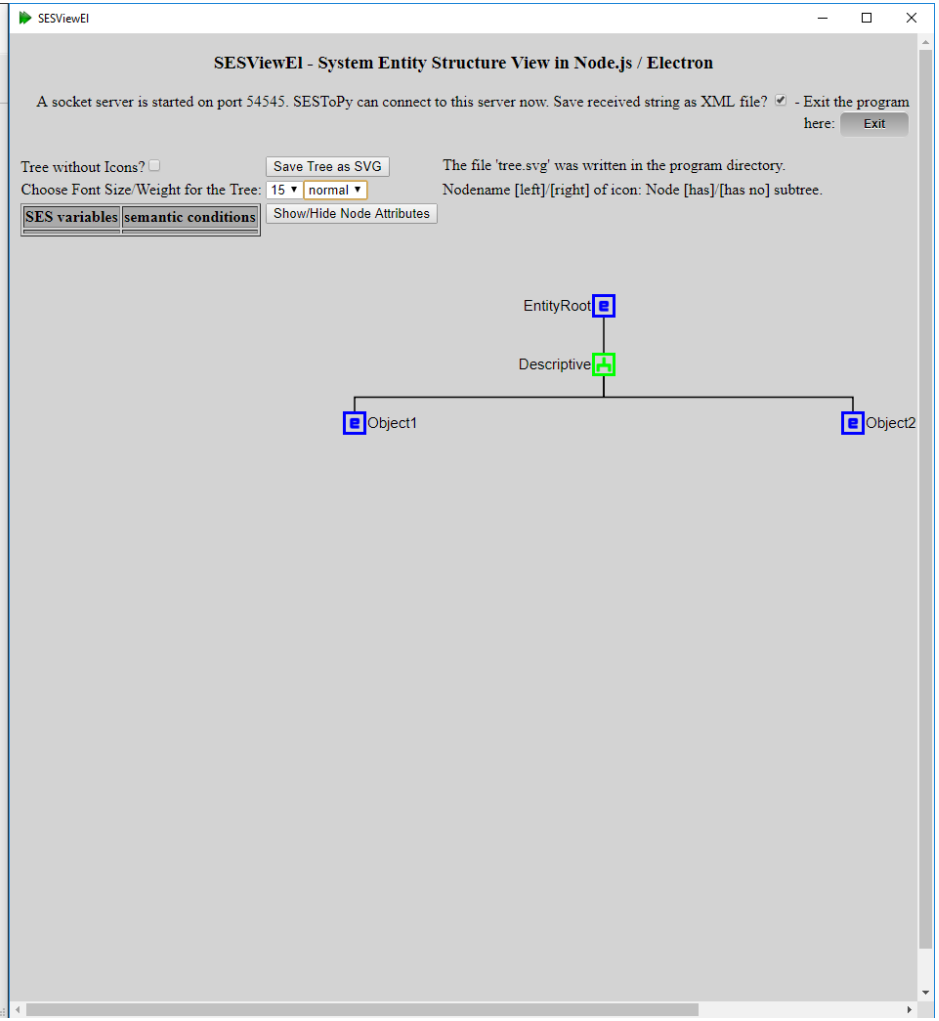
Chapter 4: Demonstration of SESToPy and SESViewEl Backup Slides

Connection of SESToPy and SESViewEl

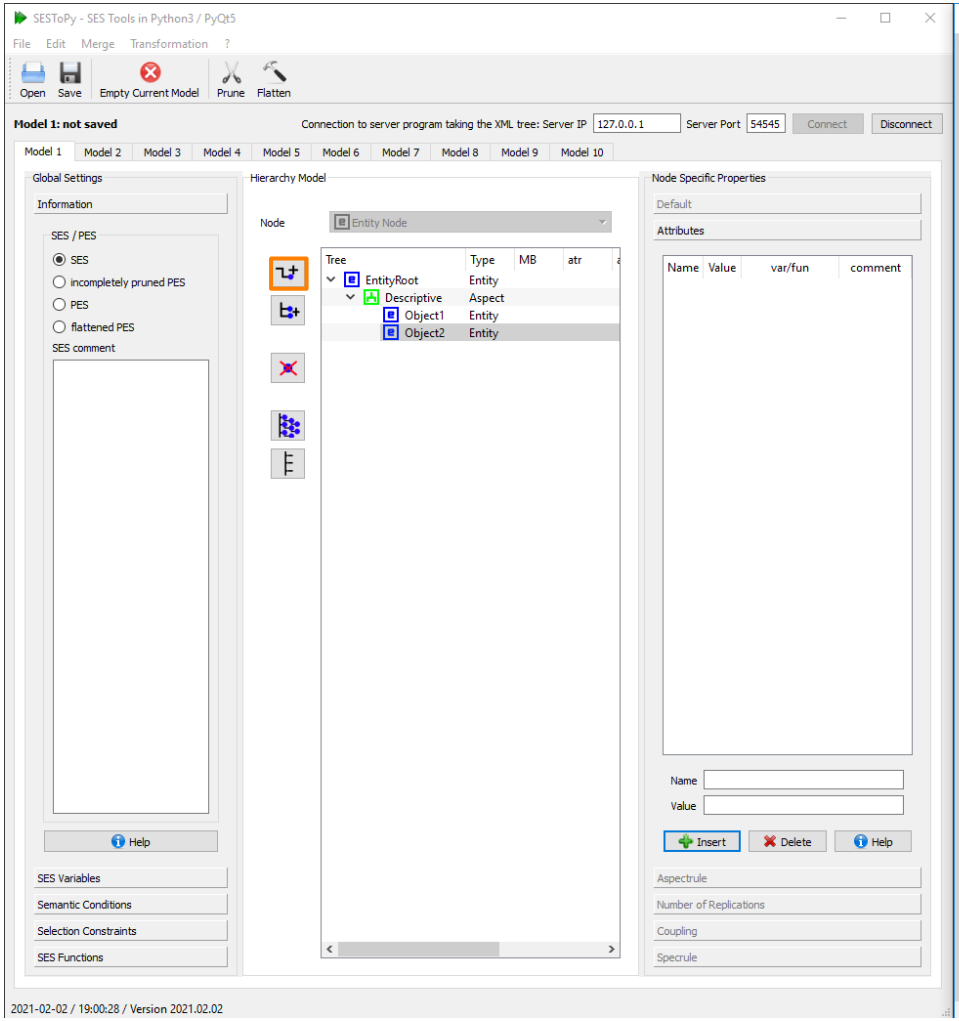


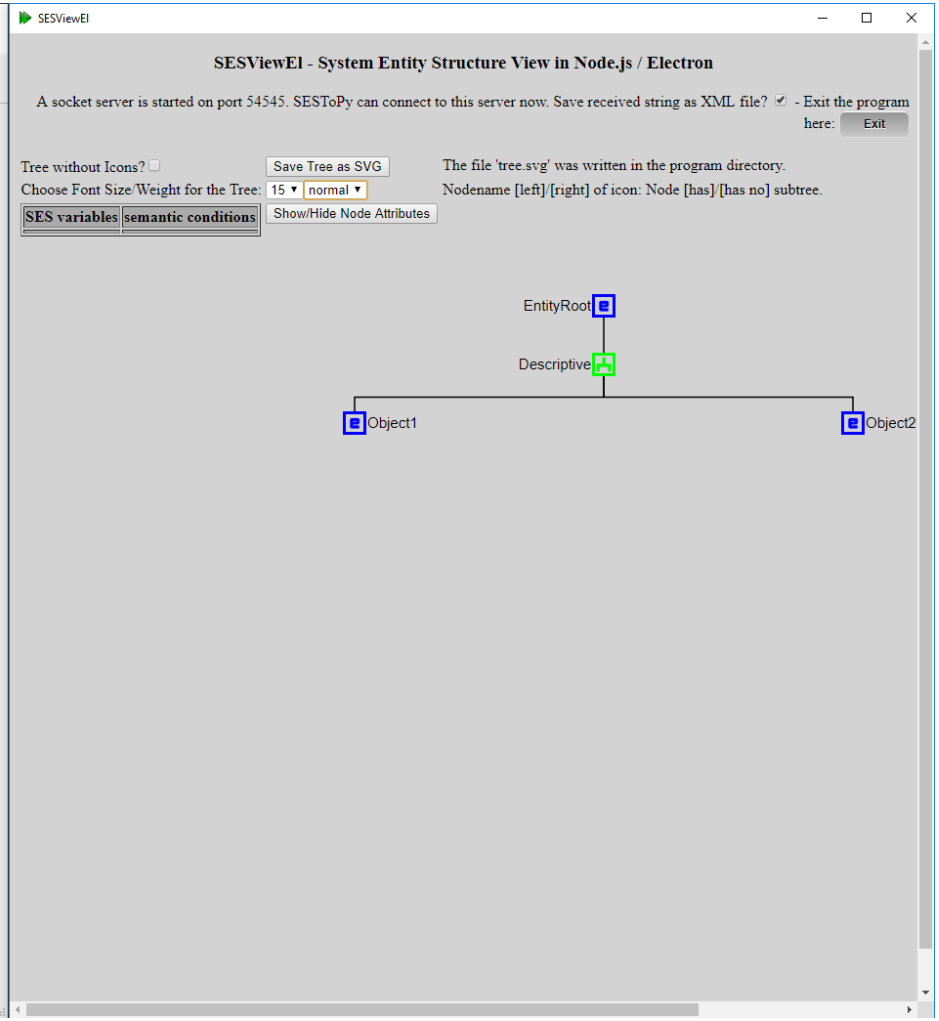
Create SES Tree with SESToPy



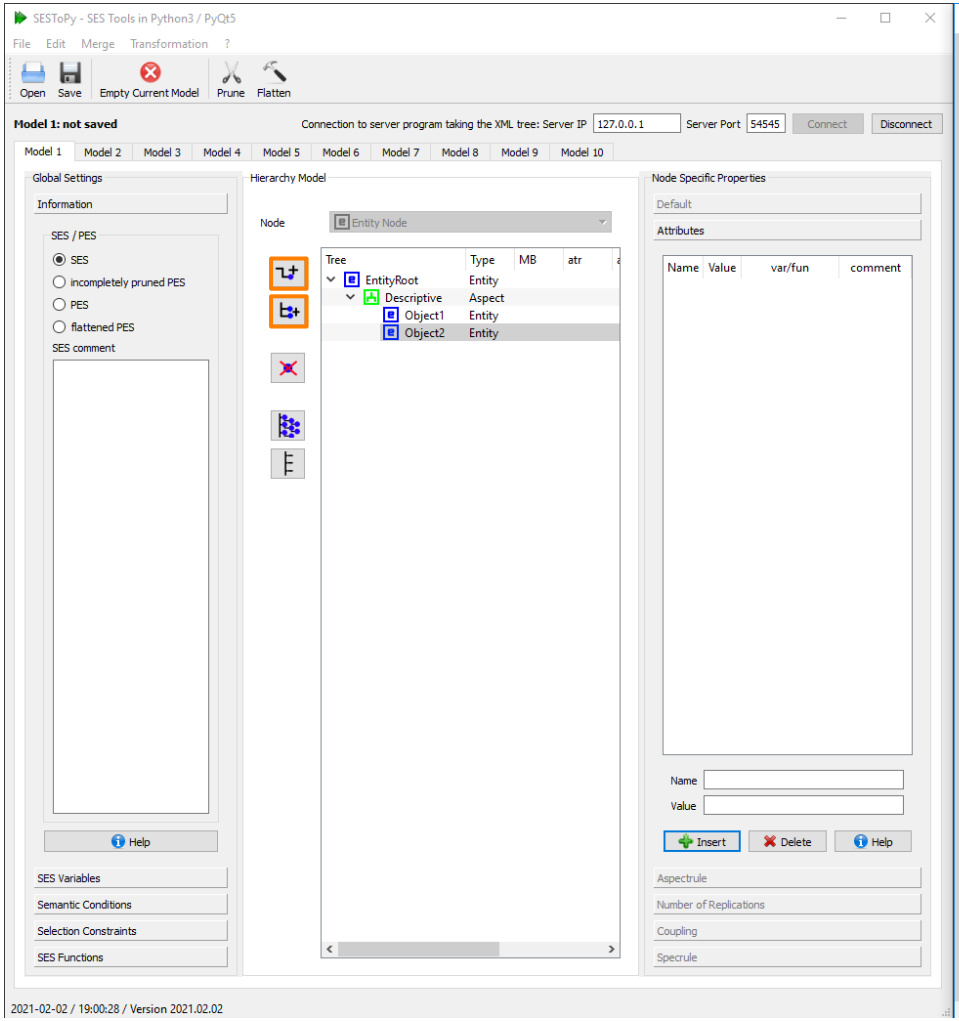


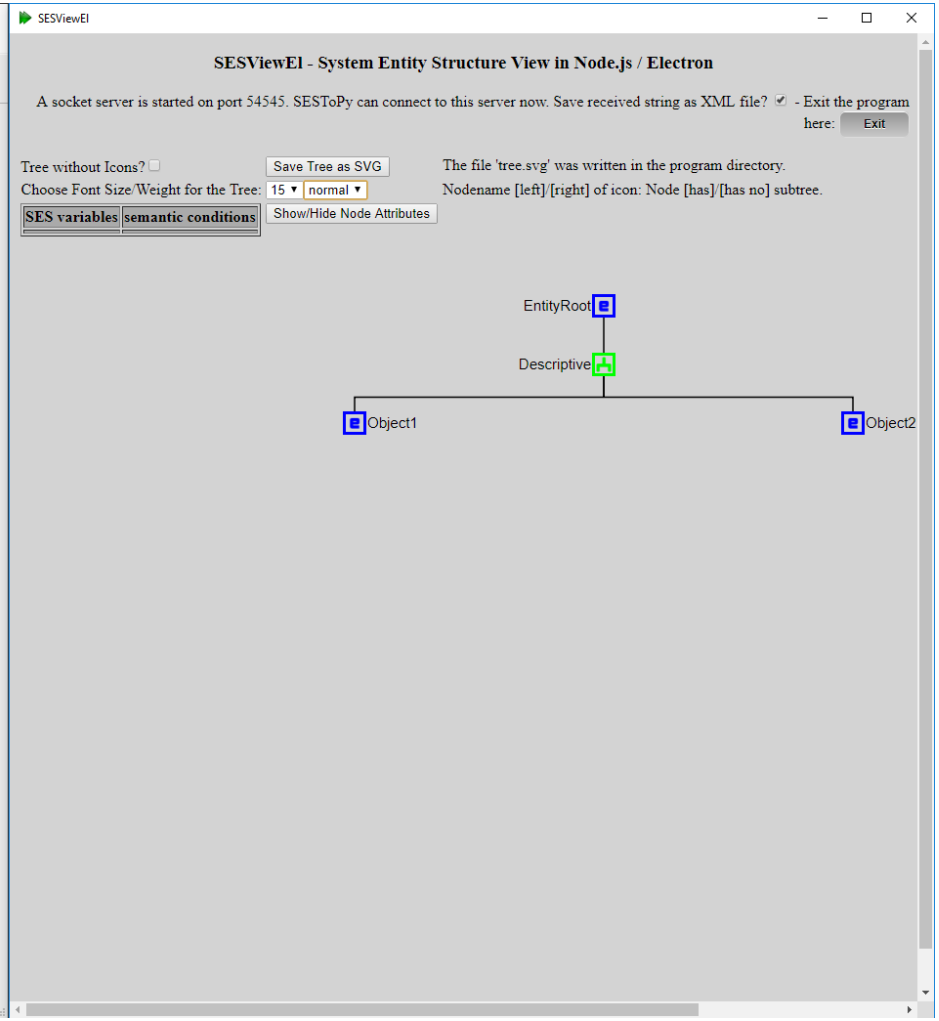
Create SES Tree with SESToPy



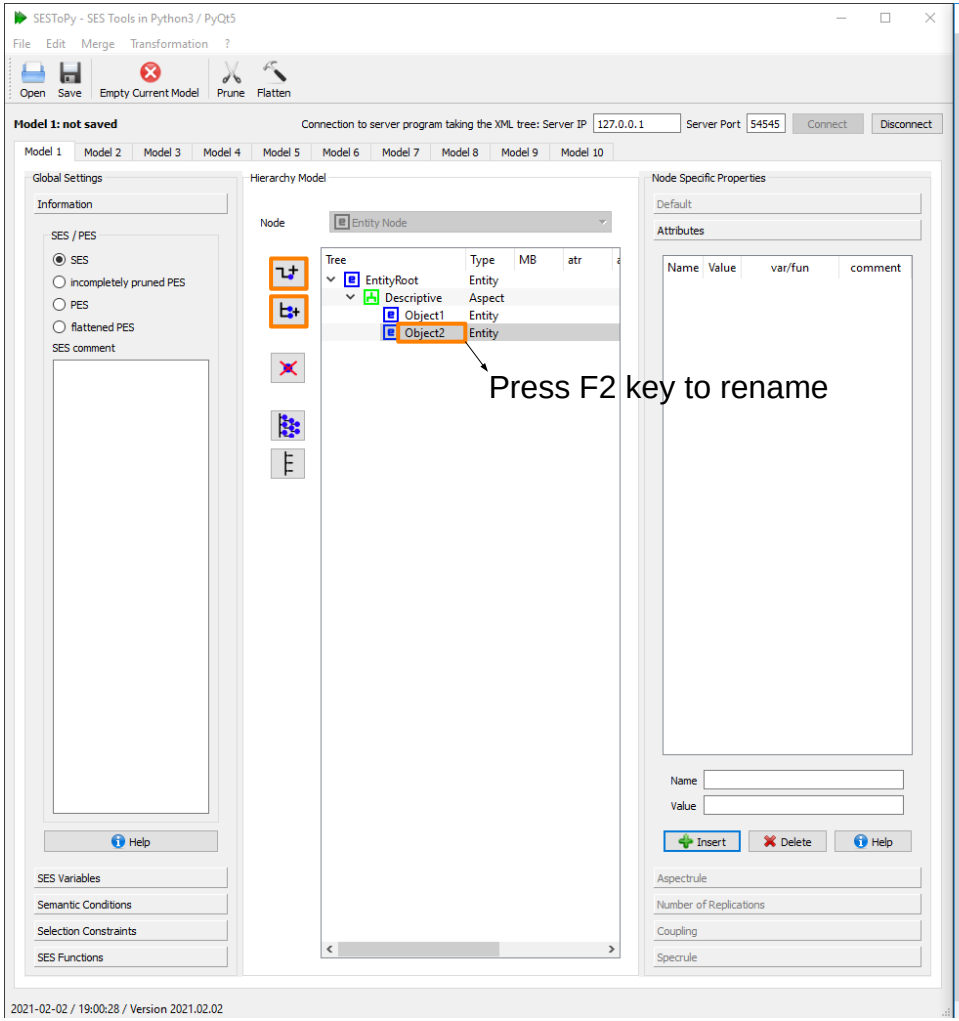


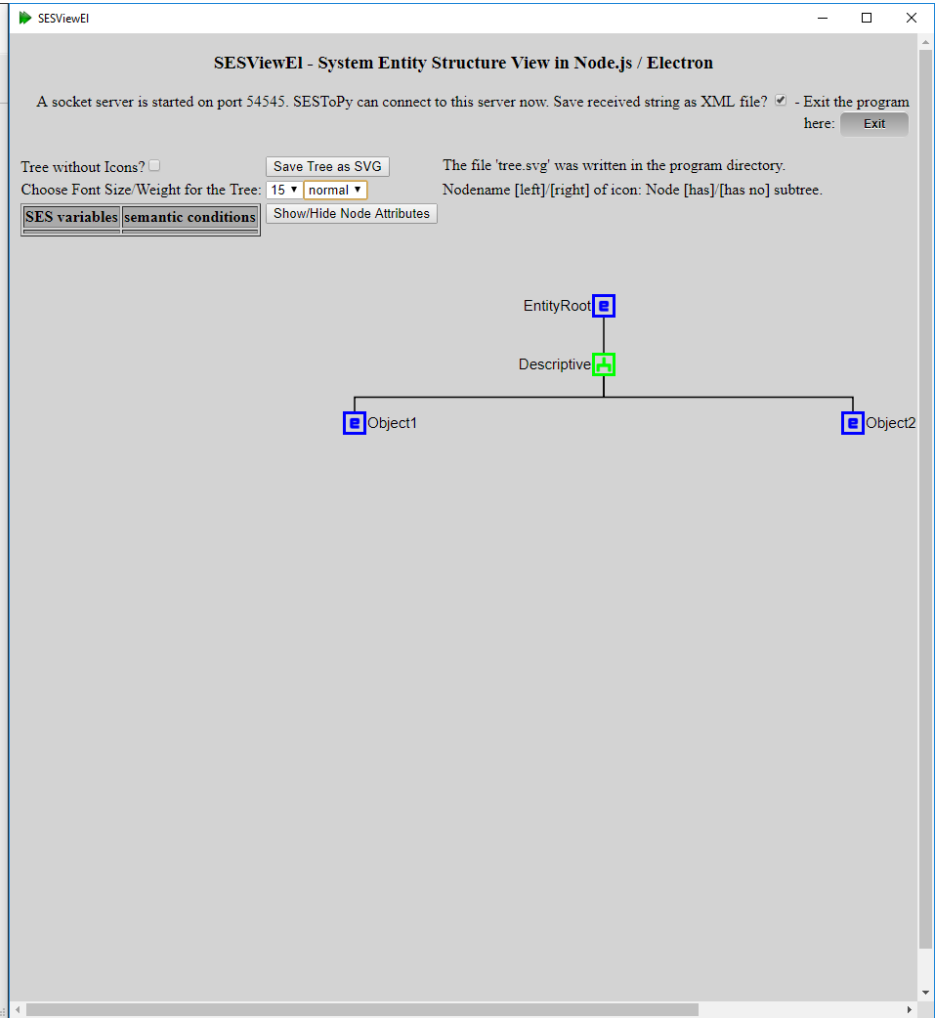
Create SES Tree with SESToPy



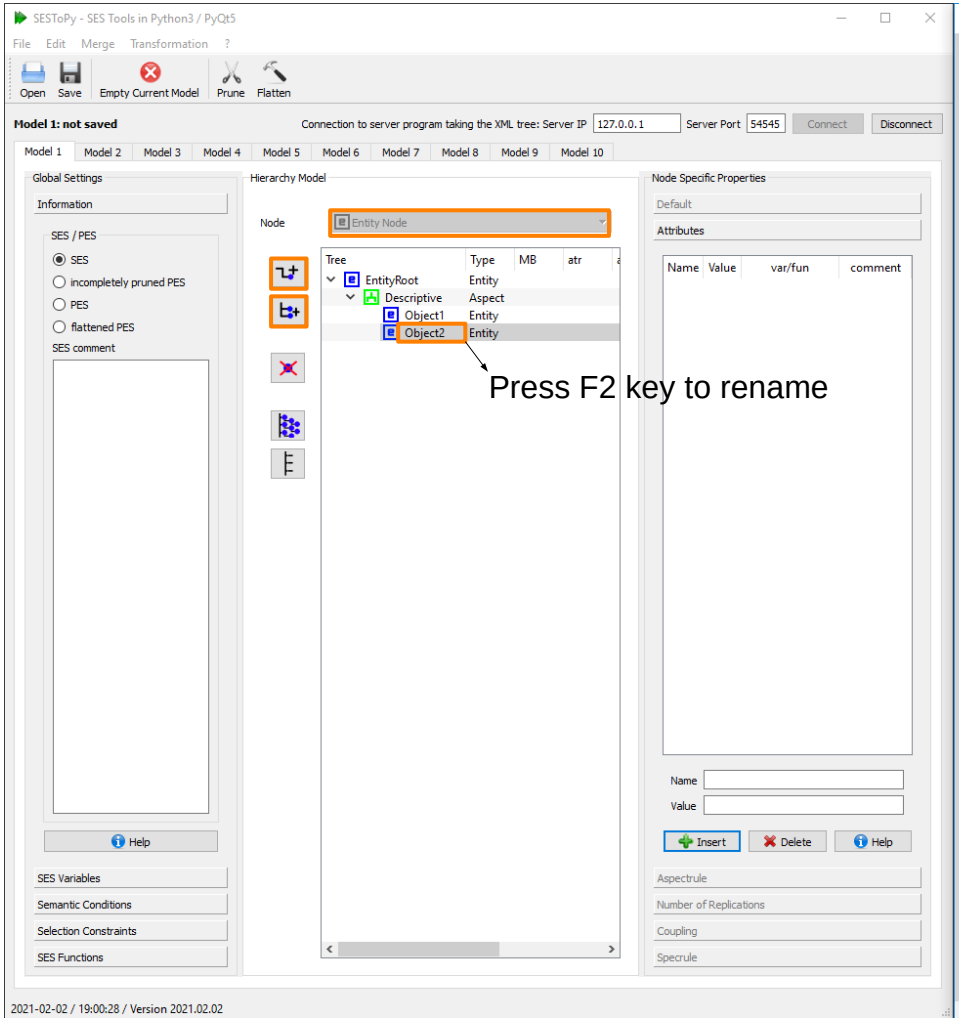


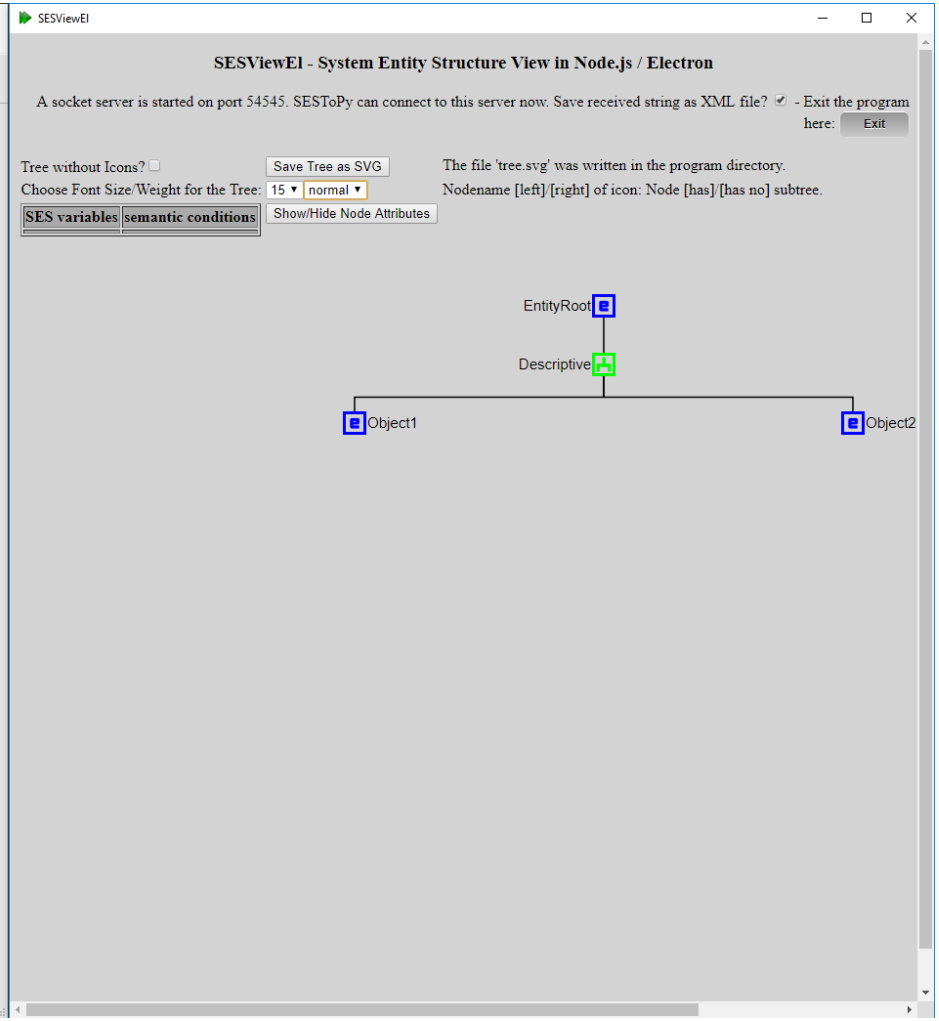
Create SES Tree with SESToPy



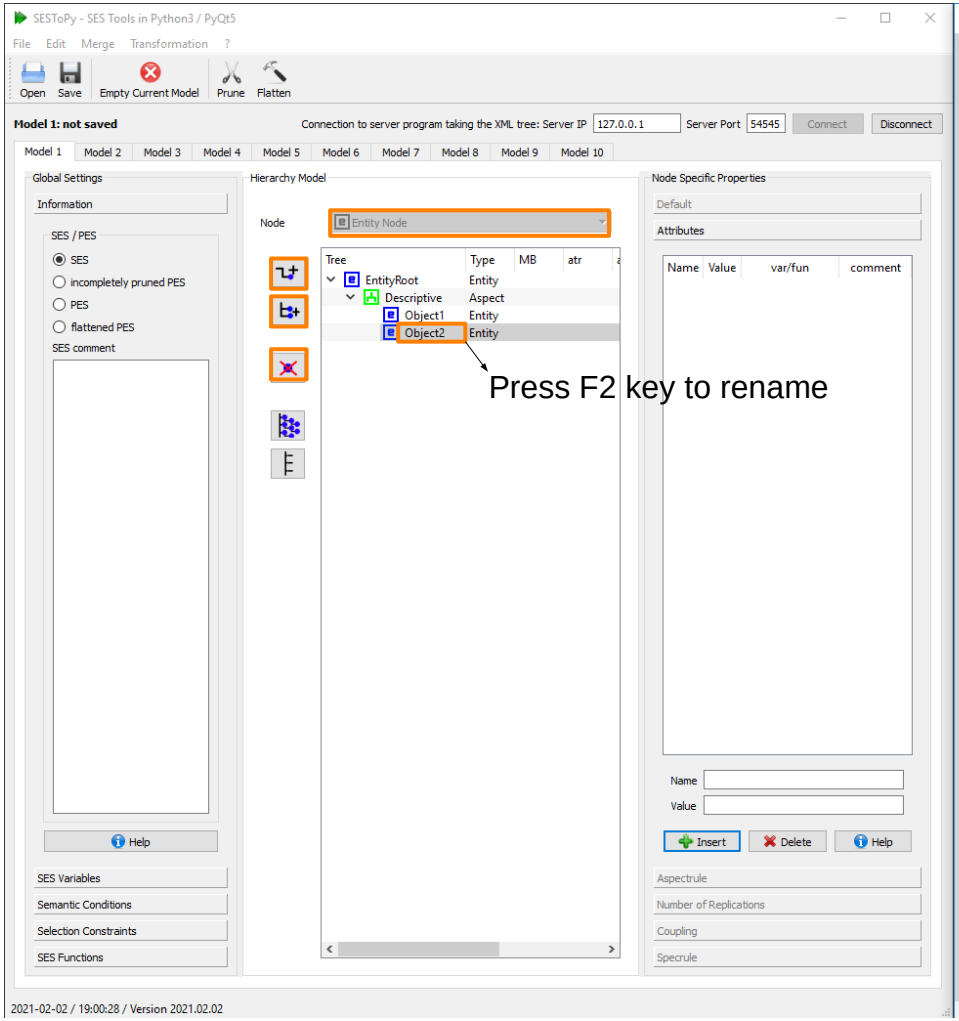


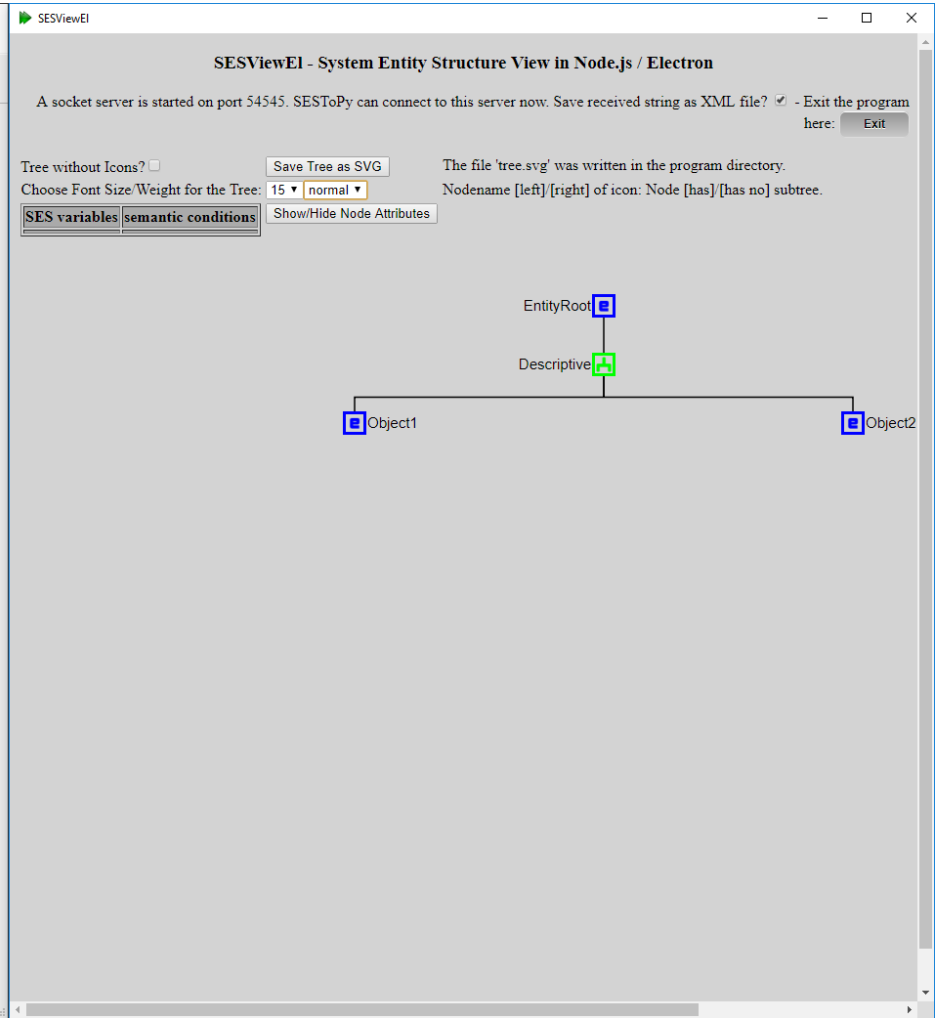
Create SES Tree with SESToPy



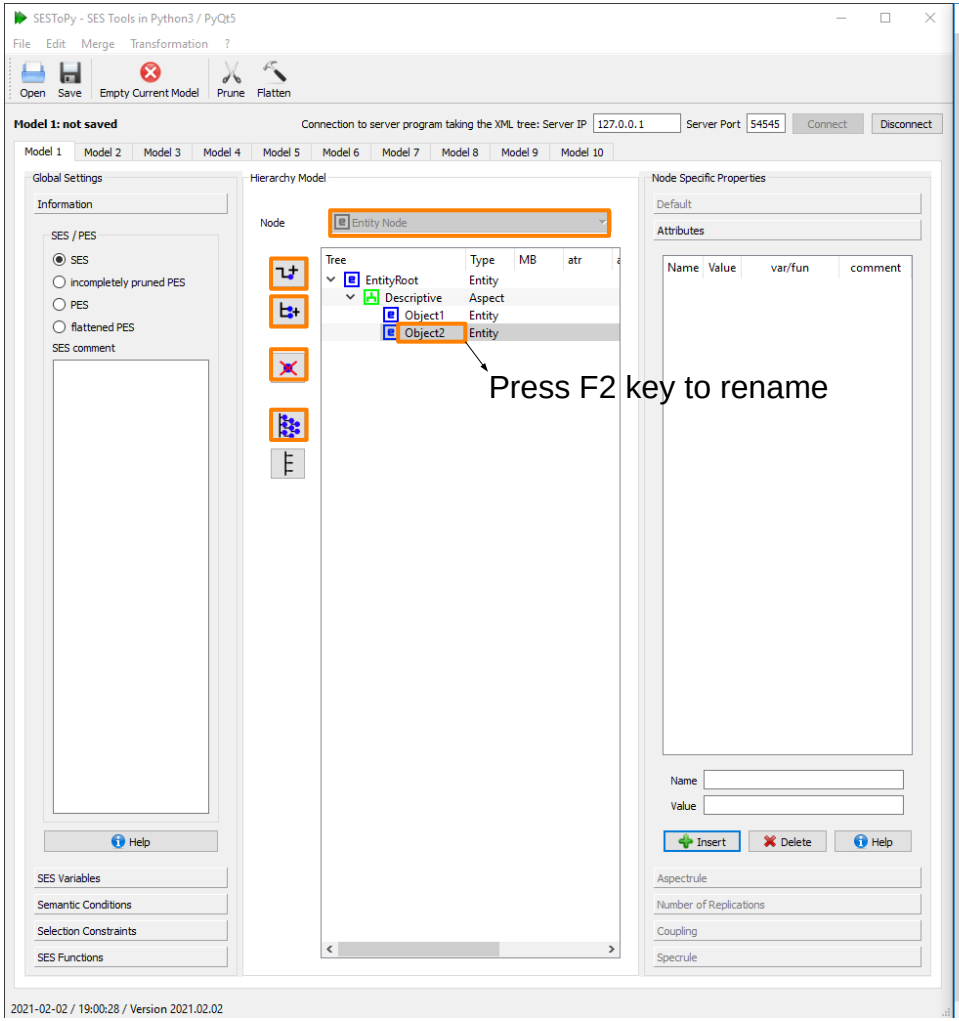


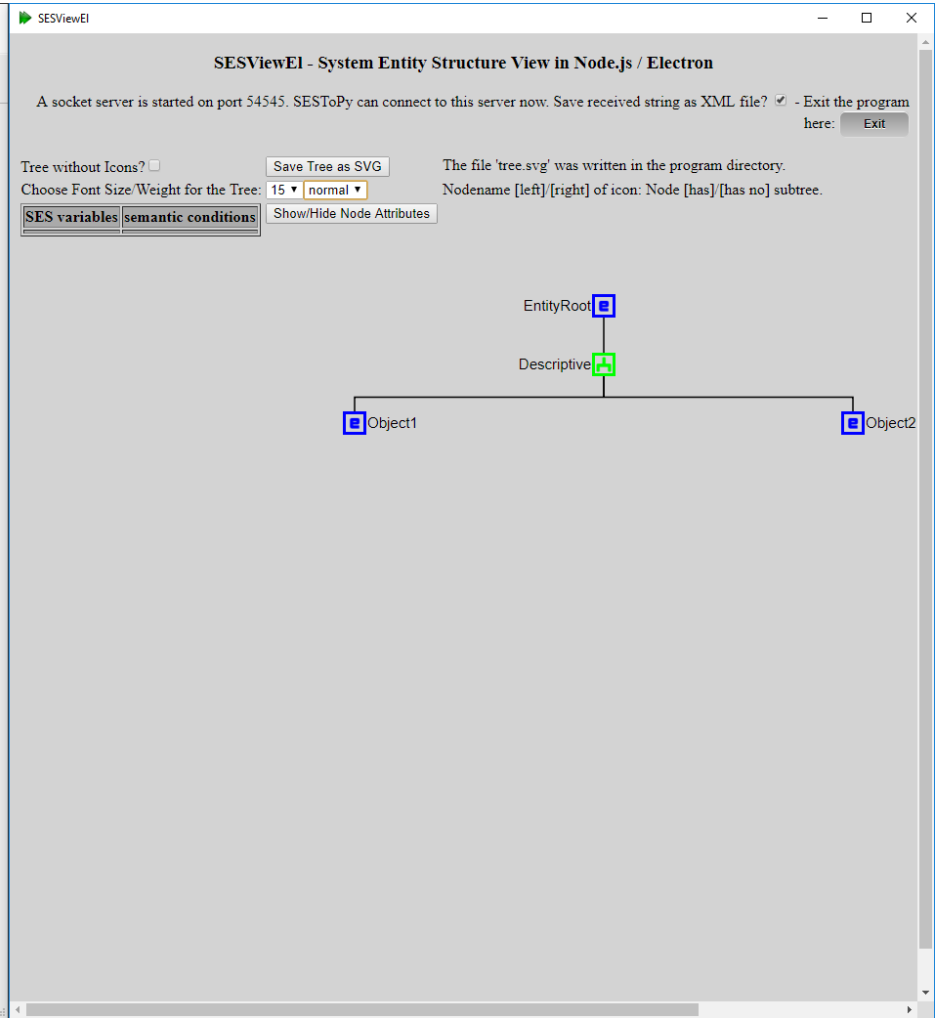
Create SES Tree with SESToPy



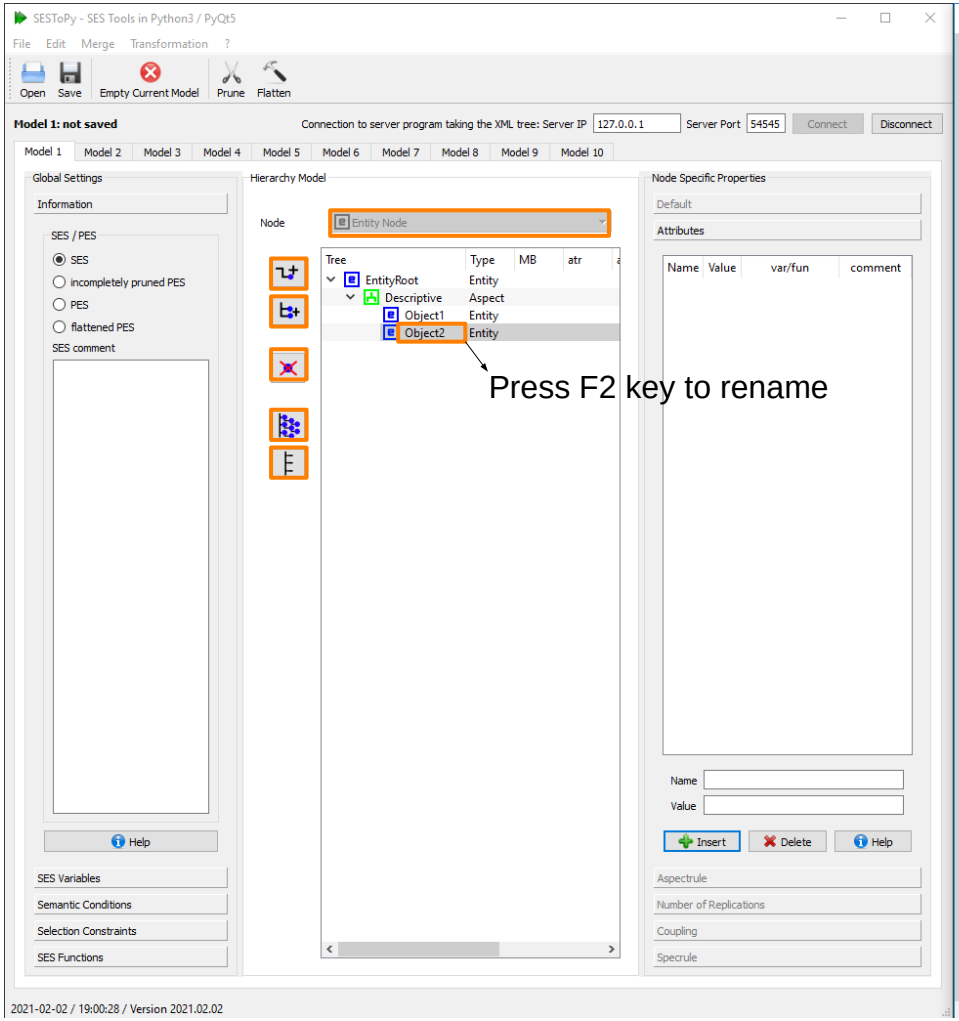


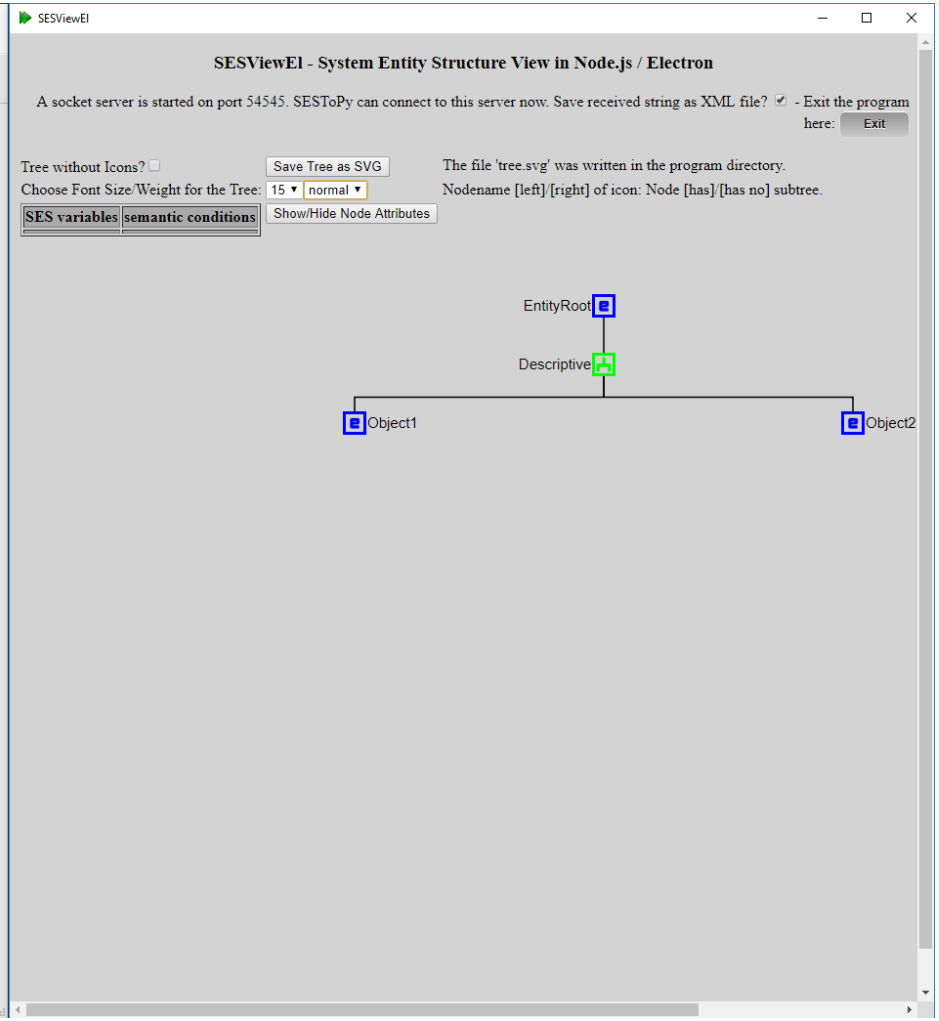
Create SES Tree with SESToPy



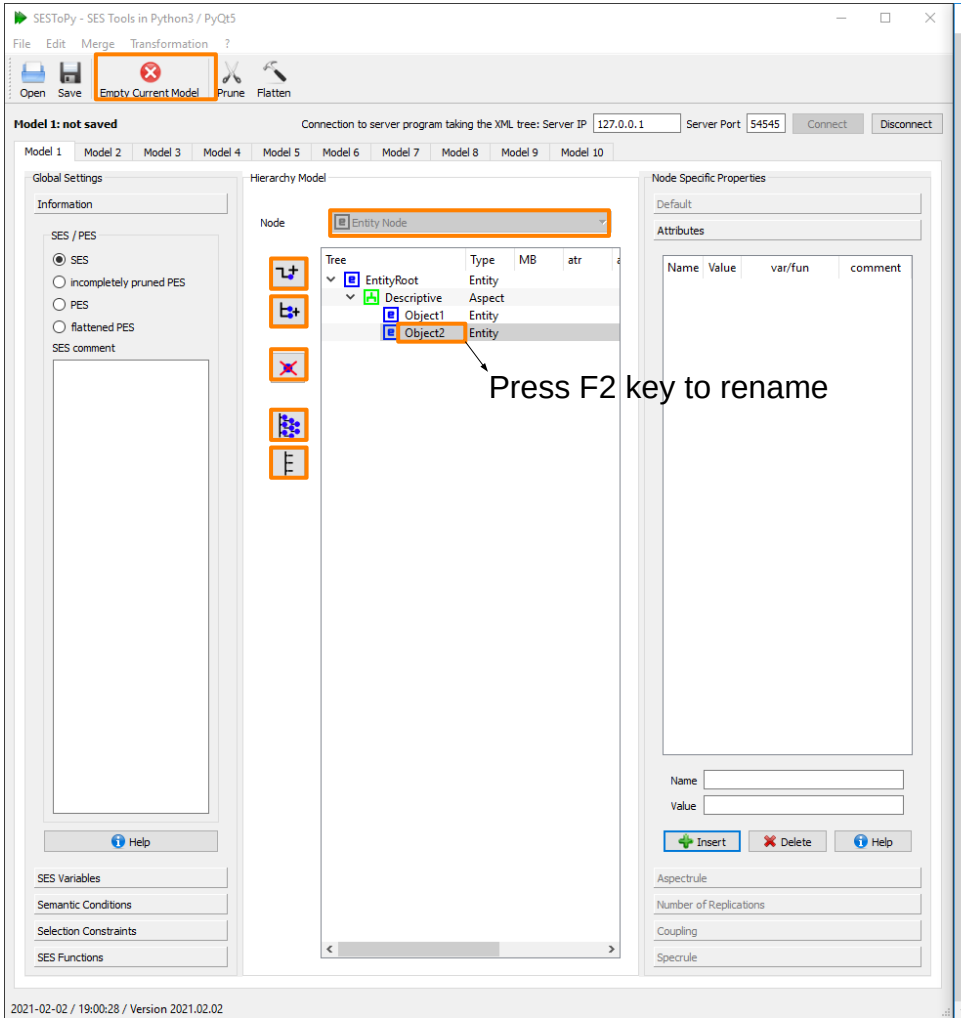


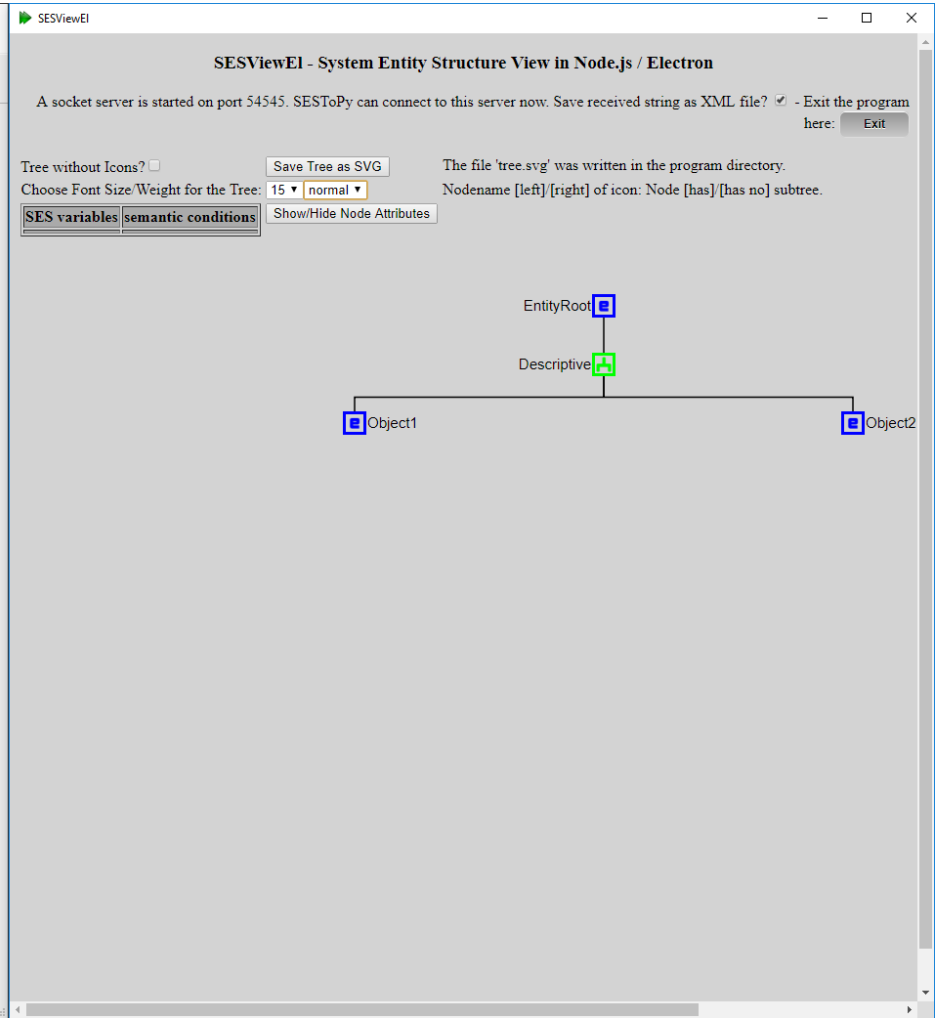
Create SES Tree with SESToPy



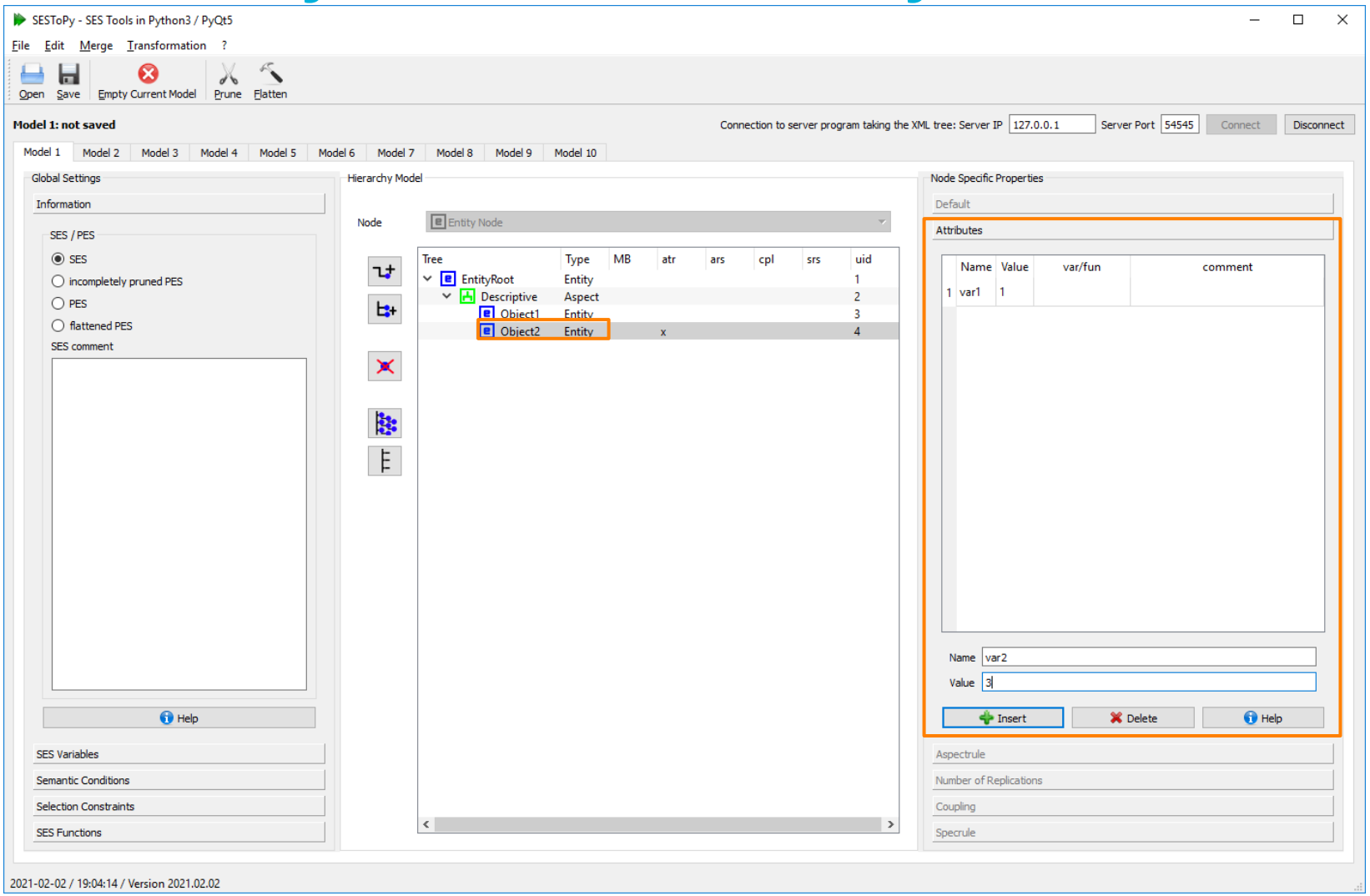


Create SES Tree with SESToPy

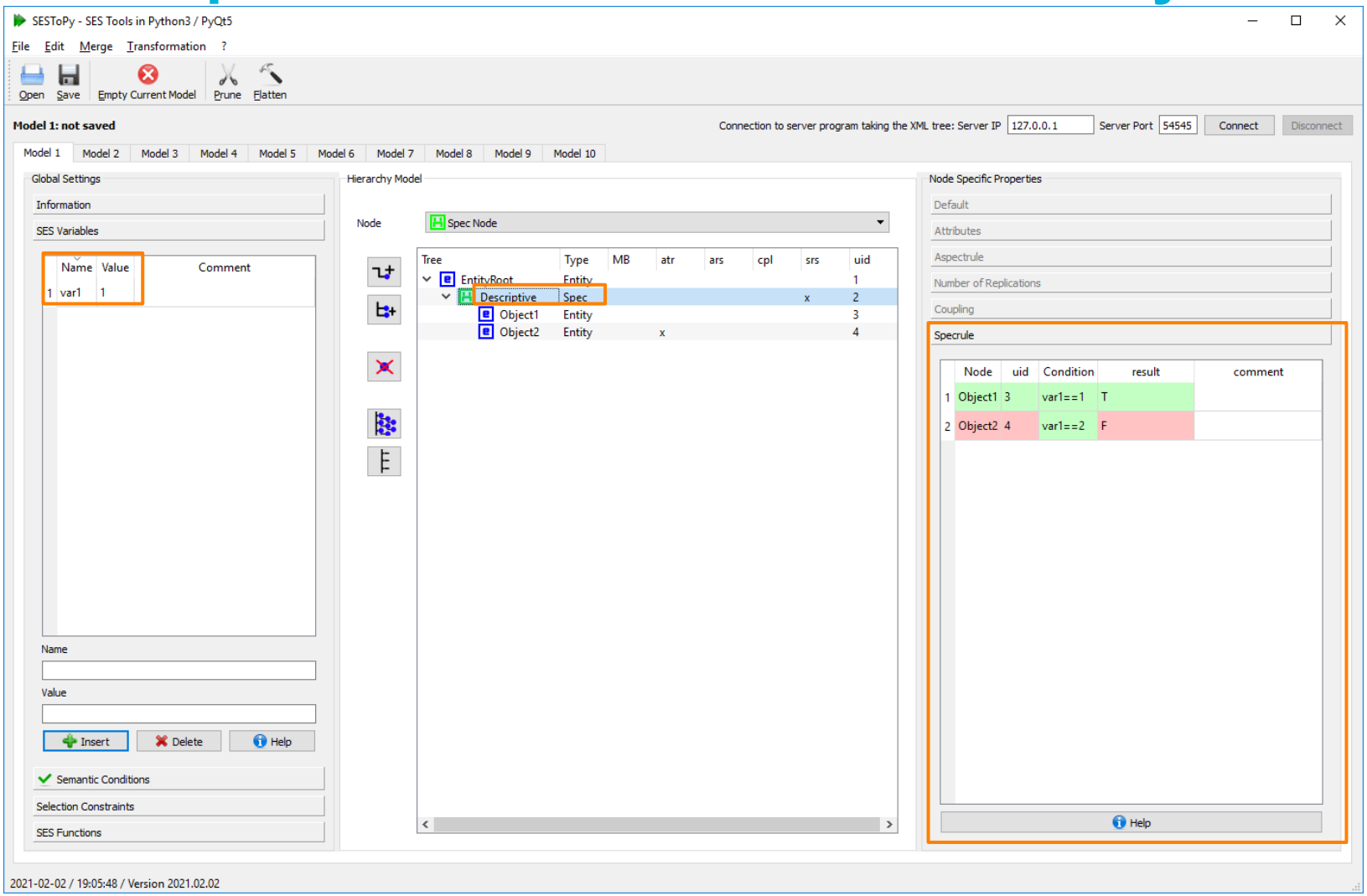




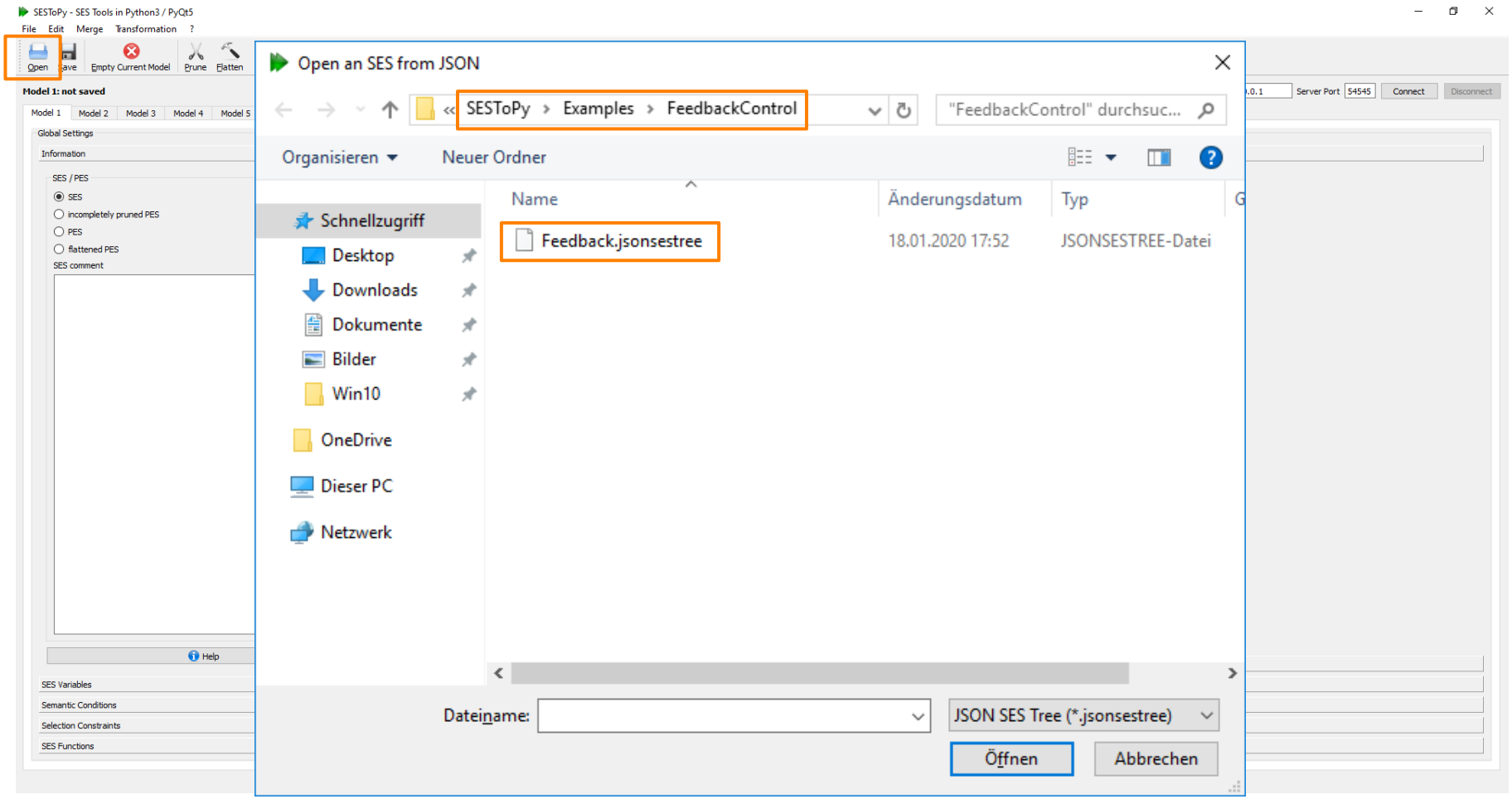
Edit Entity Node with SESToPy



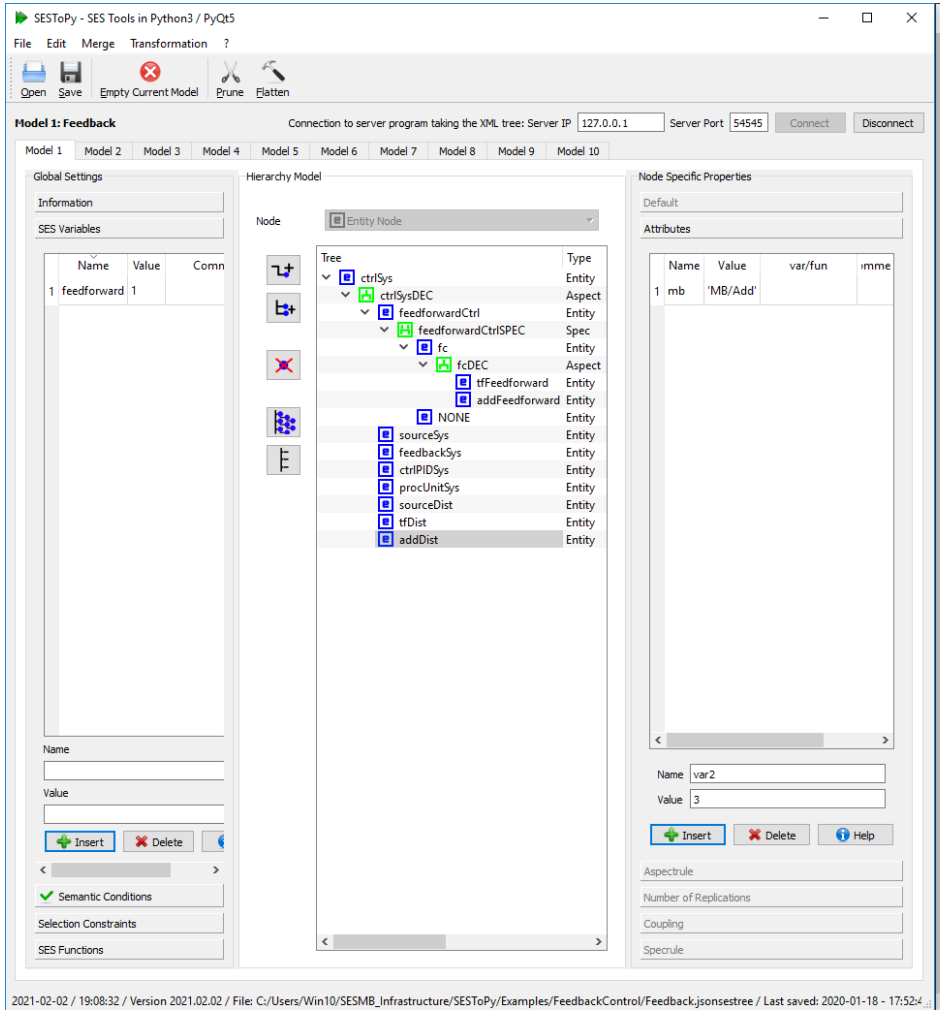
Edit Specialization Node with SESToPy

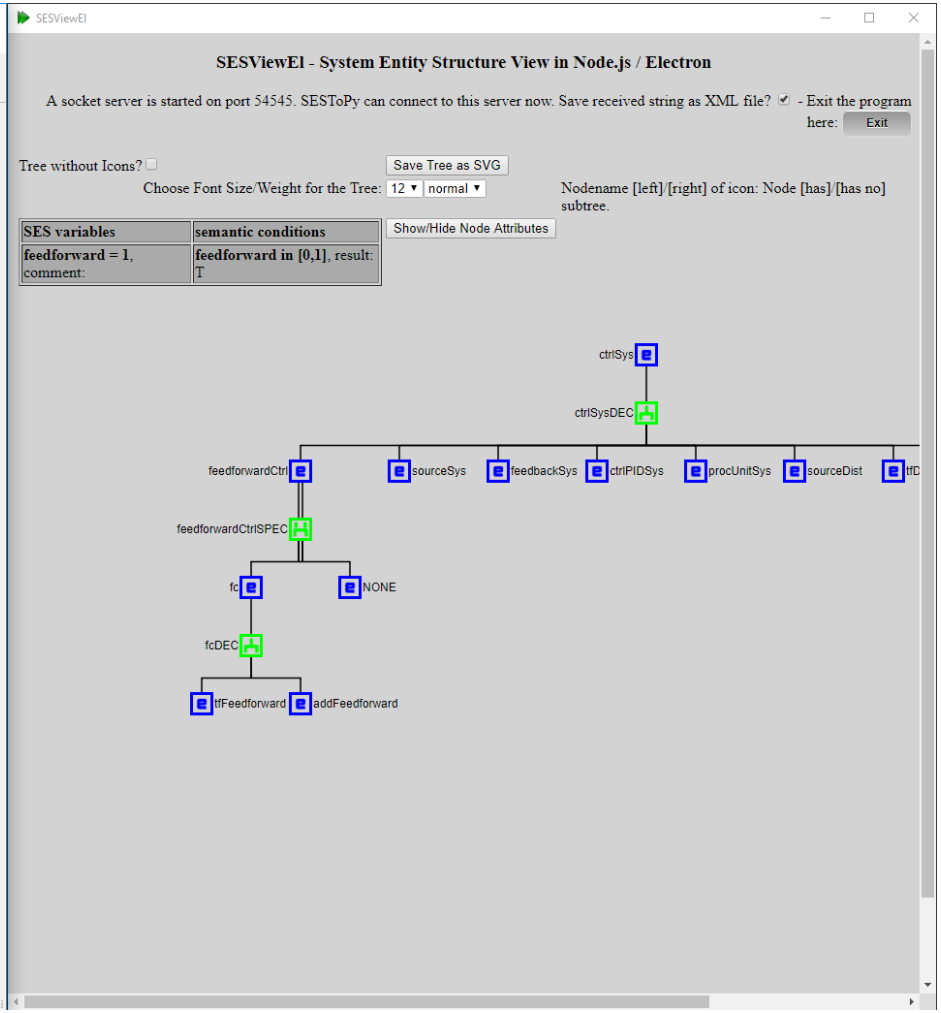


SESToPy Open the Feedback Example

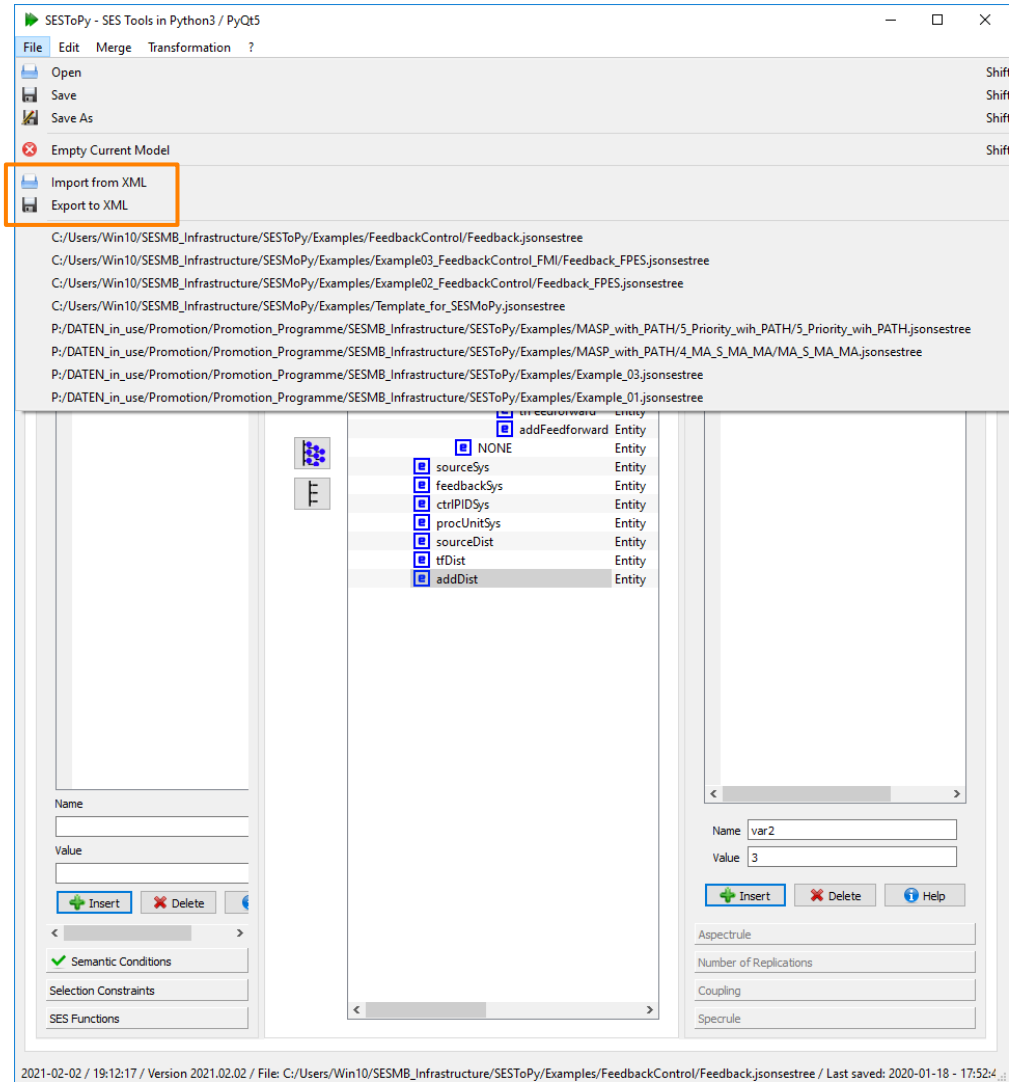


SESToPy Feedback Example

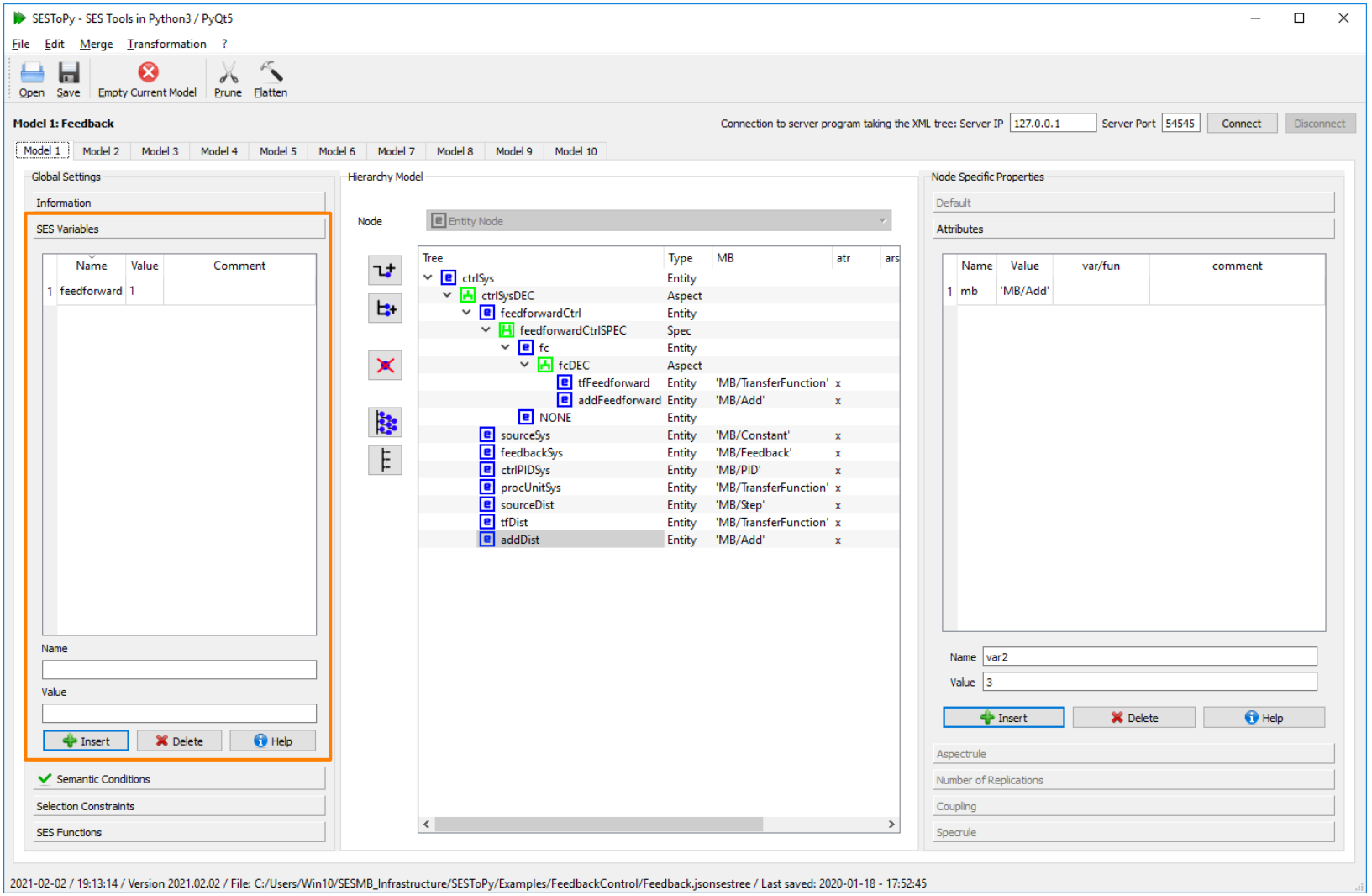




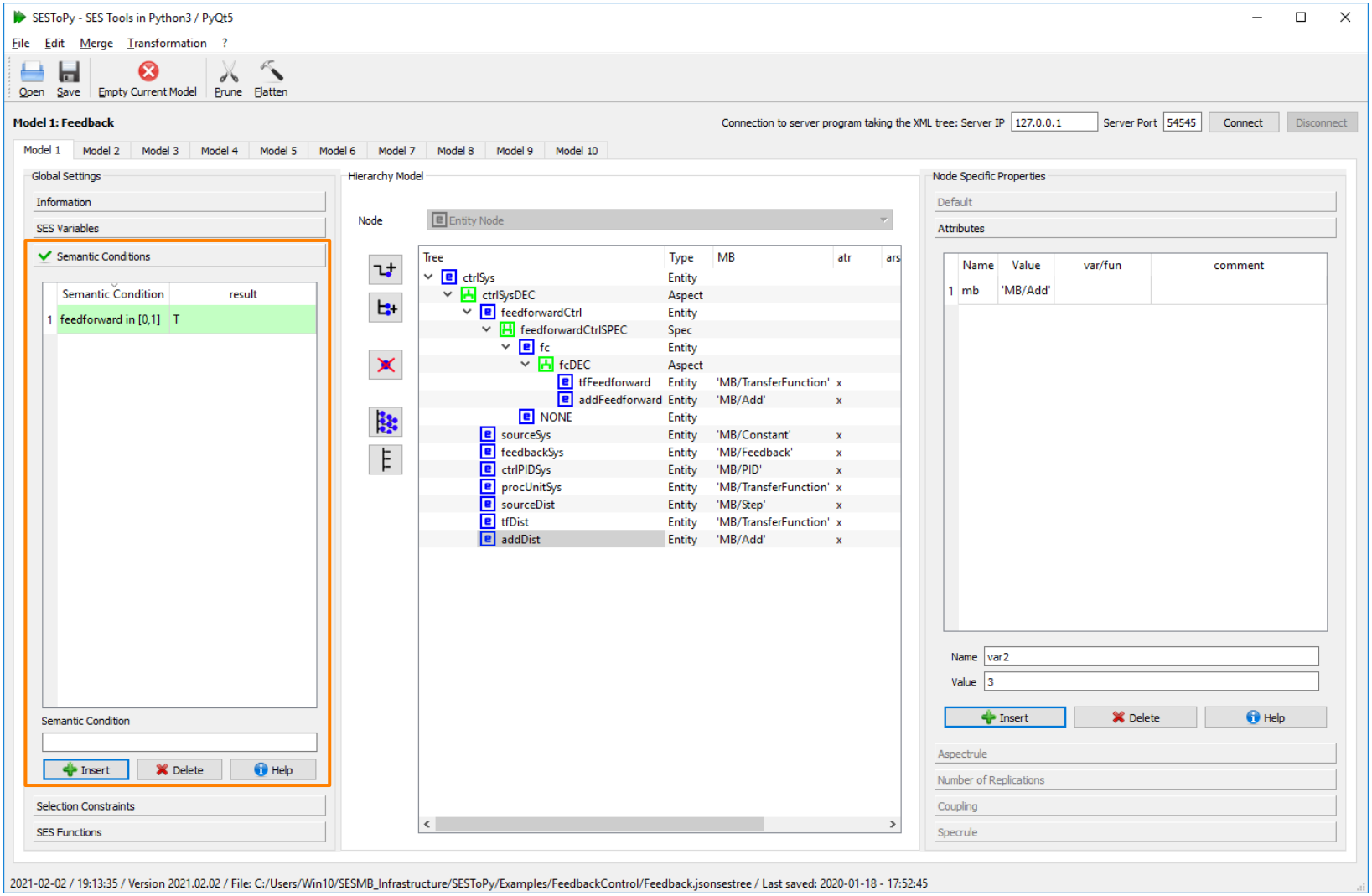
SESToPy XML Export / Import



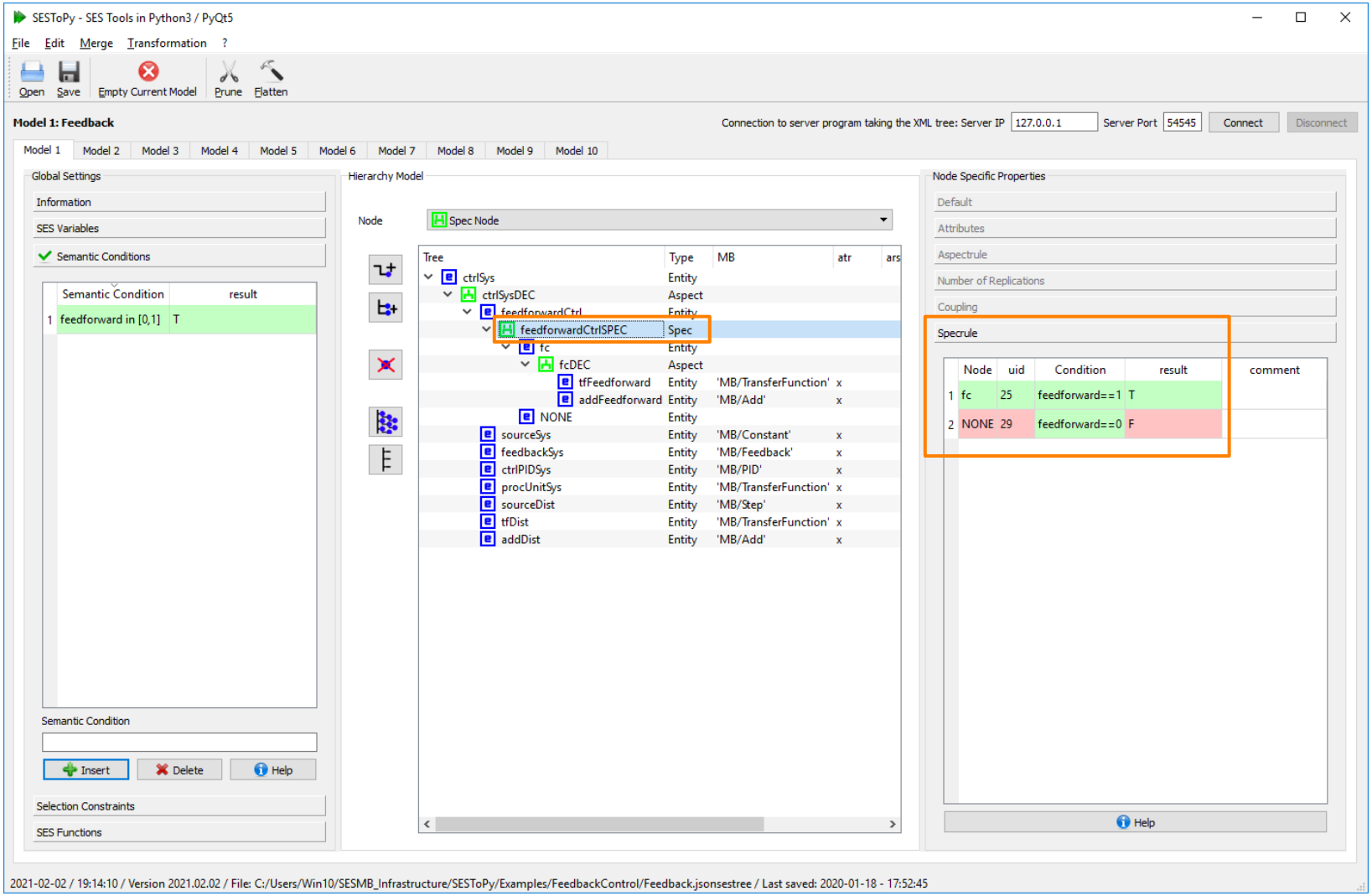
SESToPy SES Variables



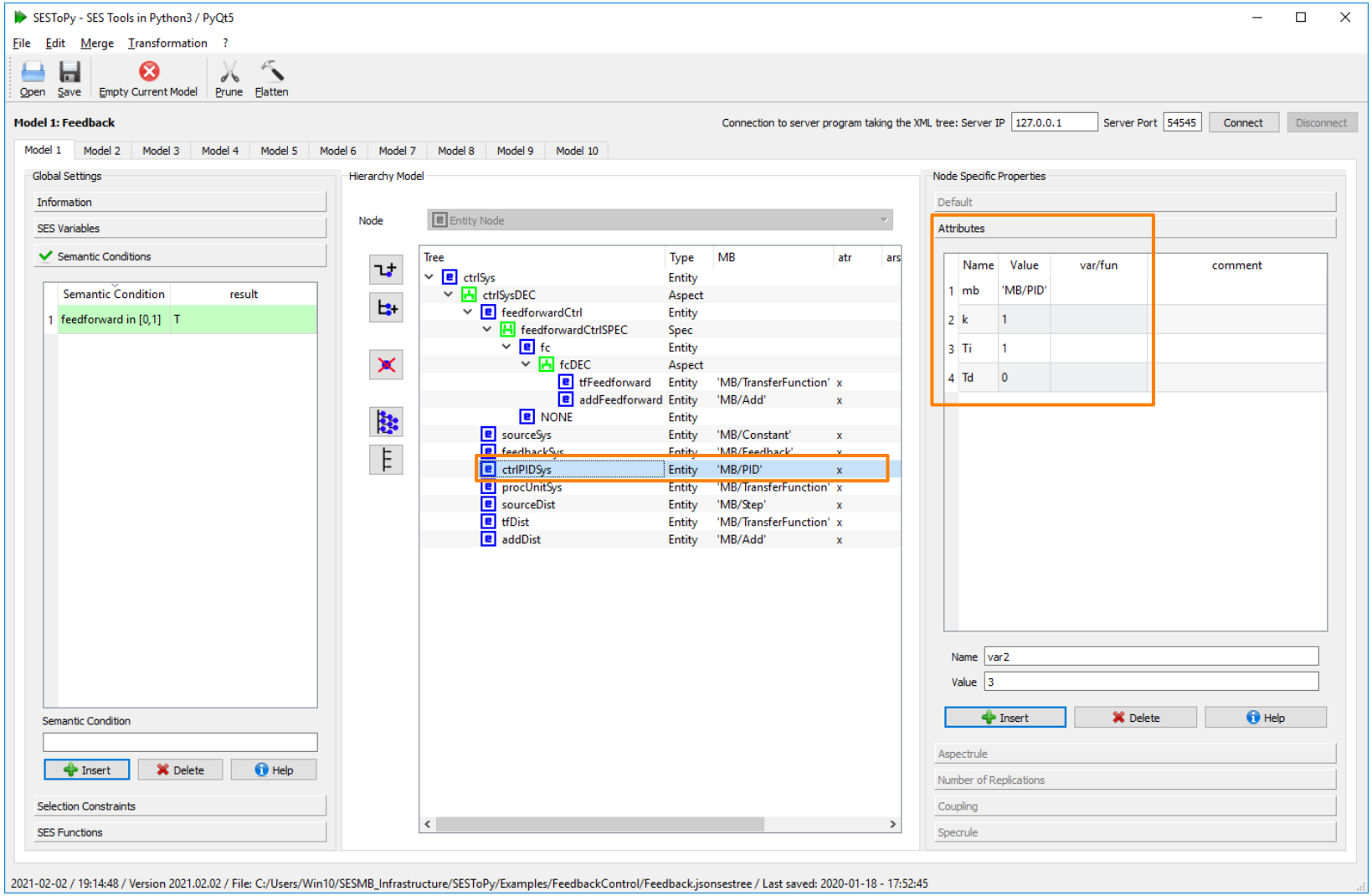
SESToPy Semantic Conditions



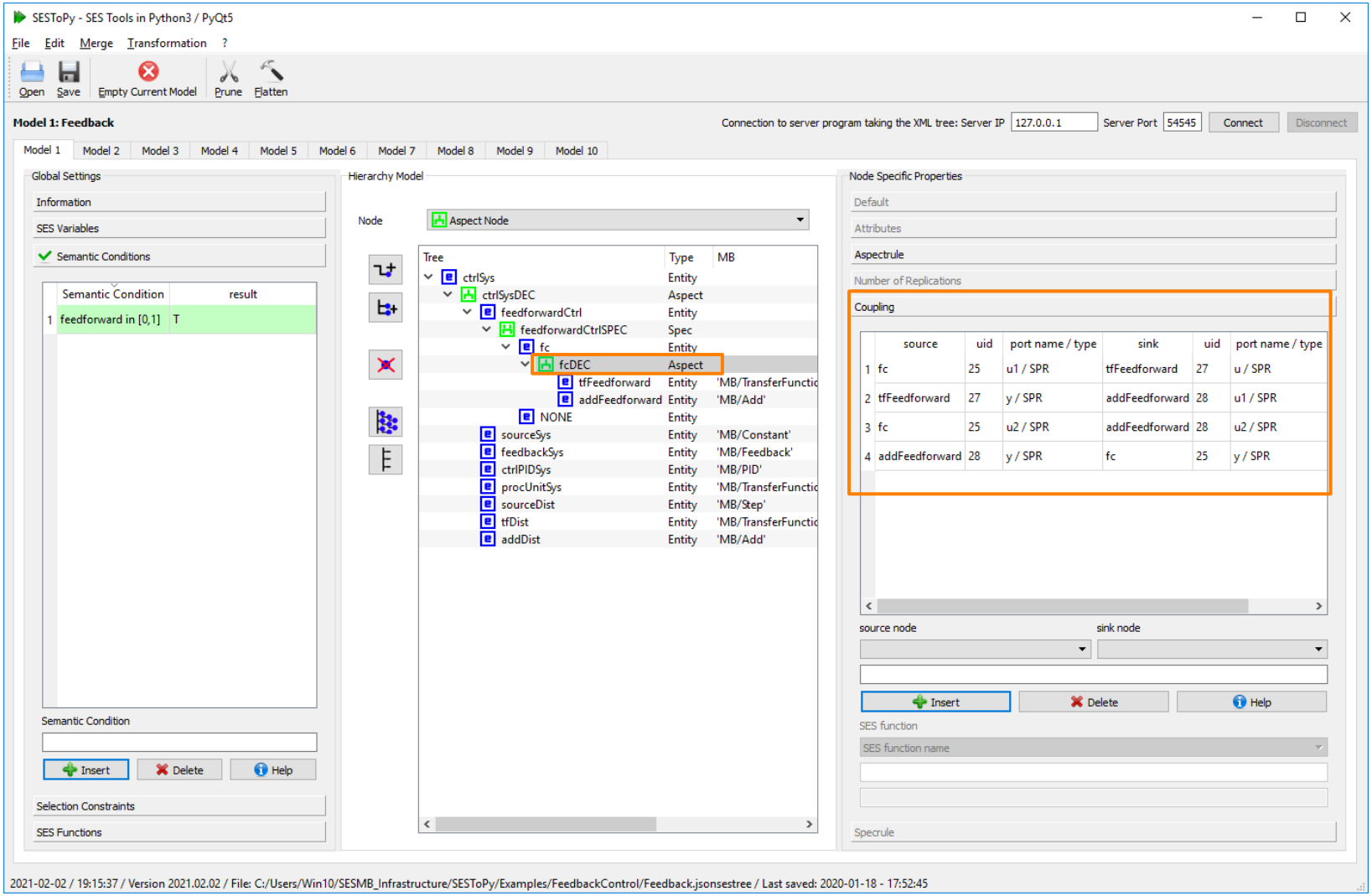
SESToPy Selection Rules (here Specrule)



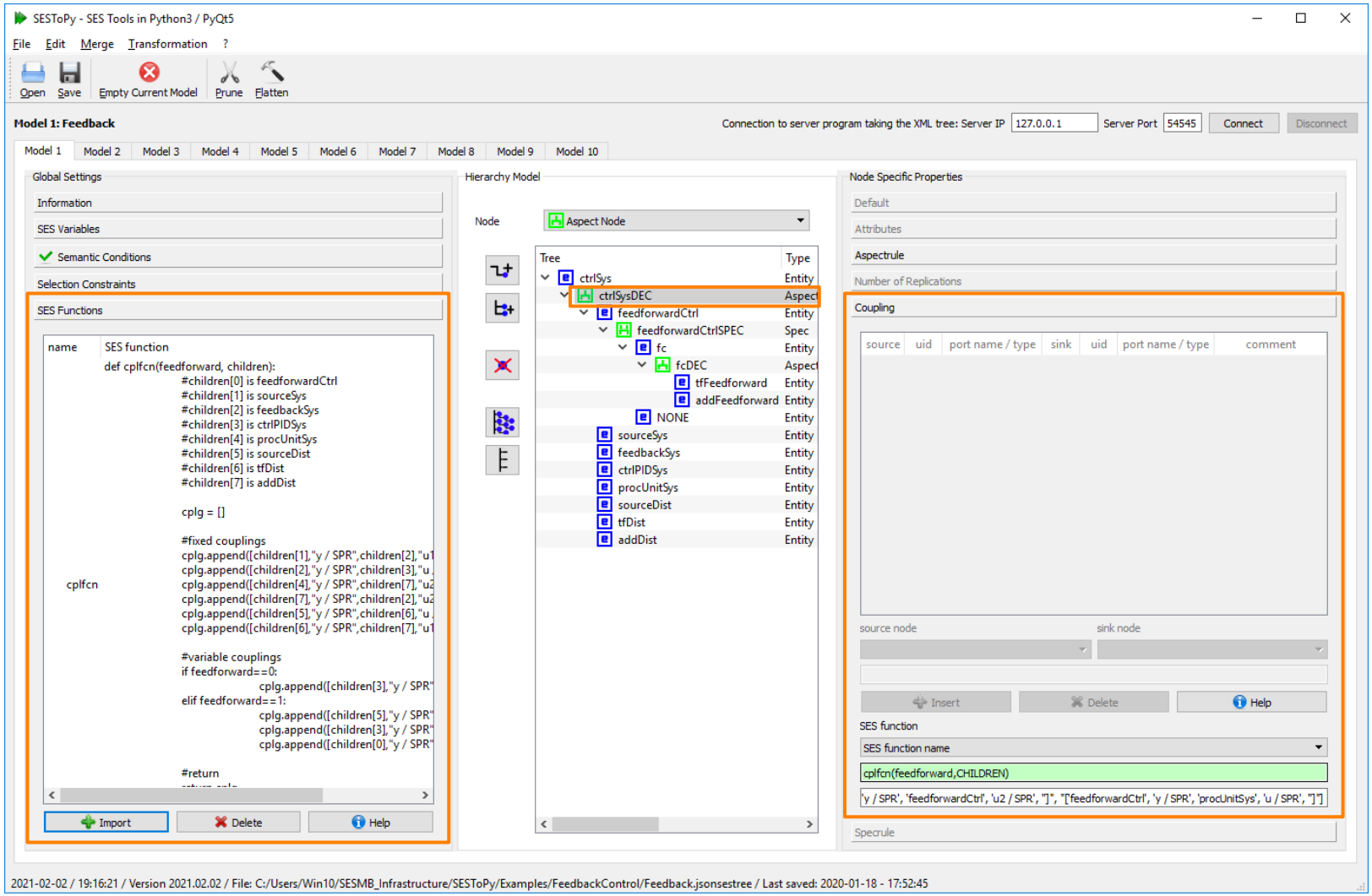
SESToPy Attributes



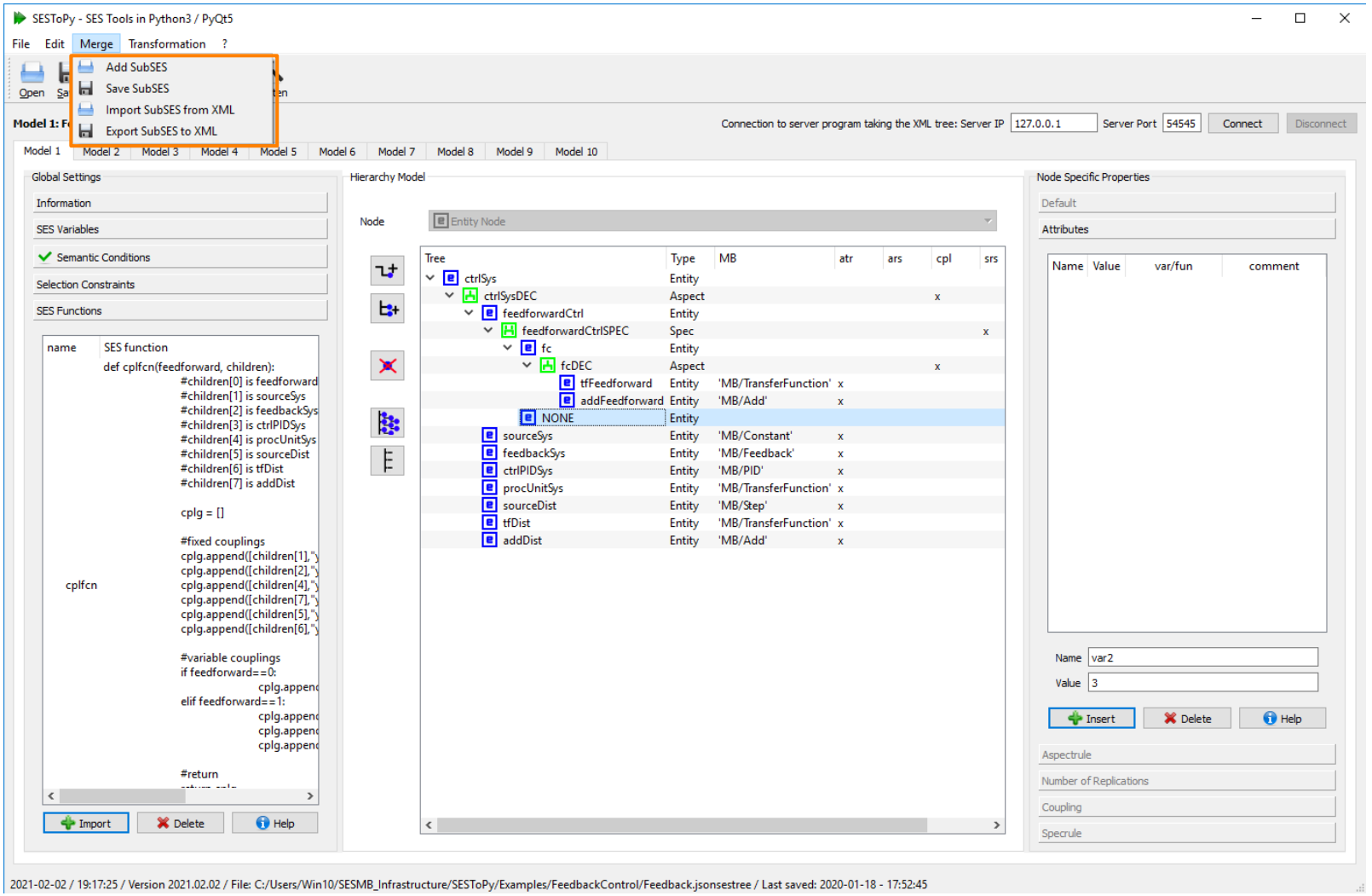
SESToPy Coupling List



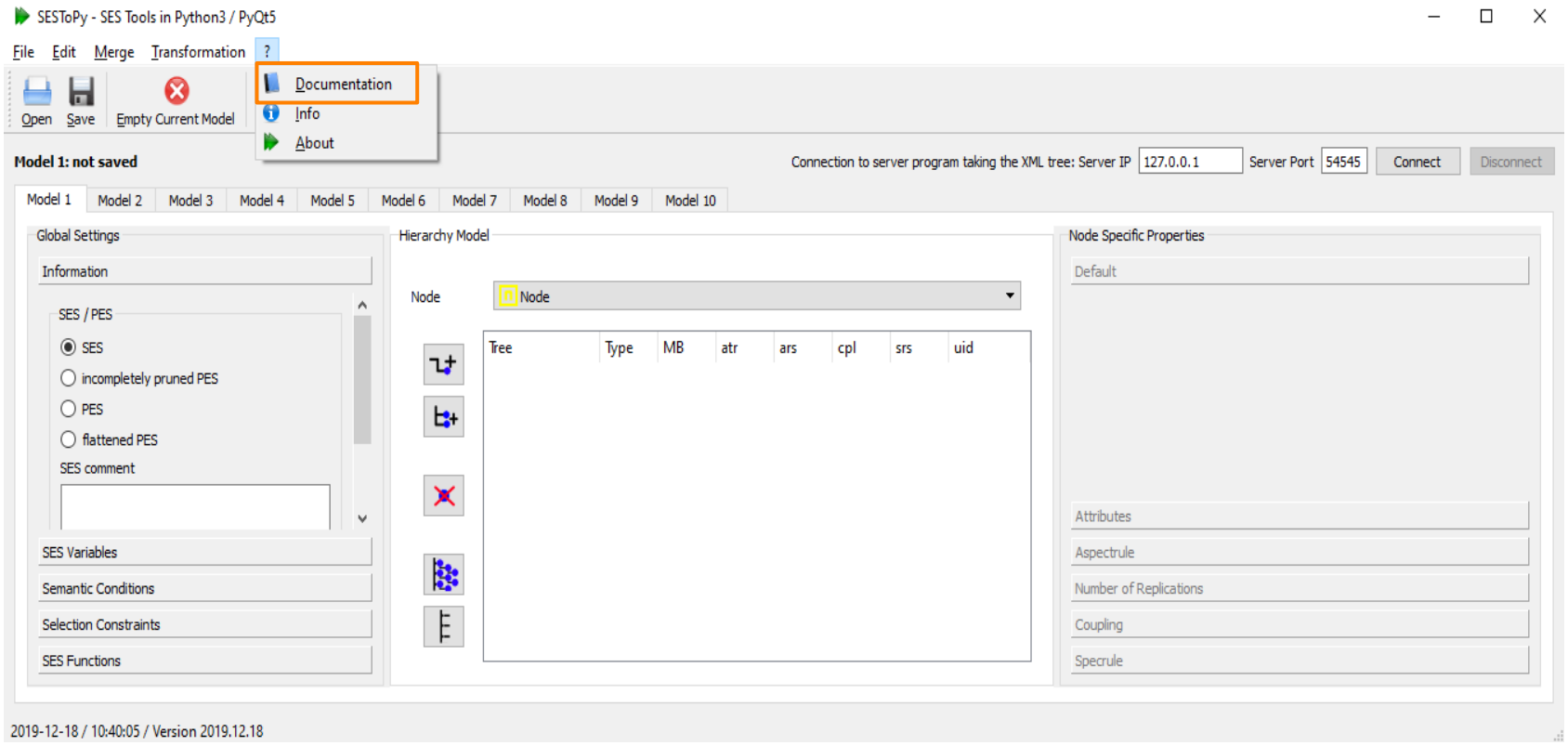
SESToPy SES Function / Coupling Function



SESToPy Merging



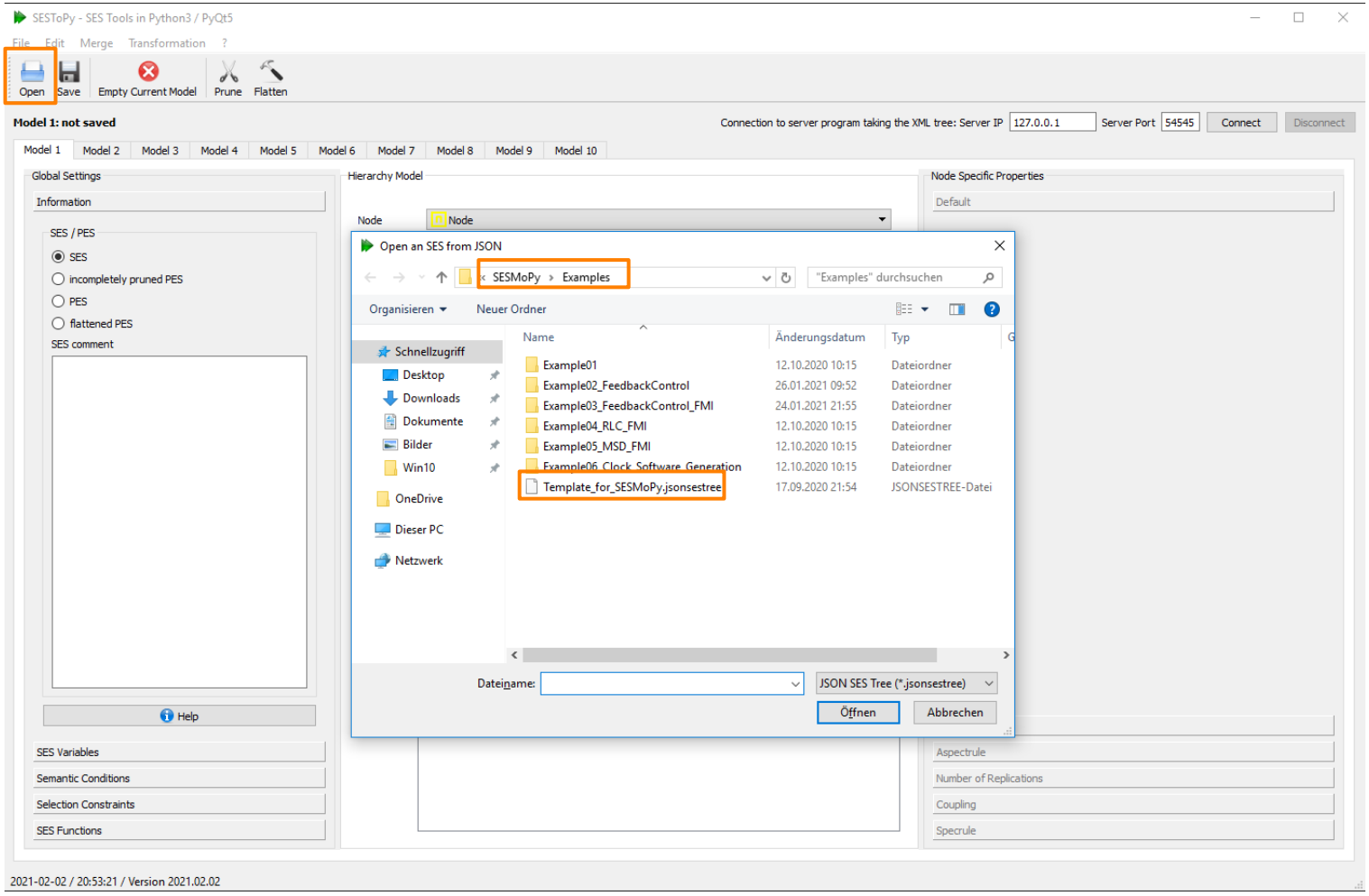
SESToPy Documentation



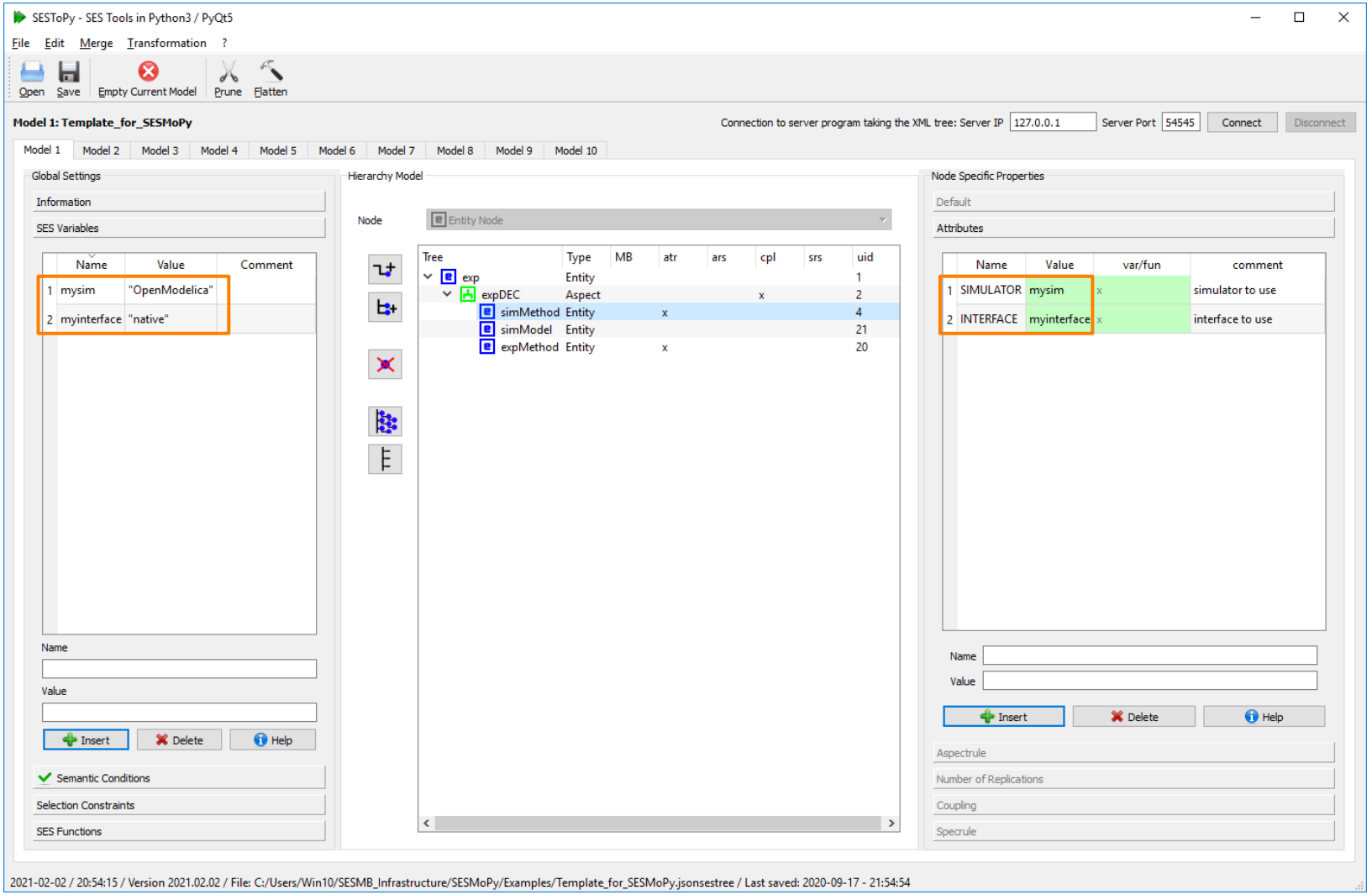
- See the documentation for more information

Chapter 5: Demonstration of SESMoPy Backup Slides

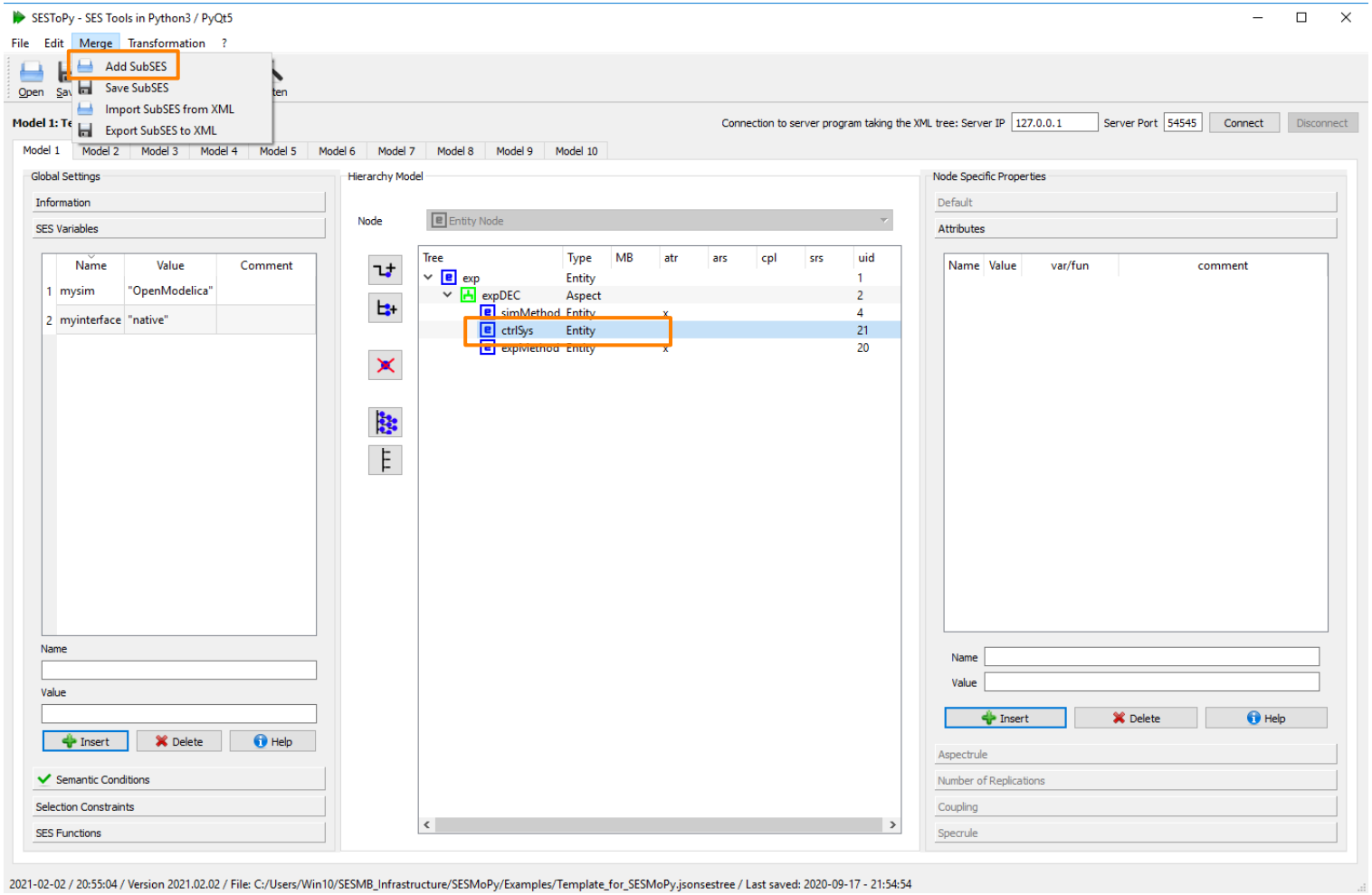
SESMoPy‘s Provisional Experimental Frame Show in SESToPy



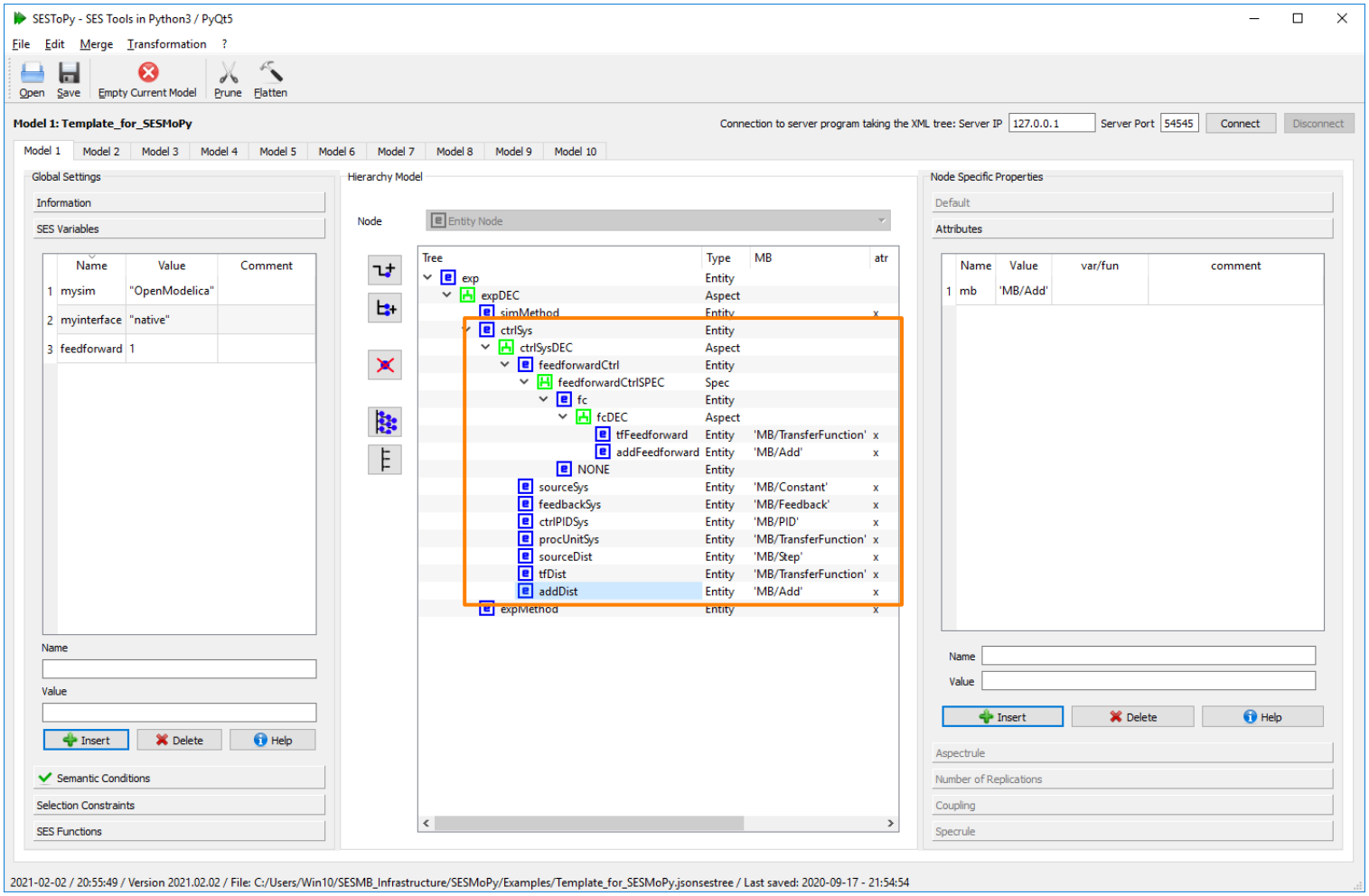
Different Simulators & Interfaces can be set



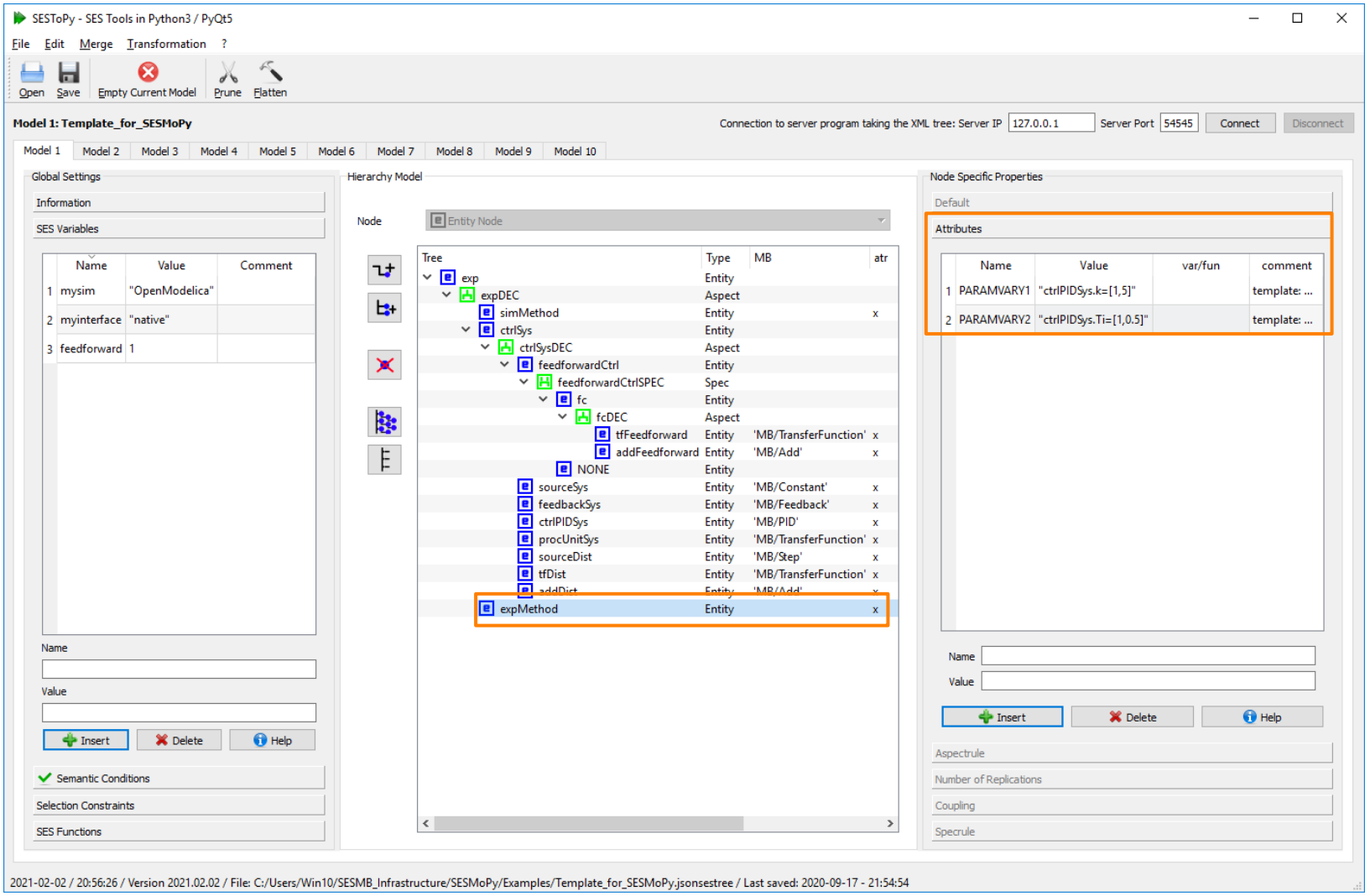
Merge Feedback SES to Provisional Experimental Frame



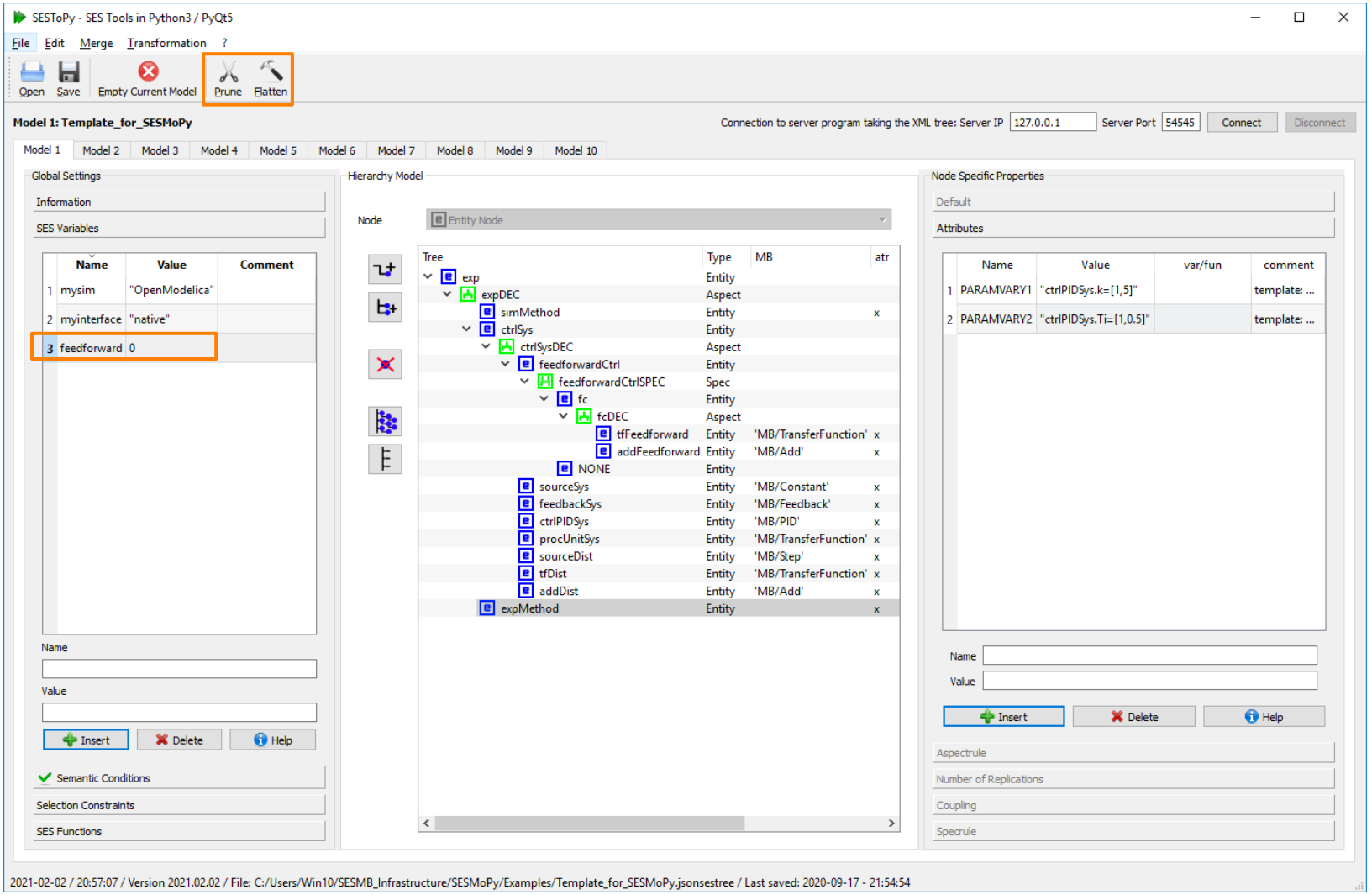
Merge Feedback SES to Provisional Experimental Frame (2)



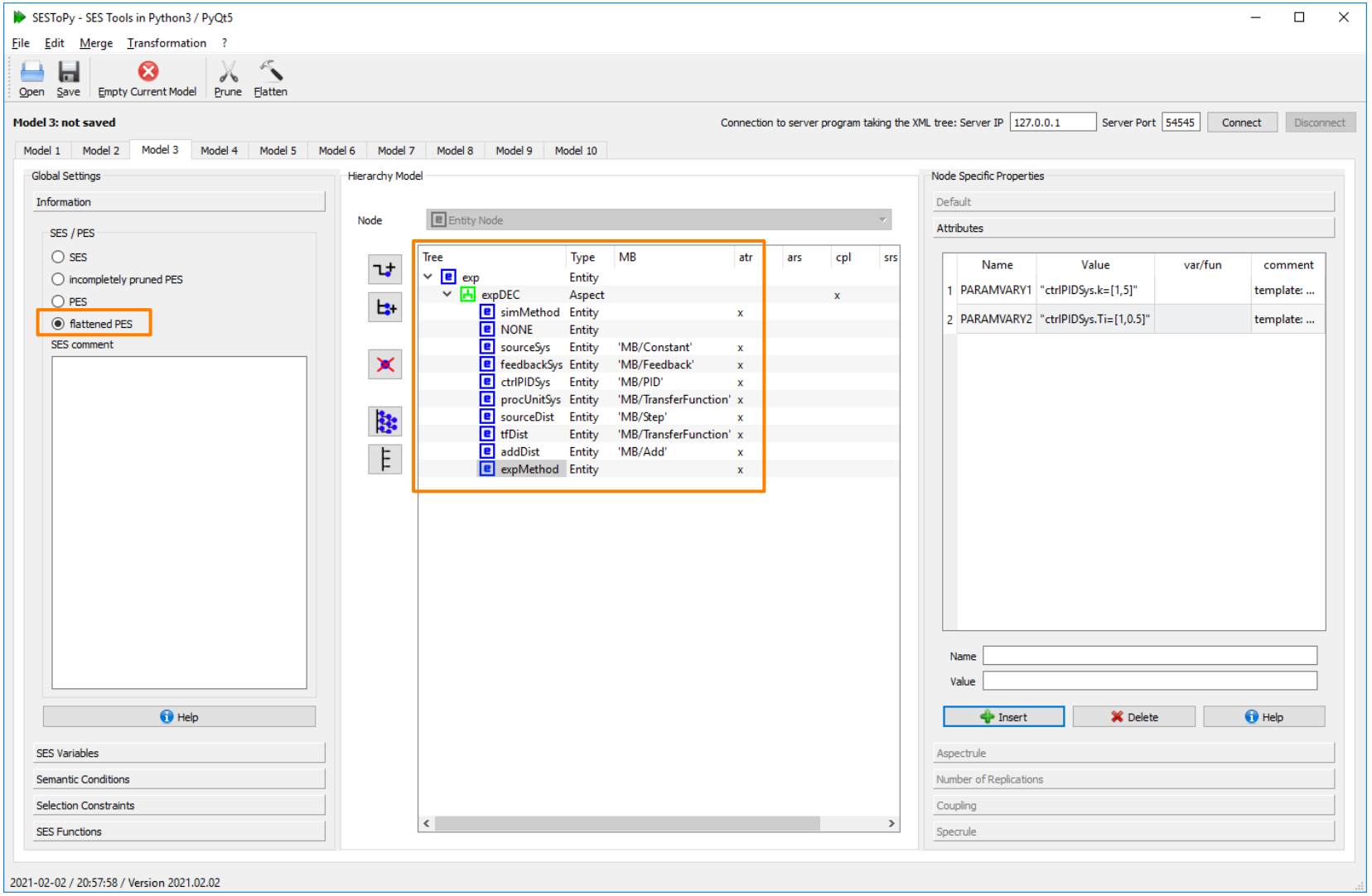
Different Parameters can be set



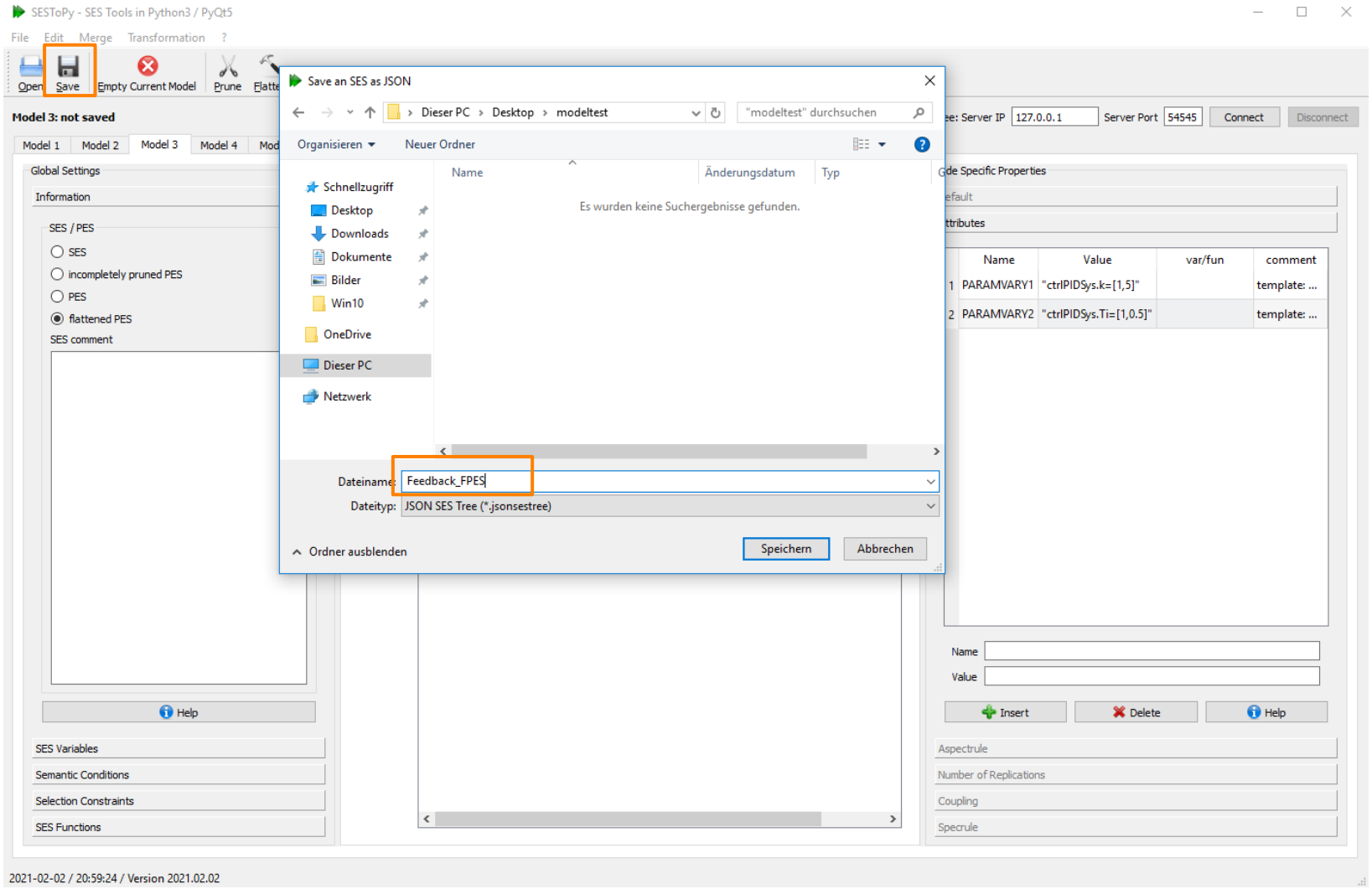
Prune & Flatten for feedforward=0



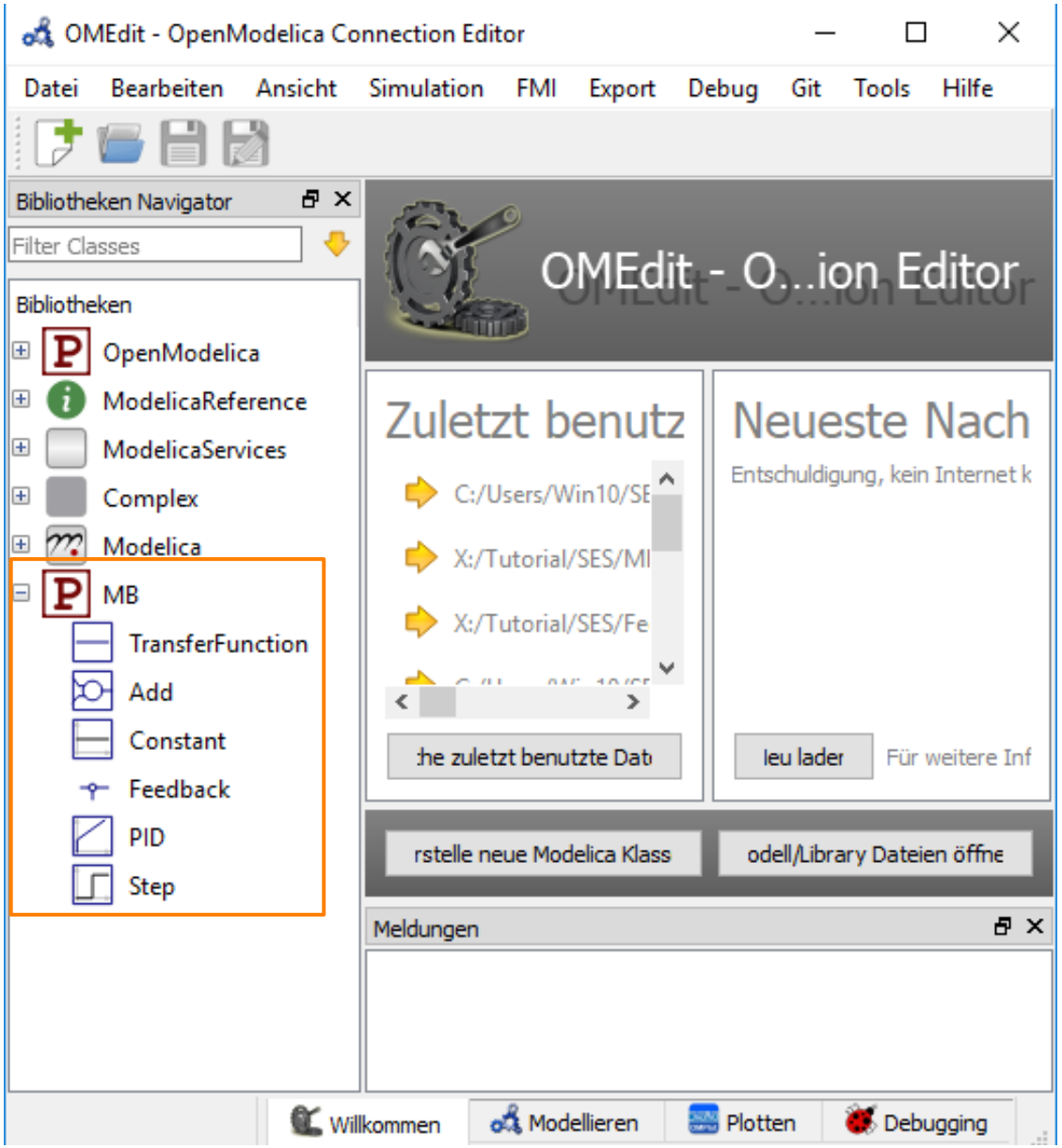
Flattened PES for feedforward=0



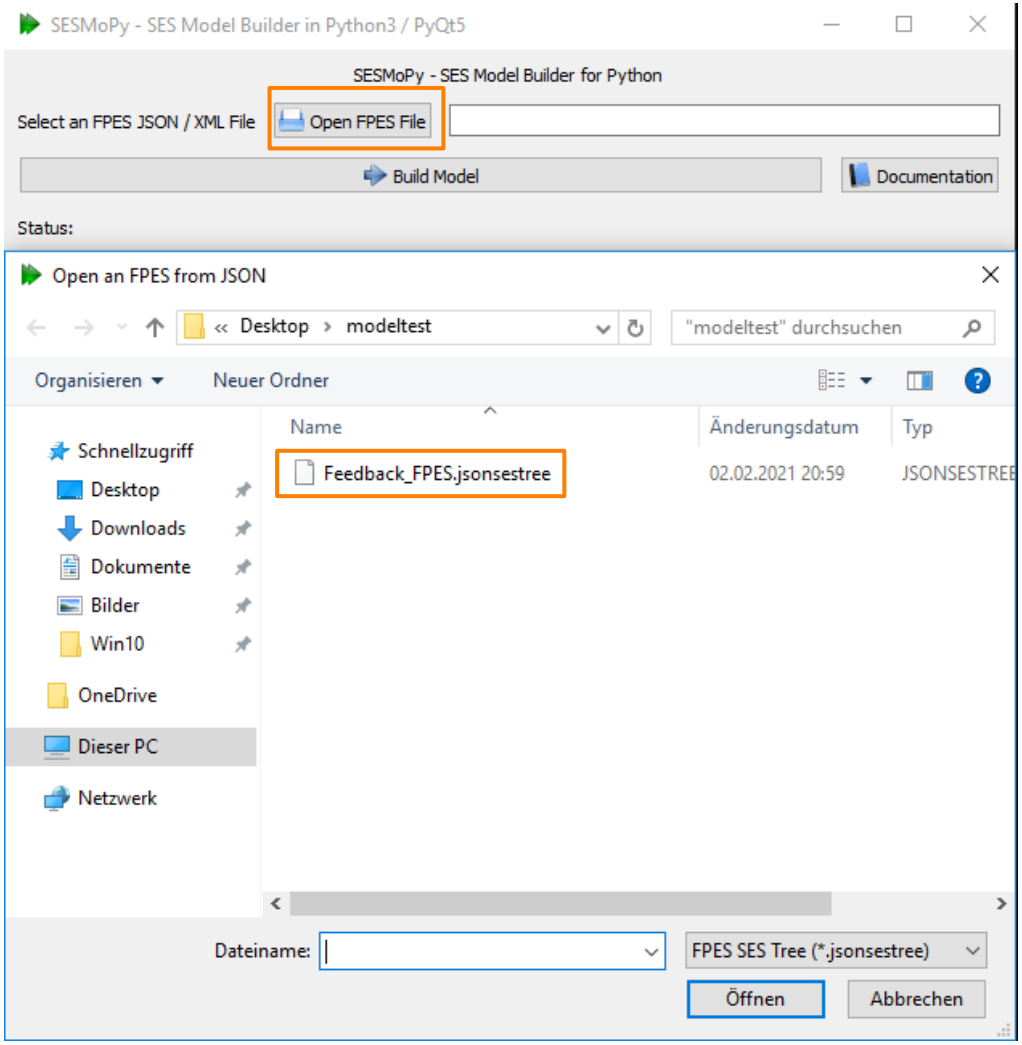
Save Flattened PES for feedforward=0



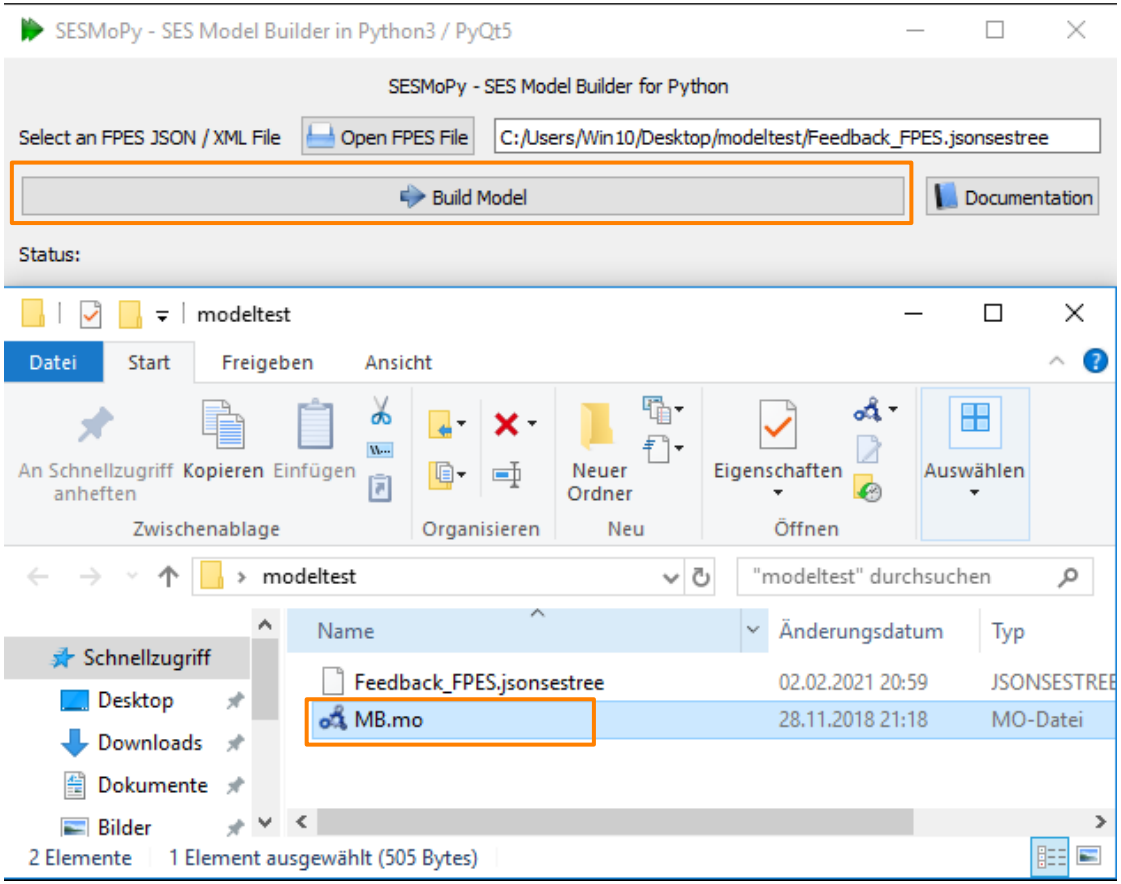
OpenModelica MB



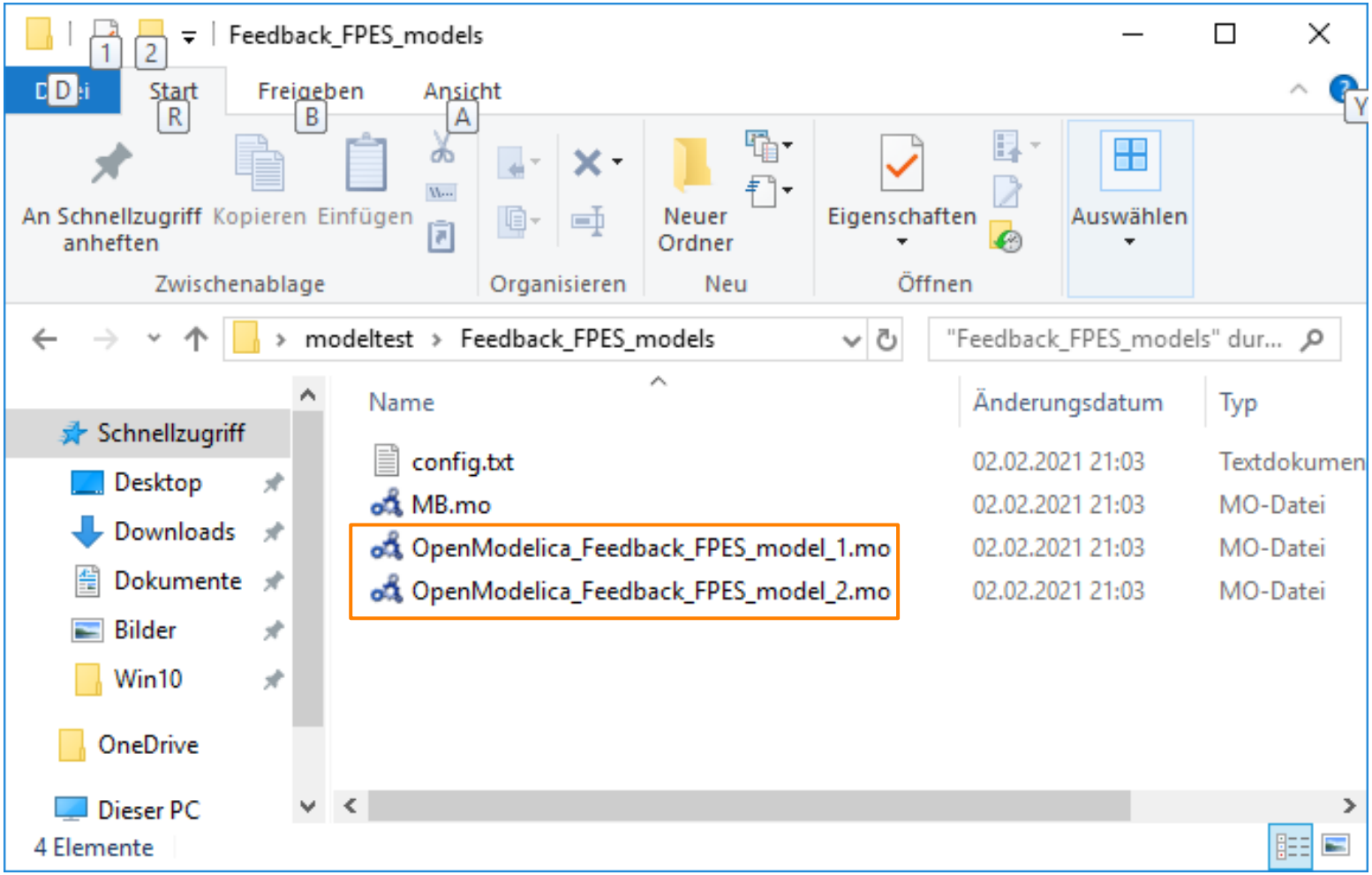
Open Flattened PES in SESMoPy



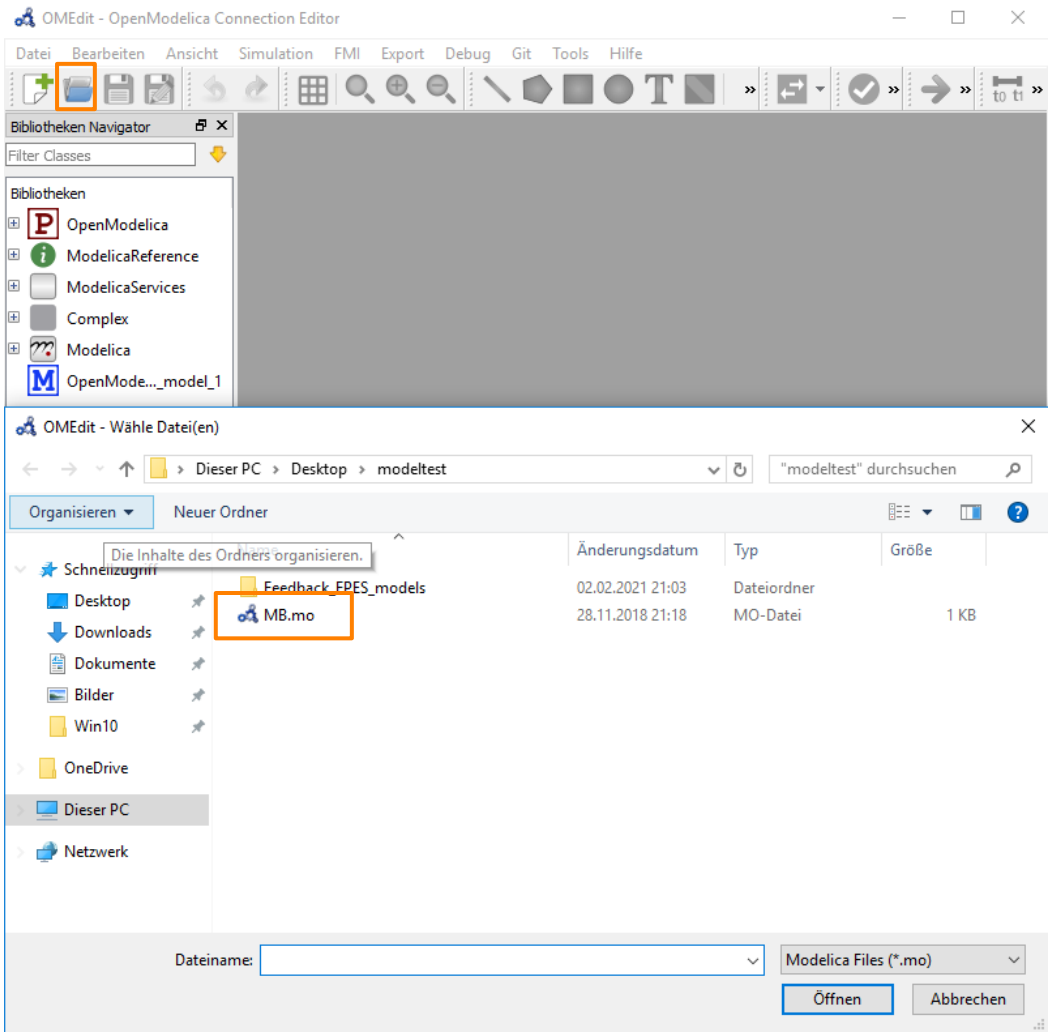
Add MB in Modelfolder & Build Model



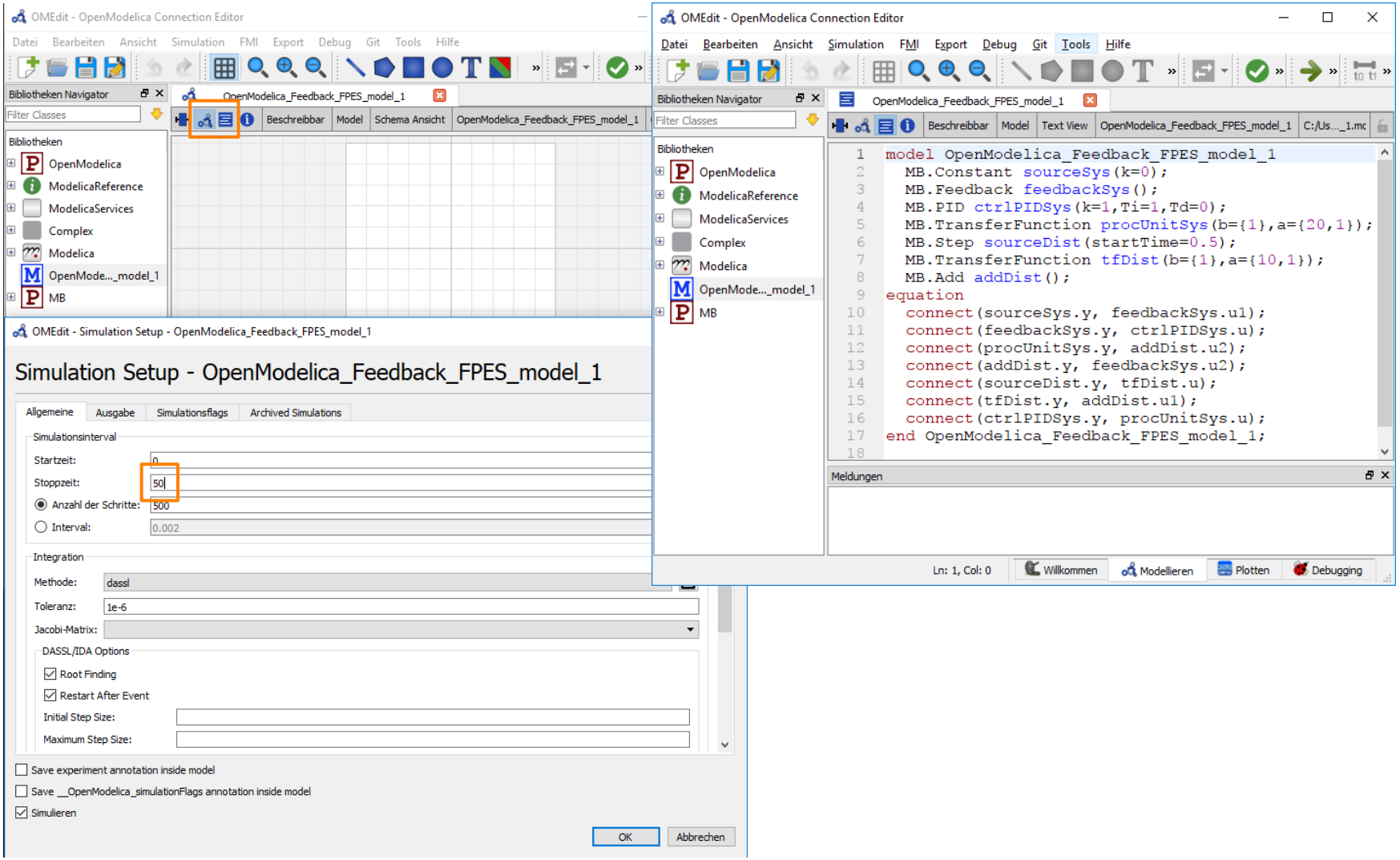
Created Models



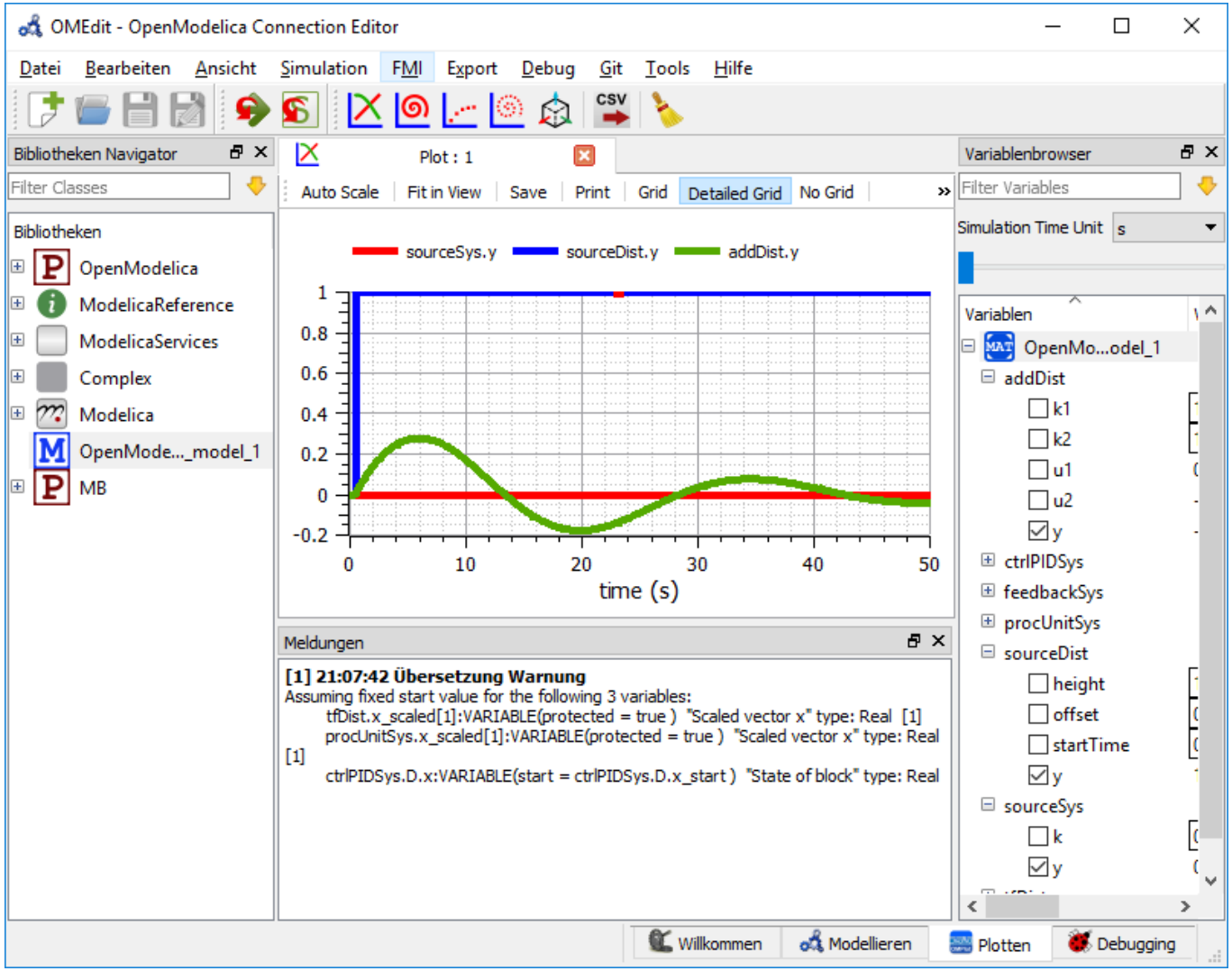
Open Created Models in OpenModelica & Load MB



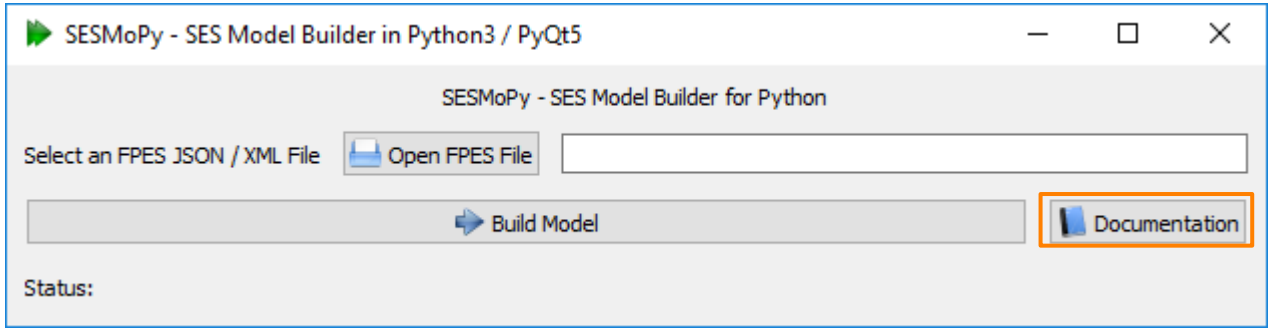
Model & Setup Simulation in OpenModelica



OpenModelica Simulation Results



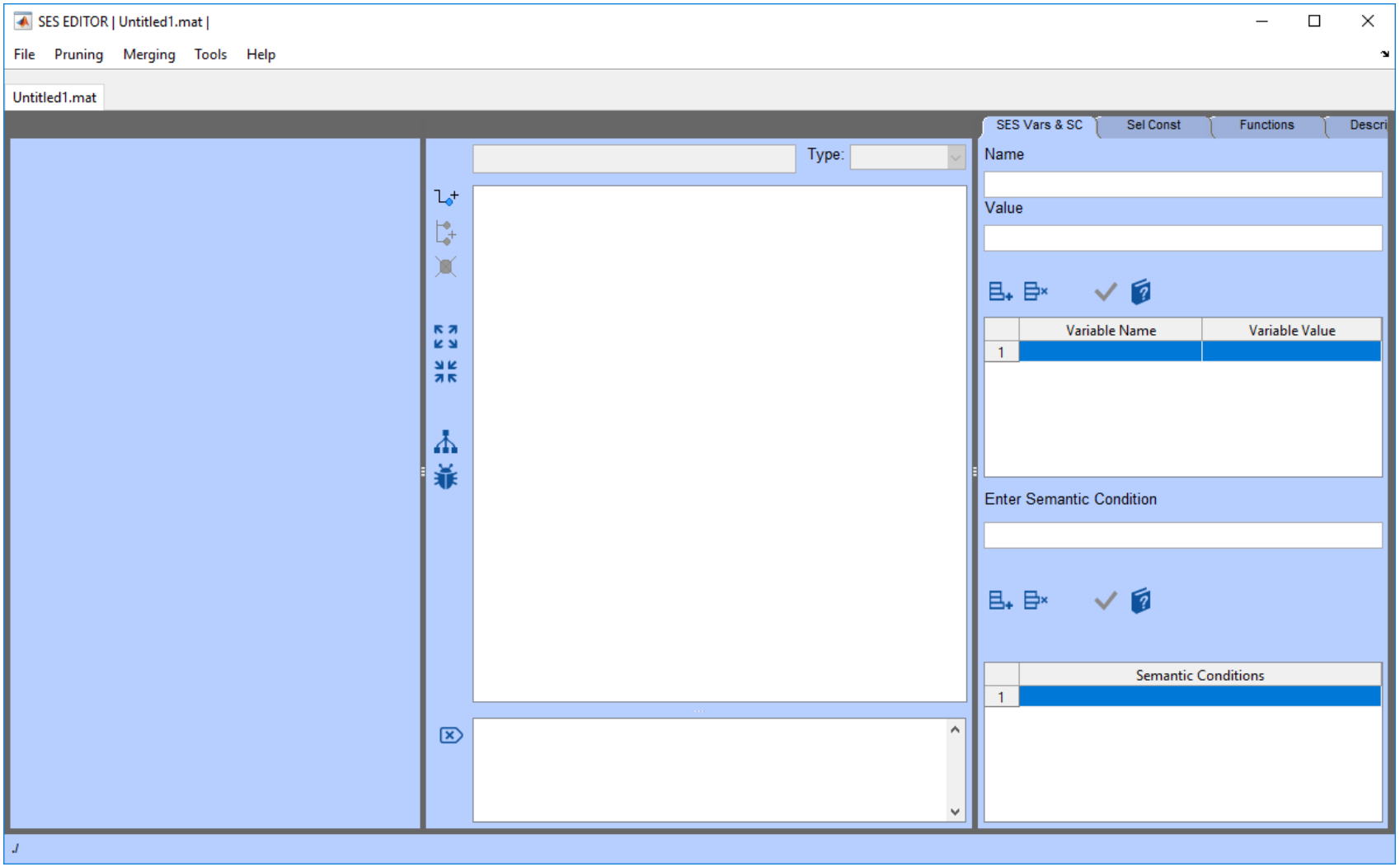
SESMoPy Documentation



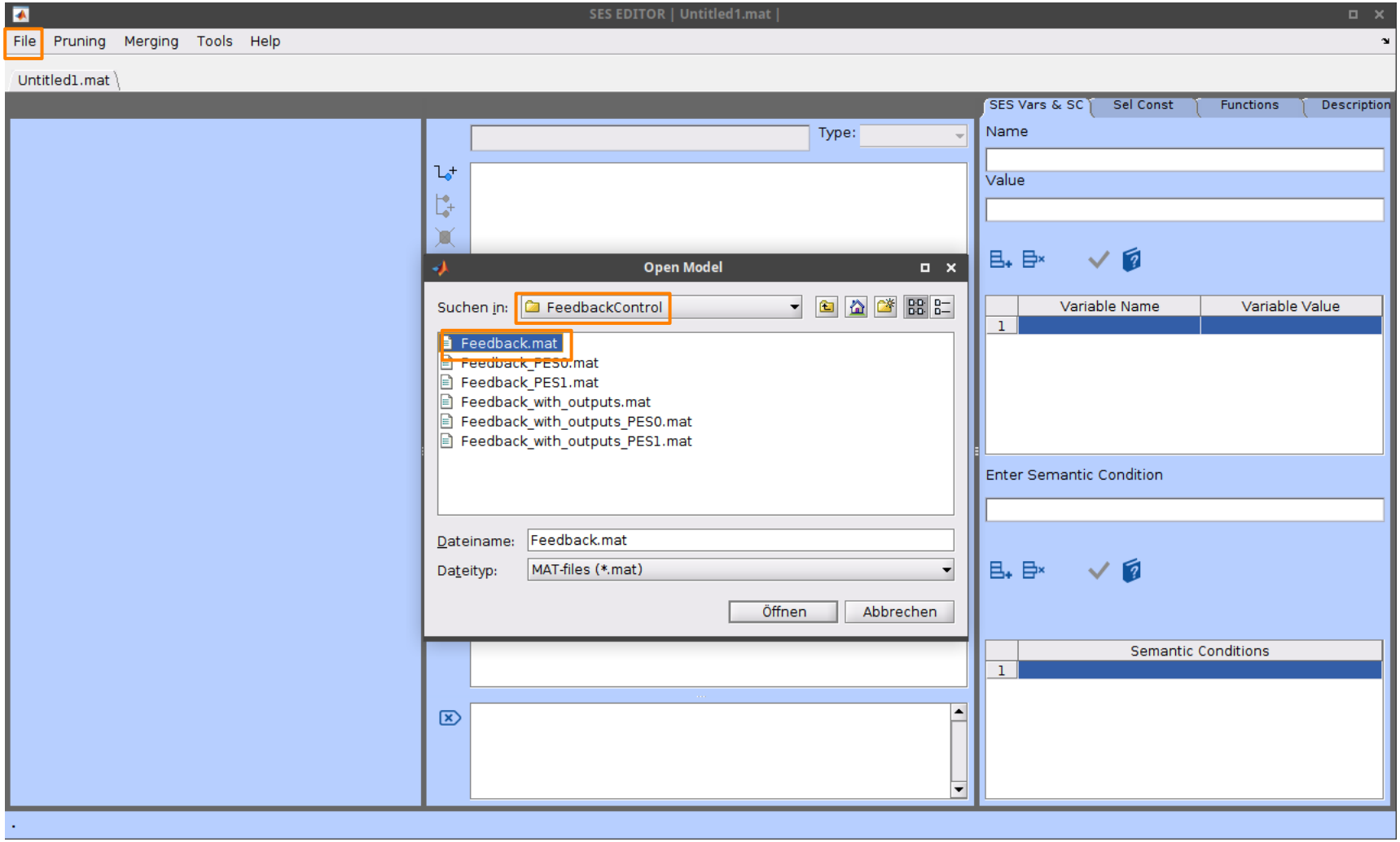
- See the documentation for more information

Chapter 6: Demonstration of the SES Toolbox Matlab Backup Slides

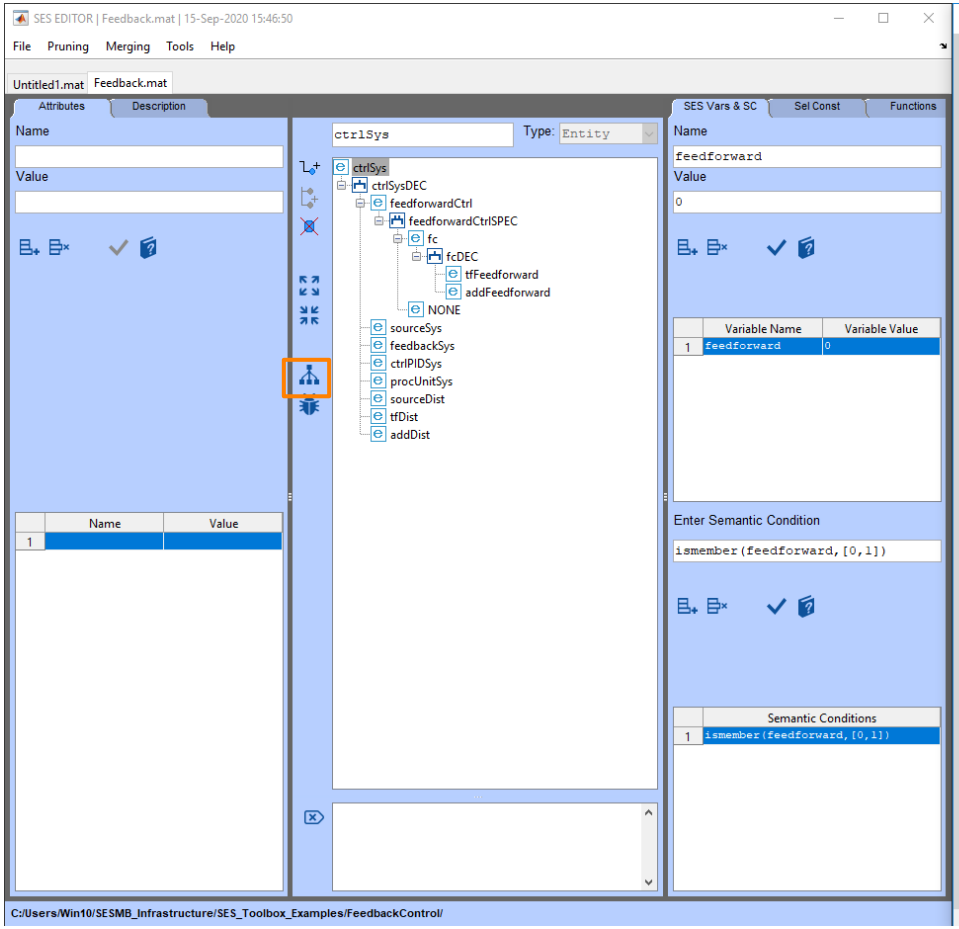
SES Tbx Matlab

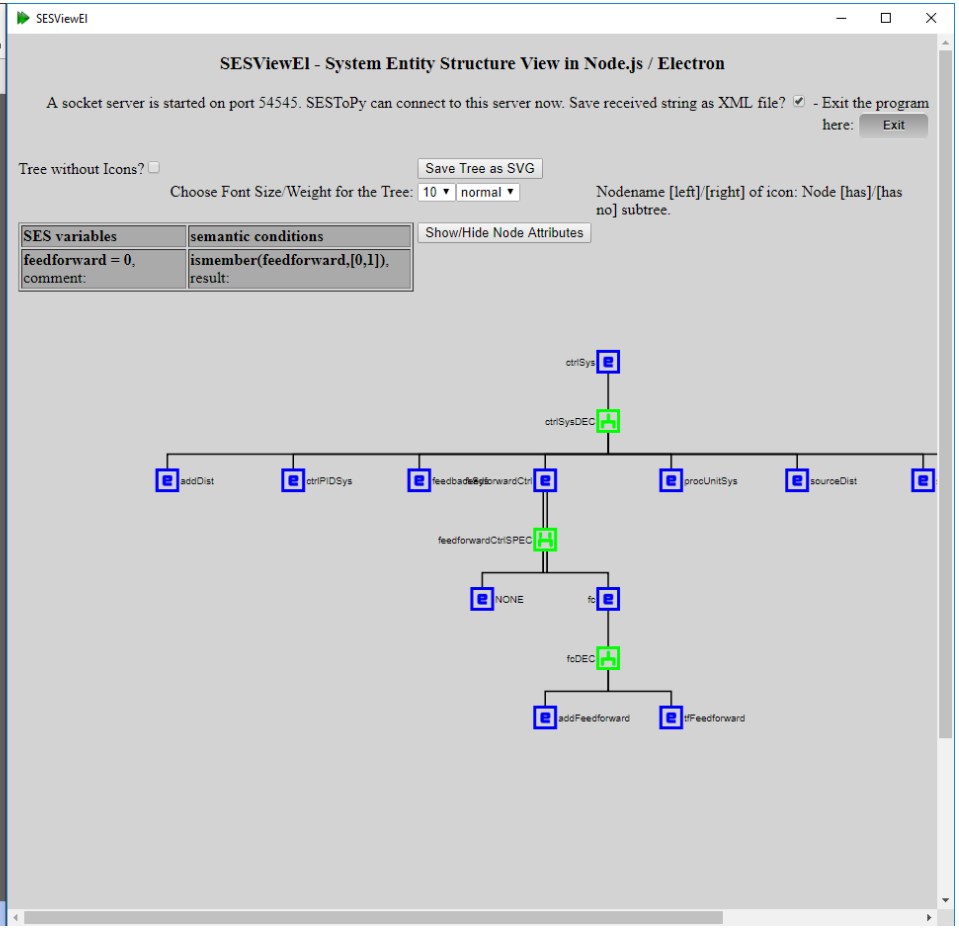


Open the Feedback Example

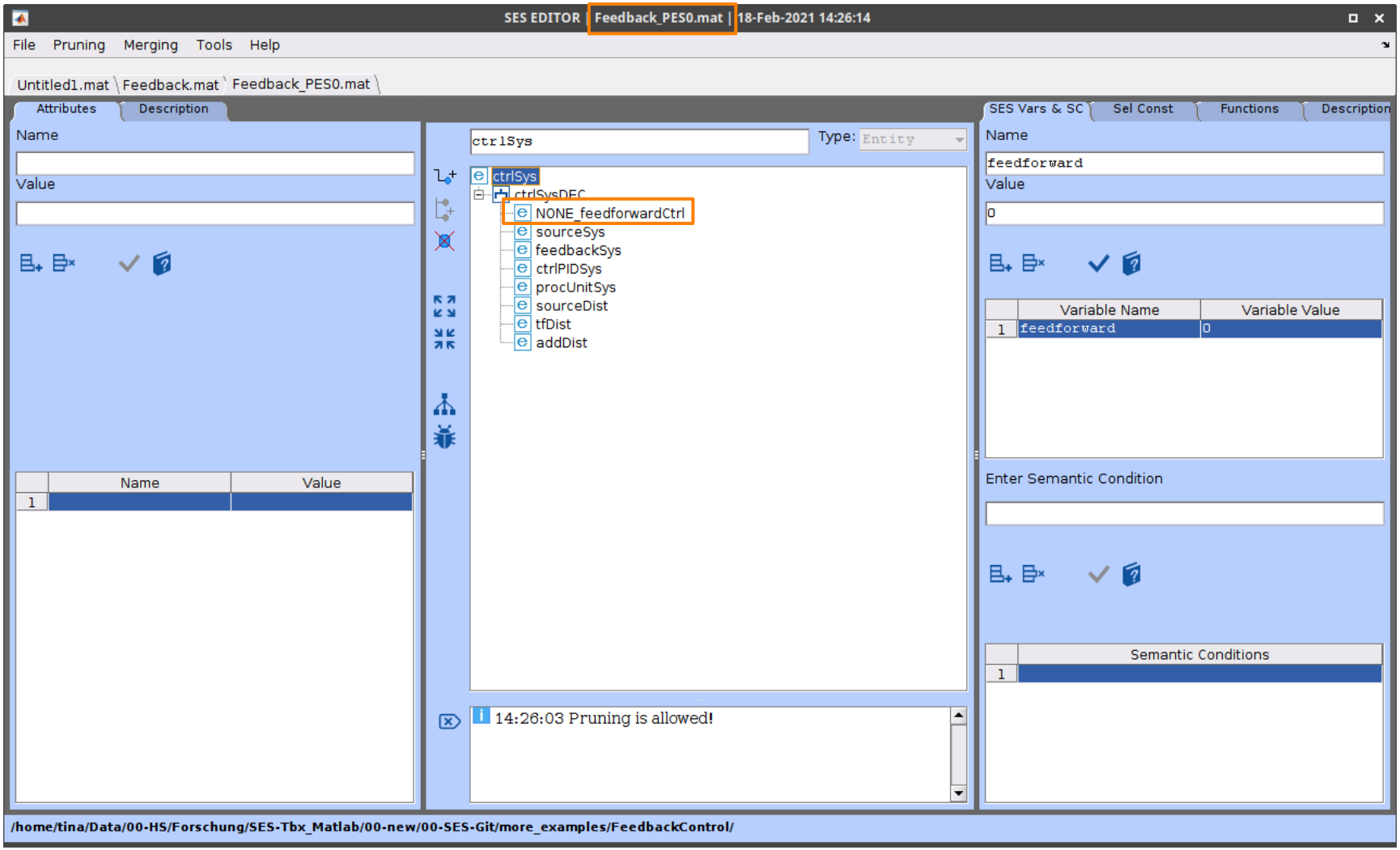


Show Tree in SESViewEl

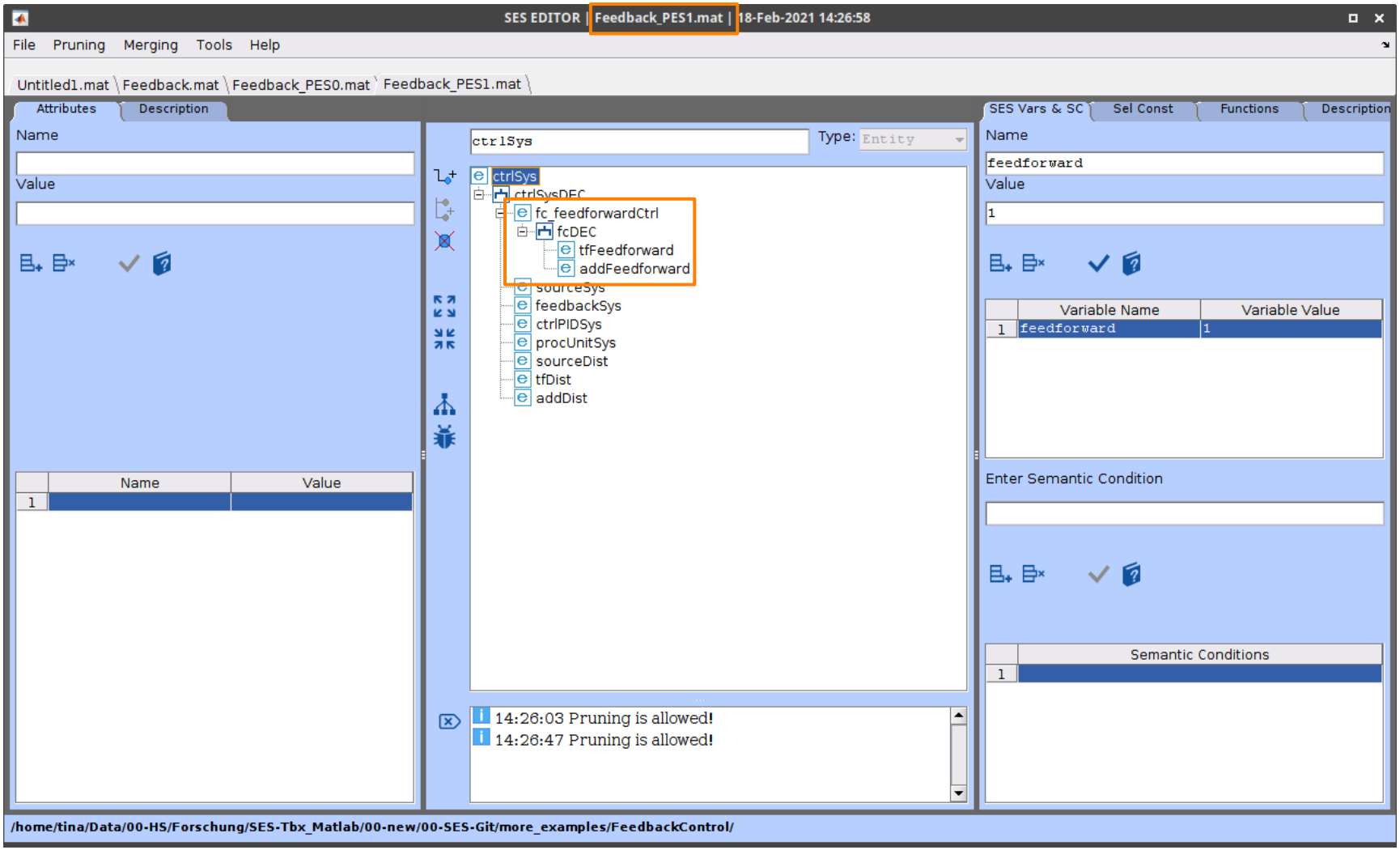




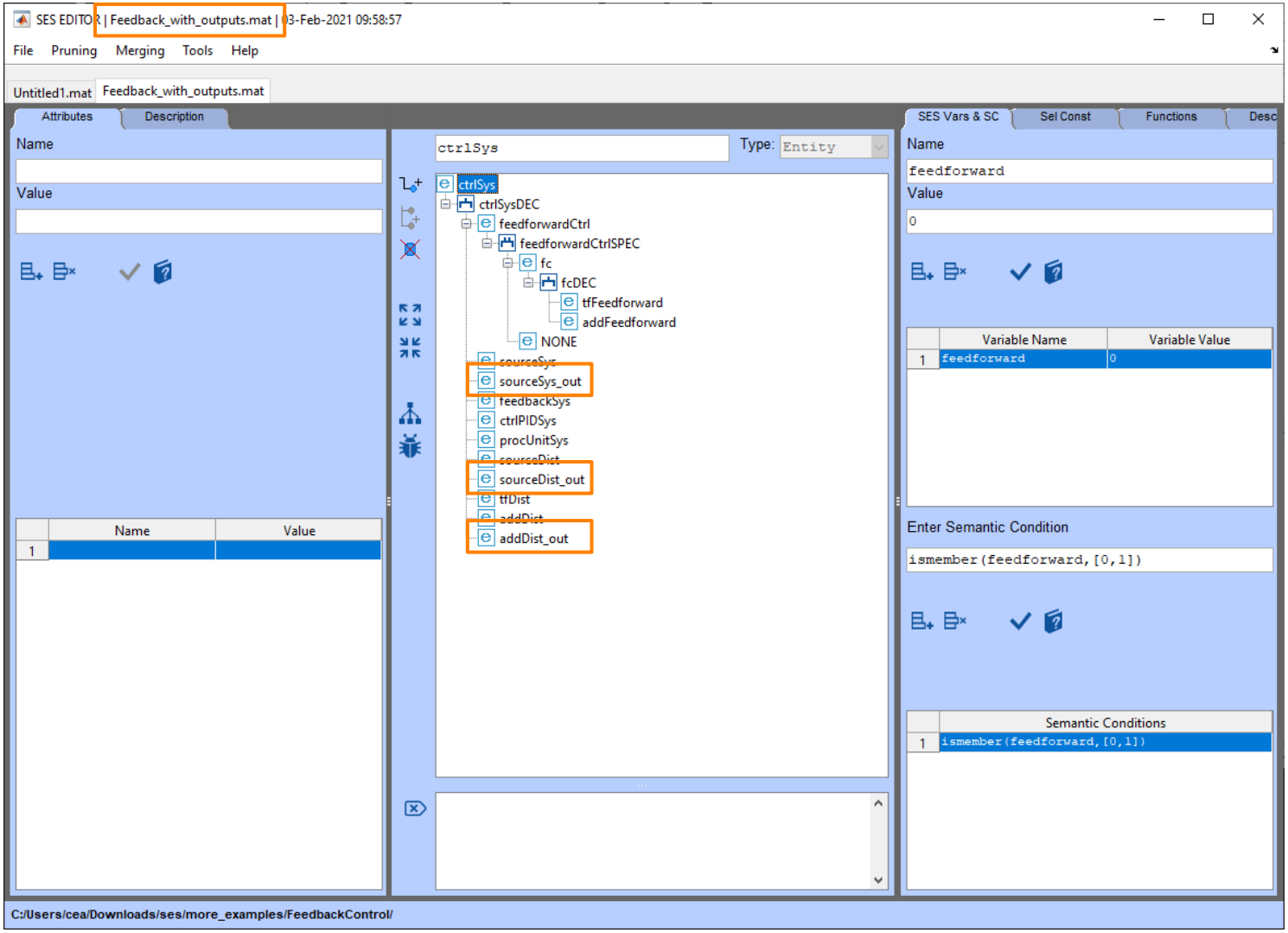
PES for feedforward=0



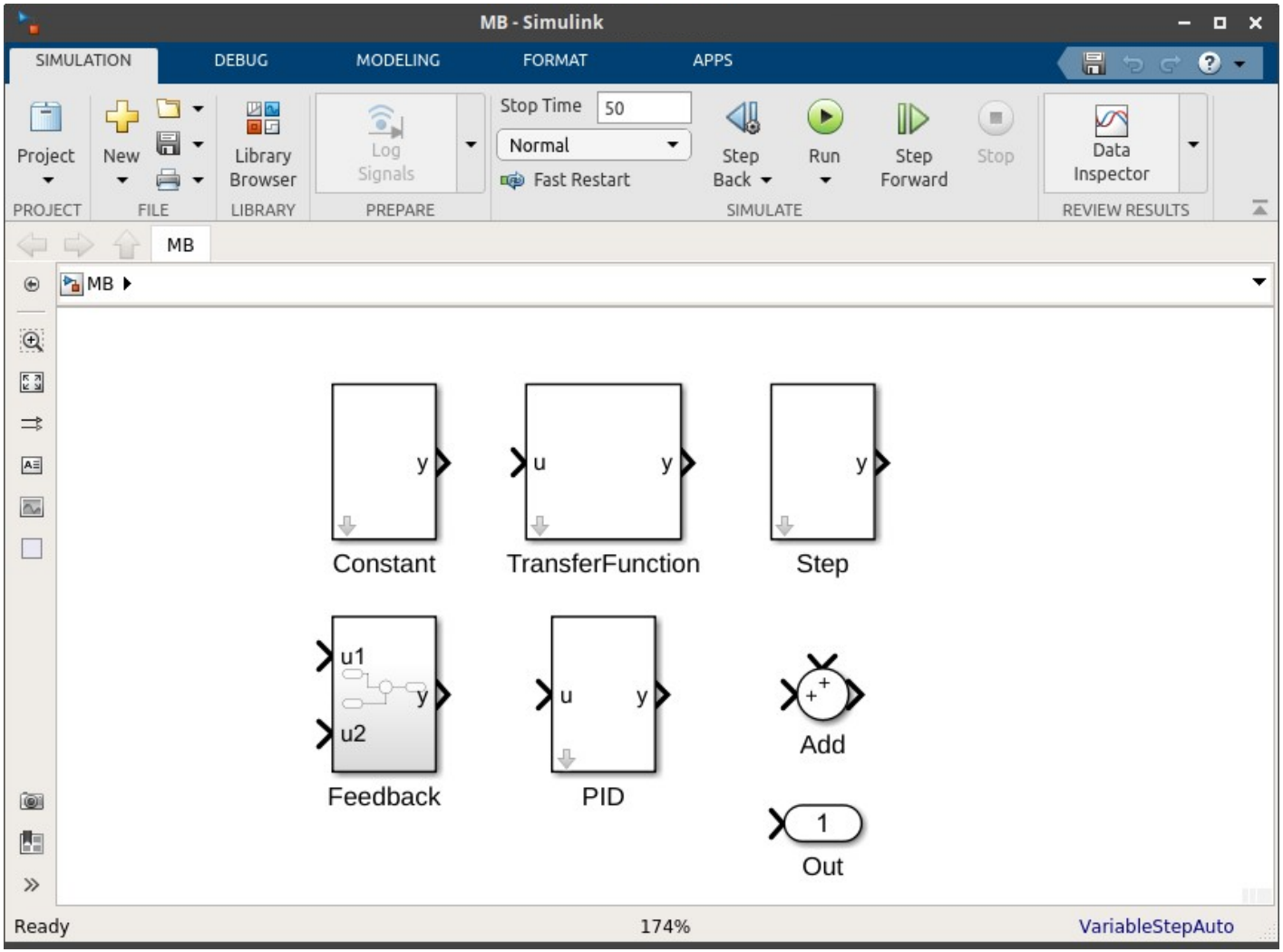
PES for feedforward=1



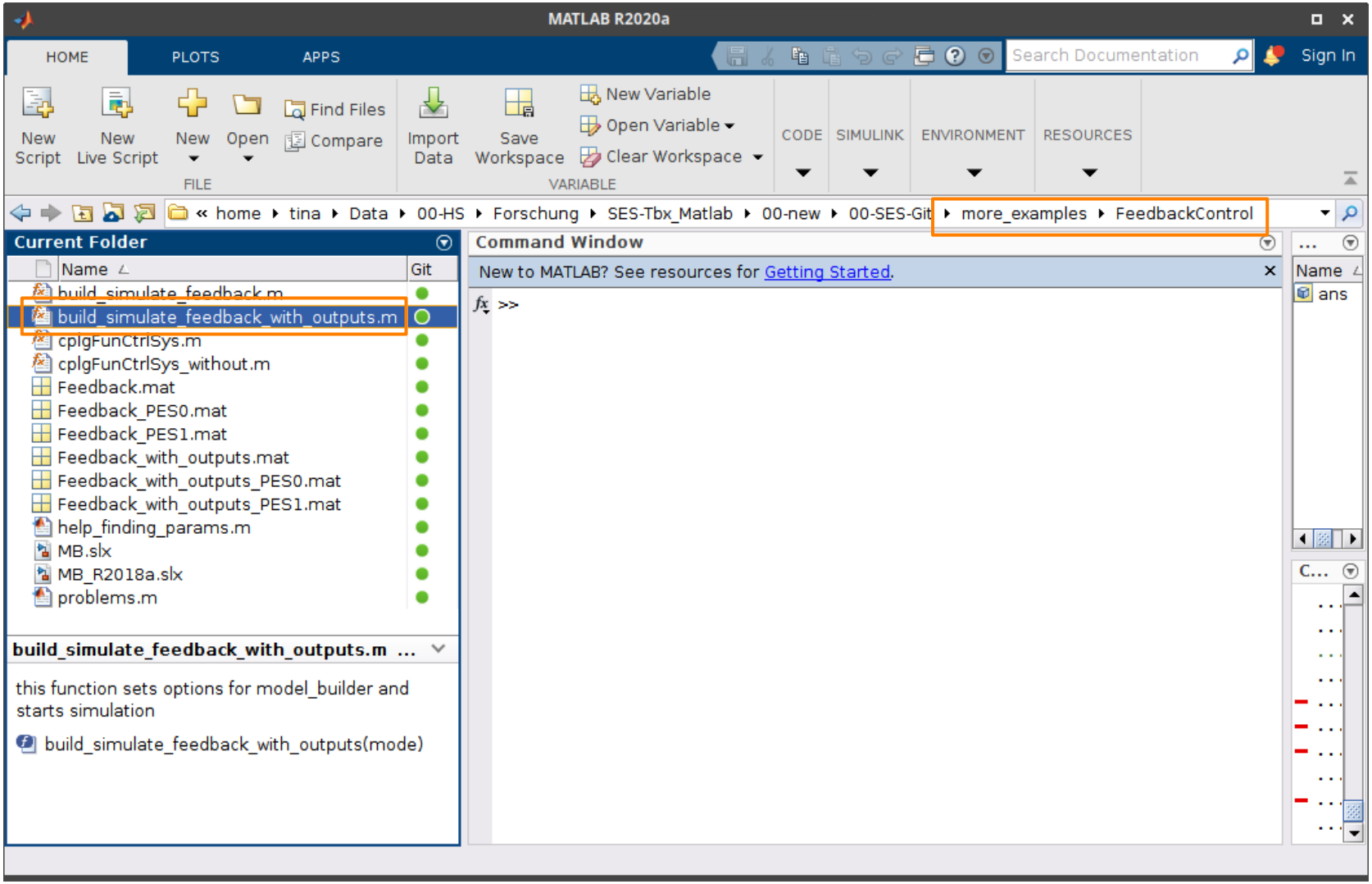
Feedback Example Out Nodes



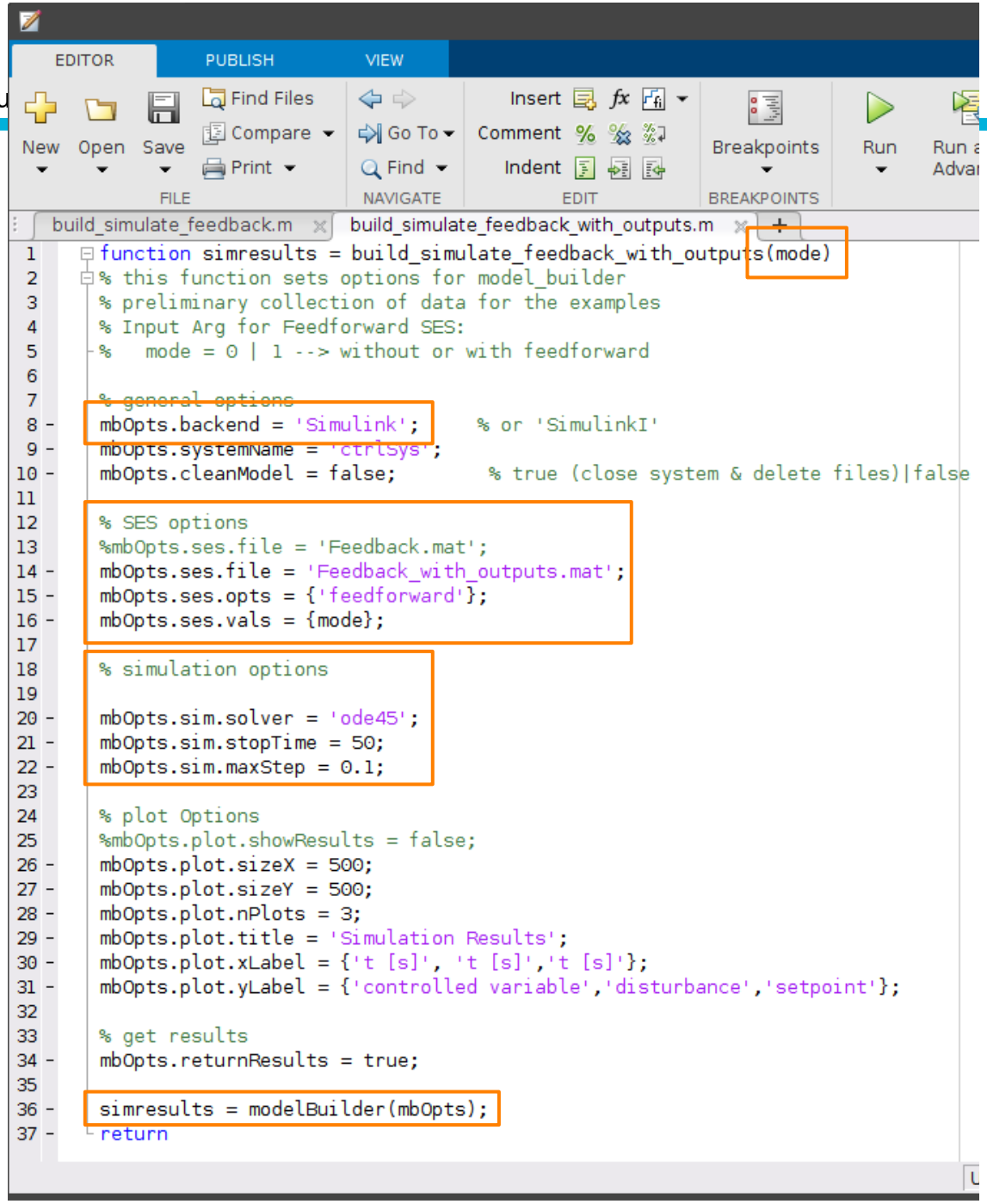
Simulink MB


MB - Simulink
SIMULATION
DEBUG
MODELING
FORMAT
APPS
Project
New
Library Browser
Log Signals
Stop Time
50
Normal
Fast Restart
Step Back
Run
Step Forward
Stop
Data Inspector
PROJECT
FILE
LIBRARY
PREPARE
SIMULATE
REVIEW RESULTS
MB
MB
y
Constant
u
y
TransferFunction
y
Step
u1
u2
y
Feedback
u
y
PID
Add
1
Out
Ready
174%
VariableStepAuto

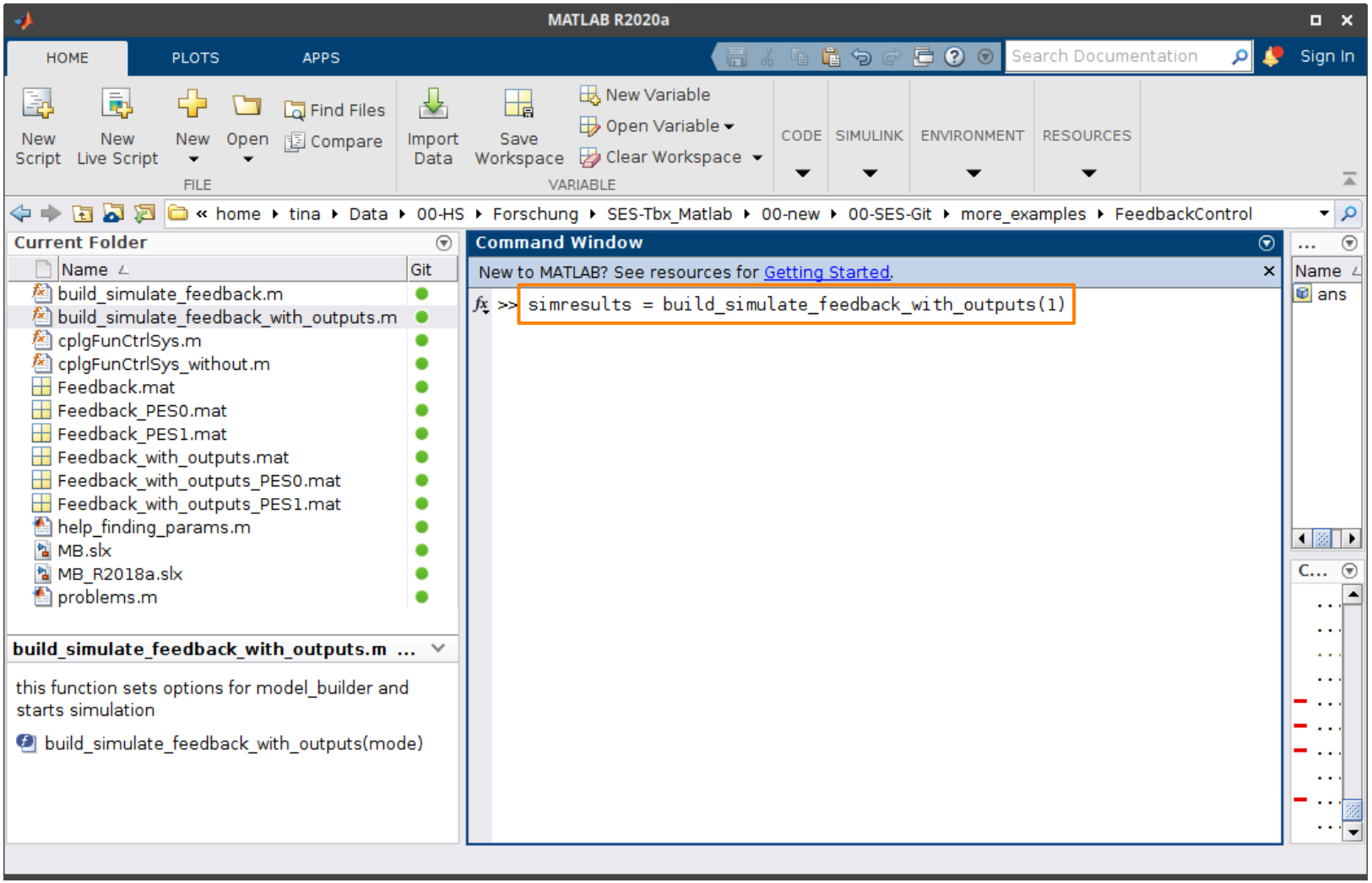
Feedback Example Directory



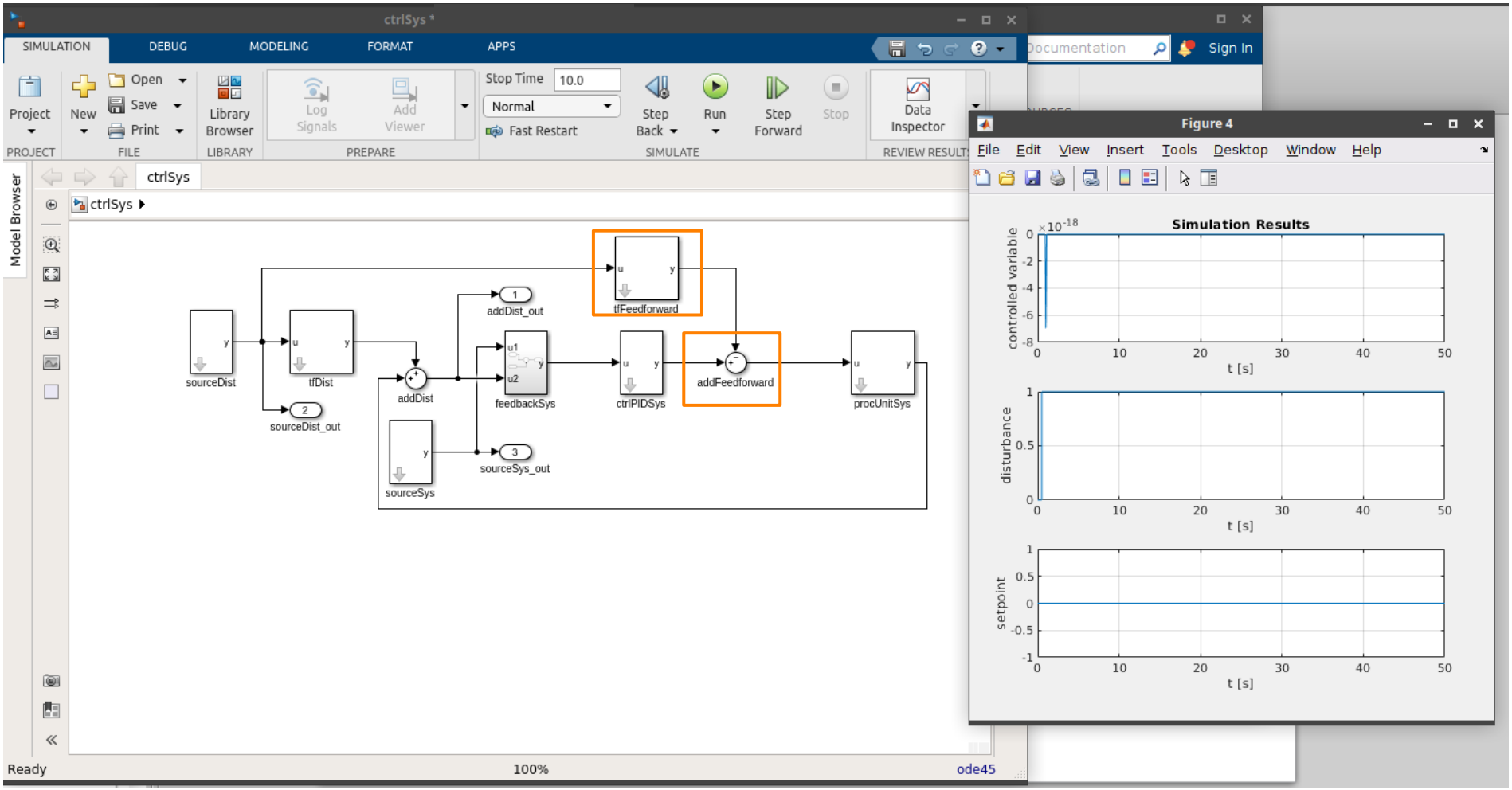
Feedback Example Function to Start Model Builder



Feedback Example Start Model Building



Feedback Example Simulink Model for feedforward=1 and Results



Challenges in Development of the Matlab Model Builder

- Convention for port names used in the SES couplings: PNNN for ‘normal‘ ports, CNN for entity ports.
  → port names are not the same as in model base.
- Parameters of Simulink blocks can be set via a mask of a block in the MB.
- Variables and data types defined in the SES need to match mask variables/Simulink parameters exactly.
- Parameter variations can be achieved via SES variables.
- Simulink needs `Out` blocks to return simulation results.
- Until now, they must be added to the SES, although not part of the system.

Chapter 7: FMI & FMI Model Specification Backup Slides

Model Building – FMI Usage

Model Building – FMI Usage

- One model FMU for each parameter variation of one structure variant

Model Building – FMI Usage

- One model FMU for each parameter variation of one structure variant
- Model FMU includes all logic and parameters
  - Completely simulator independent

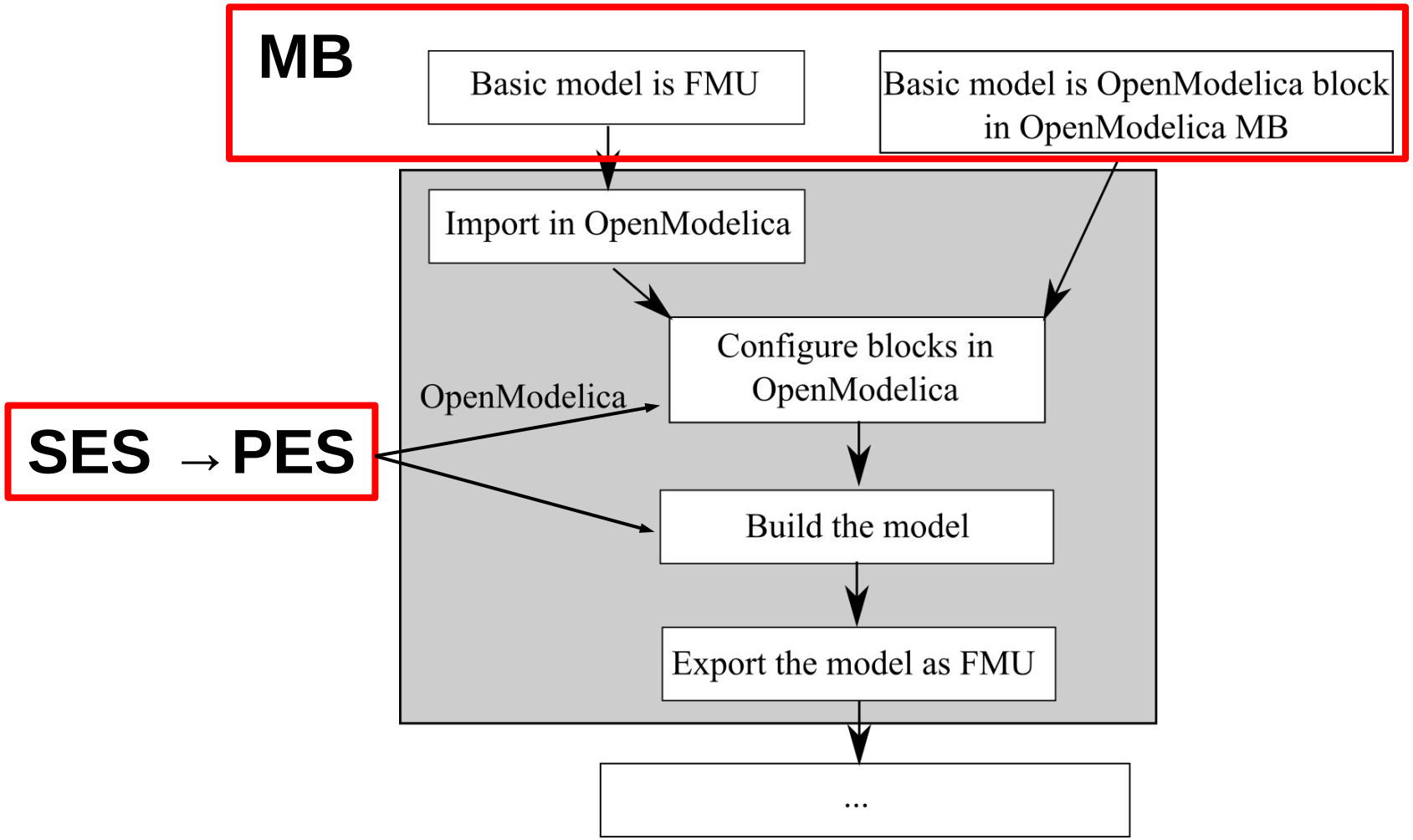
Model Building – FMI Usage

- One model FMU for each parameter variation of one structure variant
- Model FMU includes all logic and parameters
  - Completely simulator independent
- **Not needed:**
  - Simulator specific couplings (portnames, porttypes)
  - Simulator specific parameterization
  - Simulator specific syntax

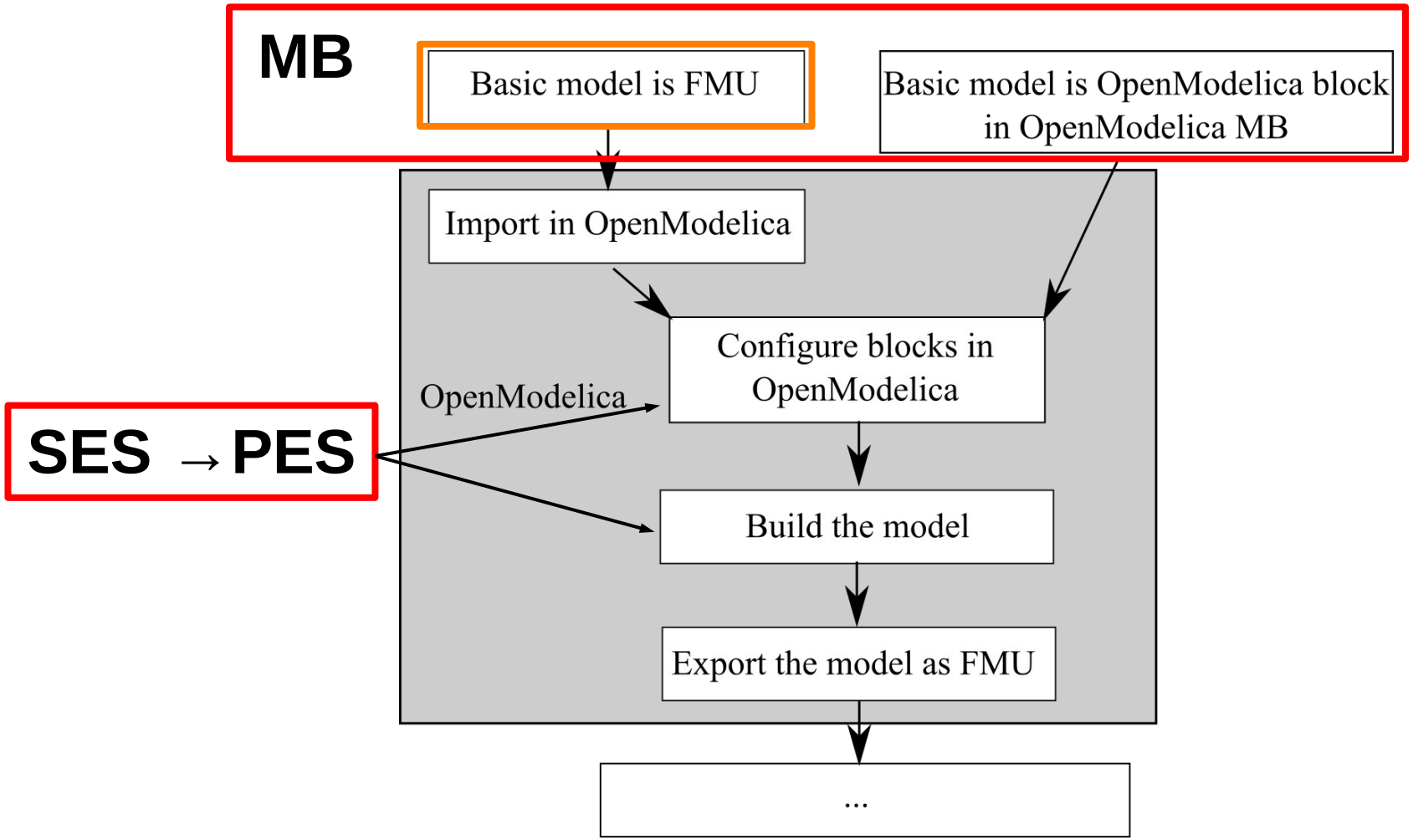
Model Building – FMI Usage

- One model FMU for each parameter variation of one structure variant
- Model FMU includes all logic and parameters
  - Completely simulator independent
- **Not needed:**
  - Simulator specific couplings (portnames, porttypes)
  - Simulator specific parameterization
  - Simulator specific syntax
- **Needed:**
  - Simulator needs to support FMI

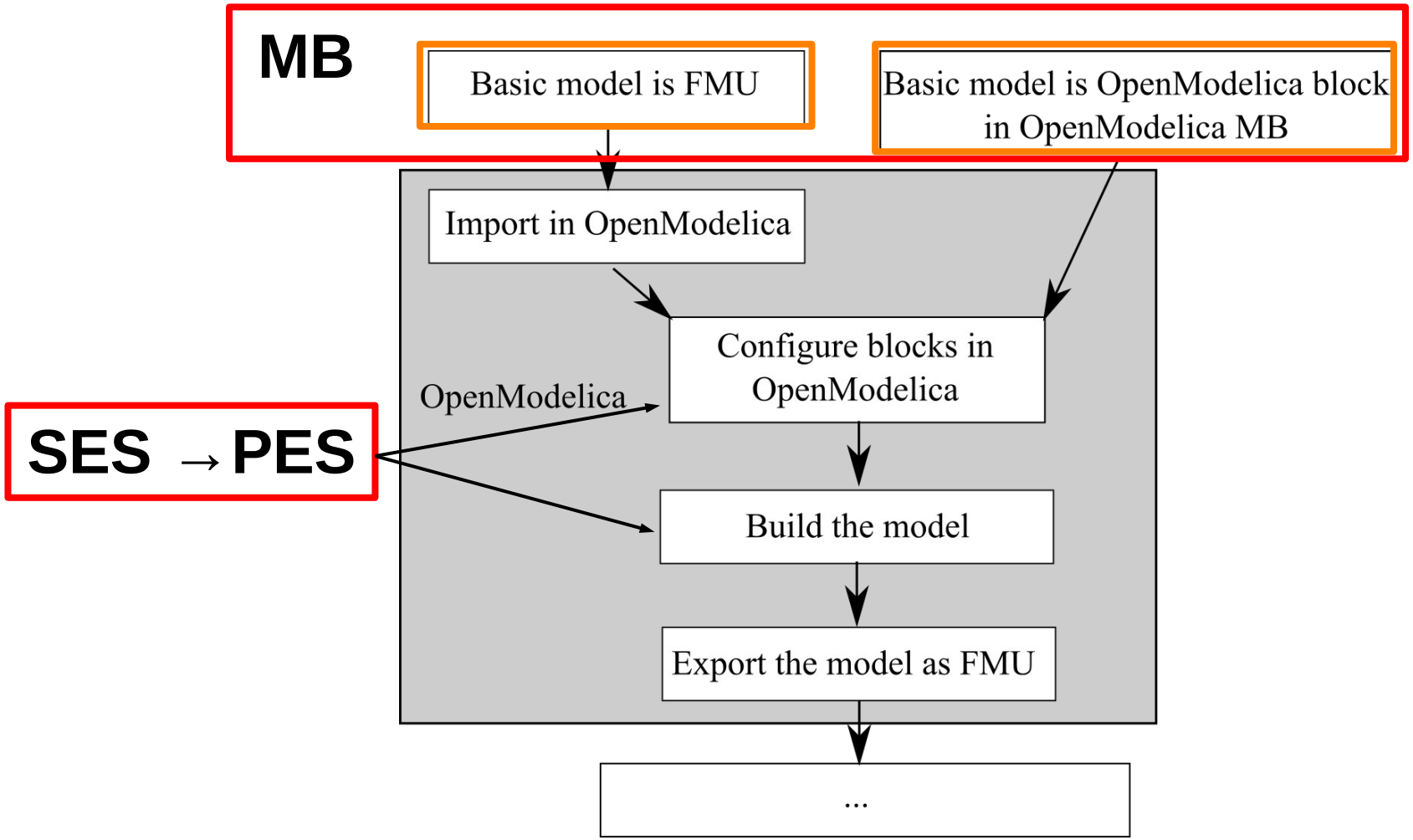
Model Building Using FMI (state of the art)



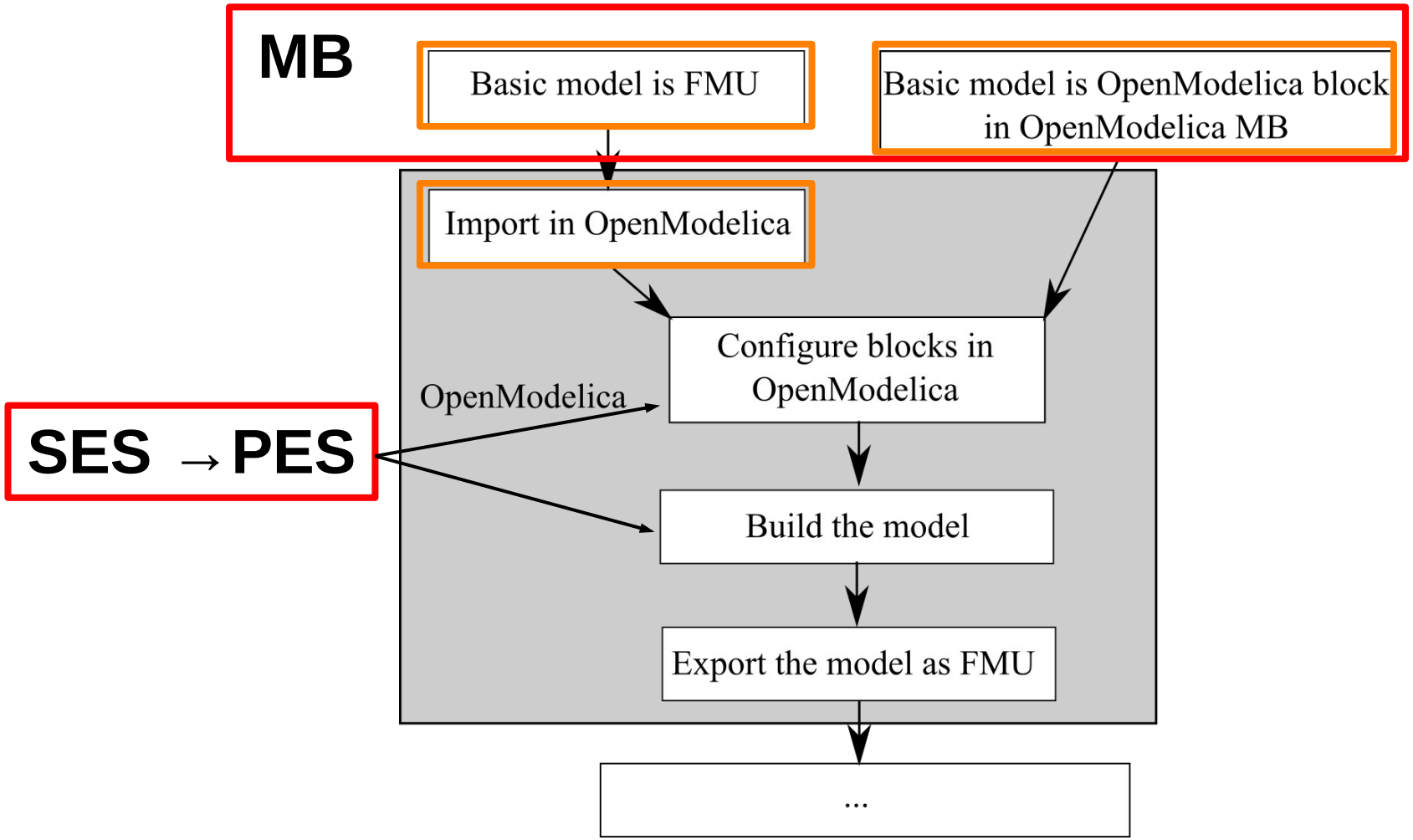
Model Building Using FMI (state of the art)



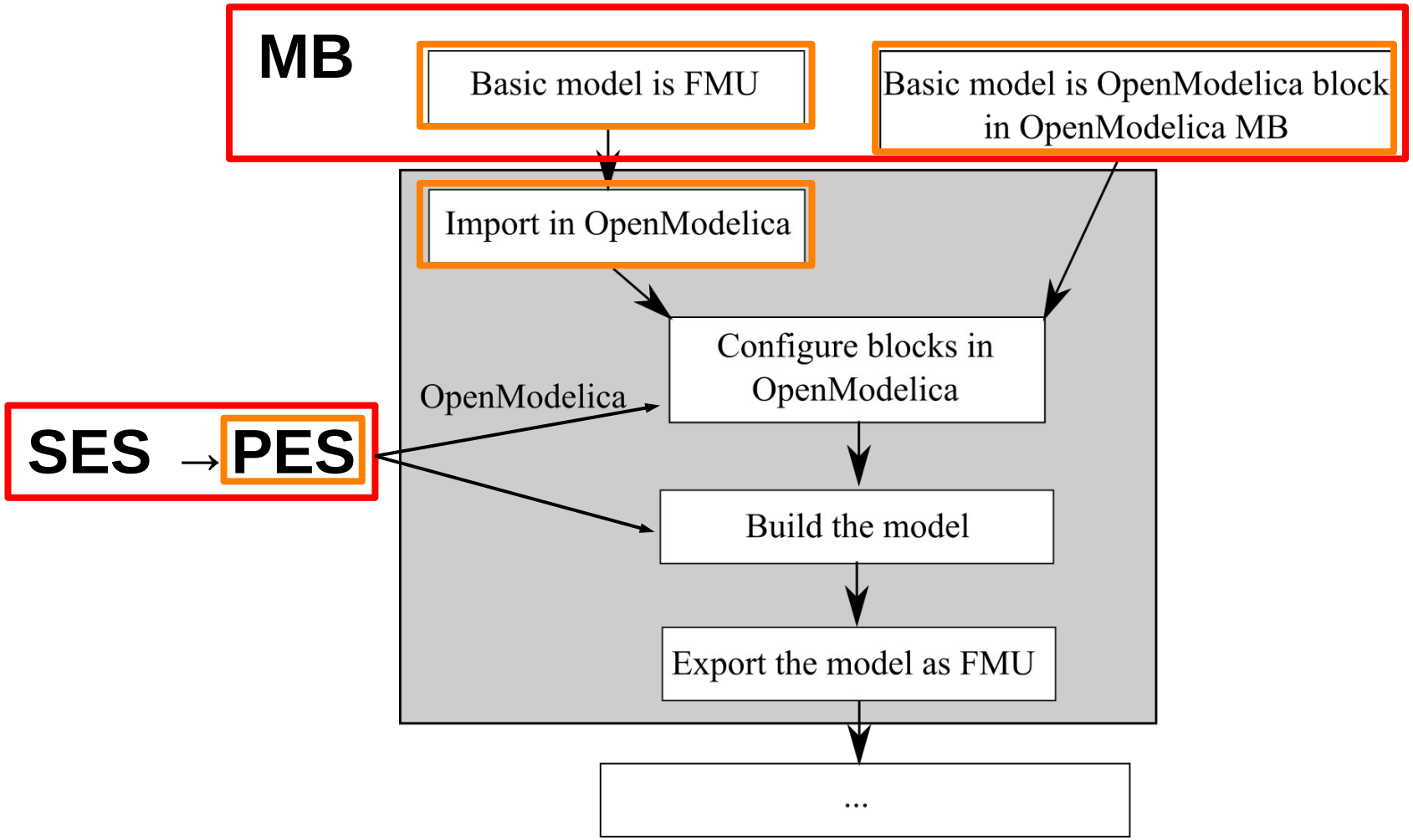
Model Building Using FMI (state of the art)



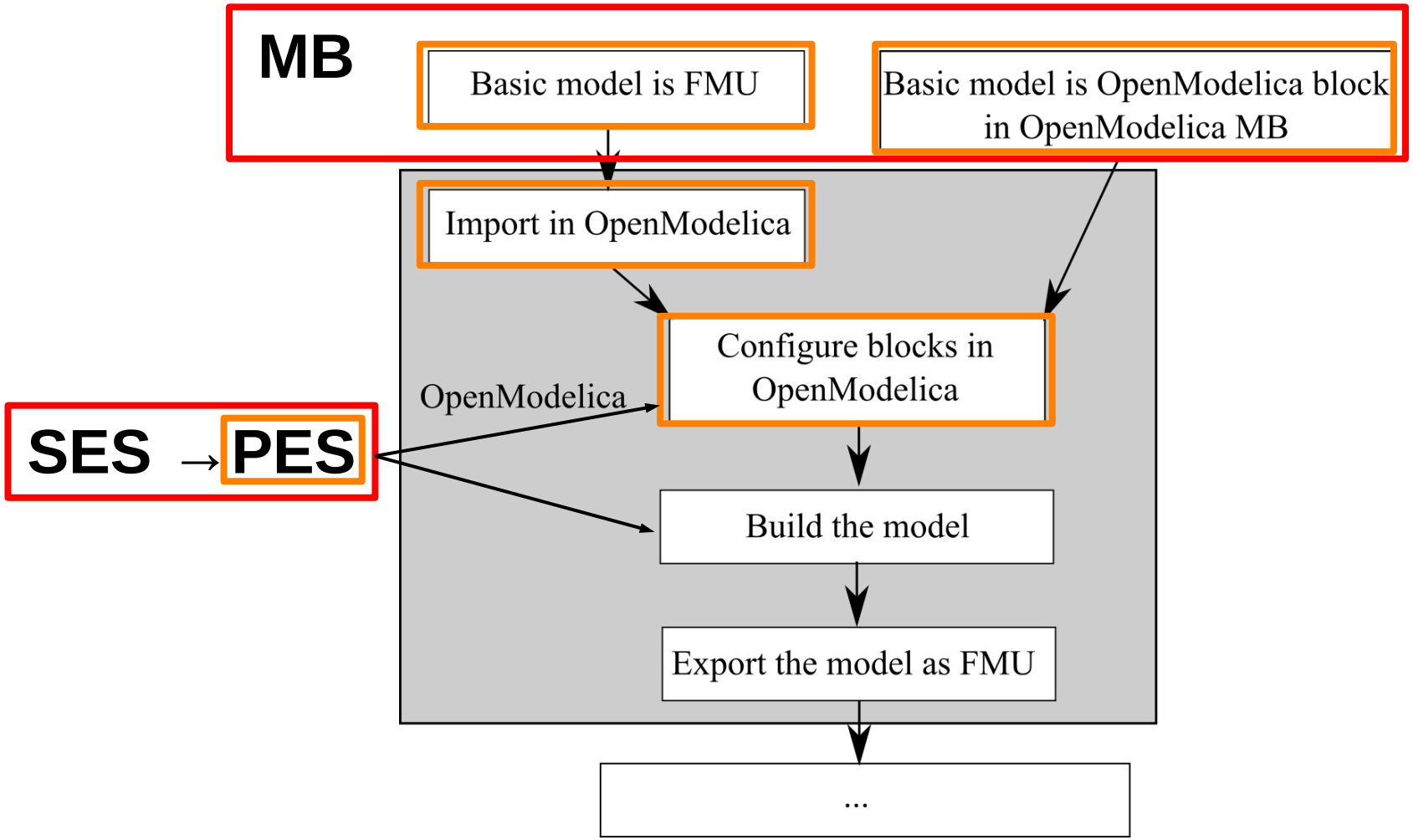
Model Building Using FMI (state of the art)



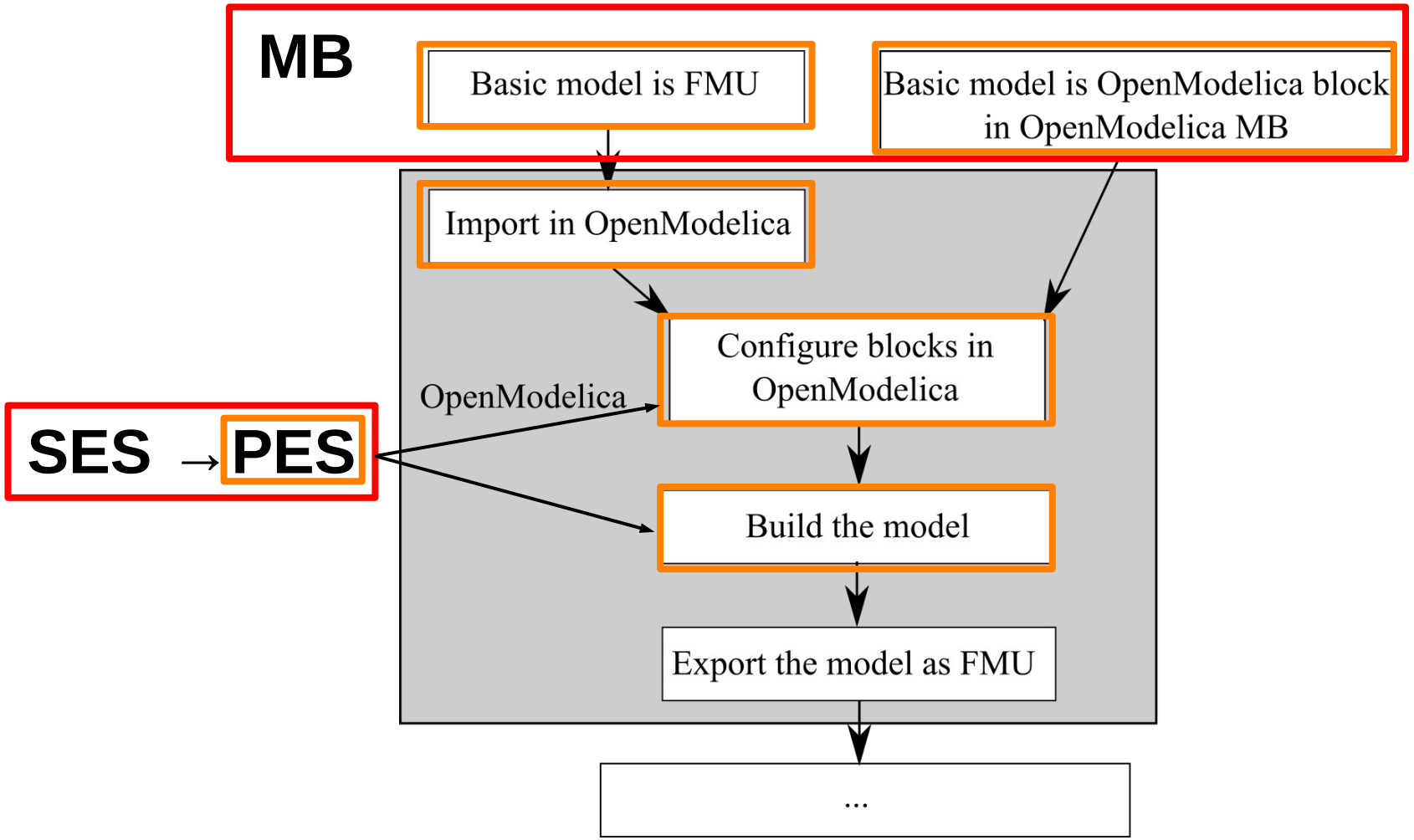
Model Building Using FMI (state of the art)



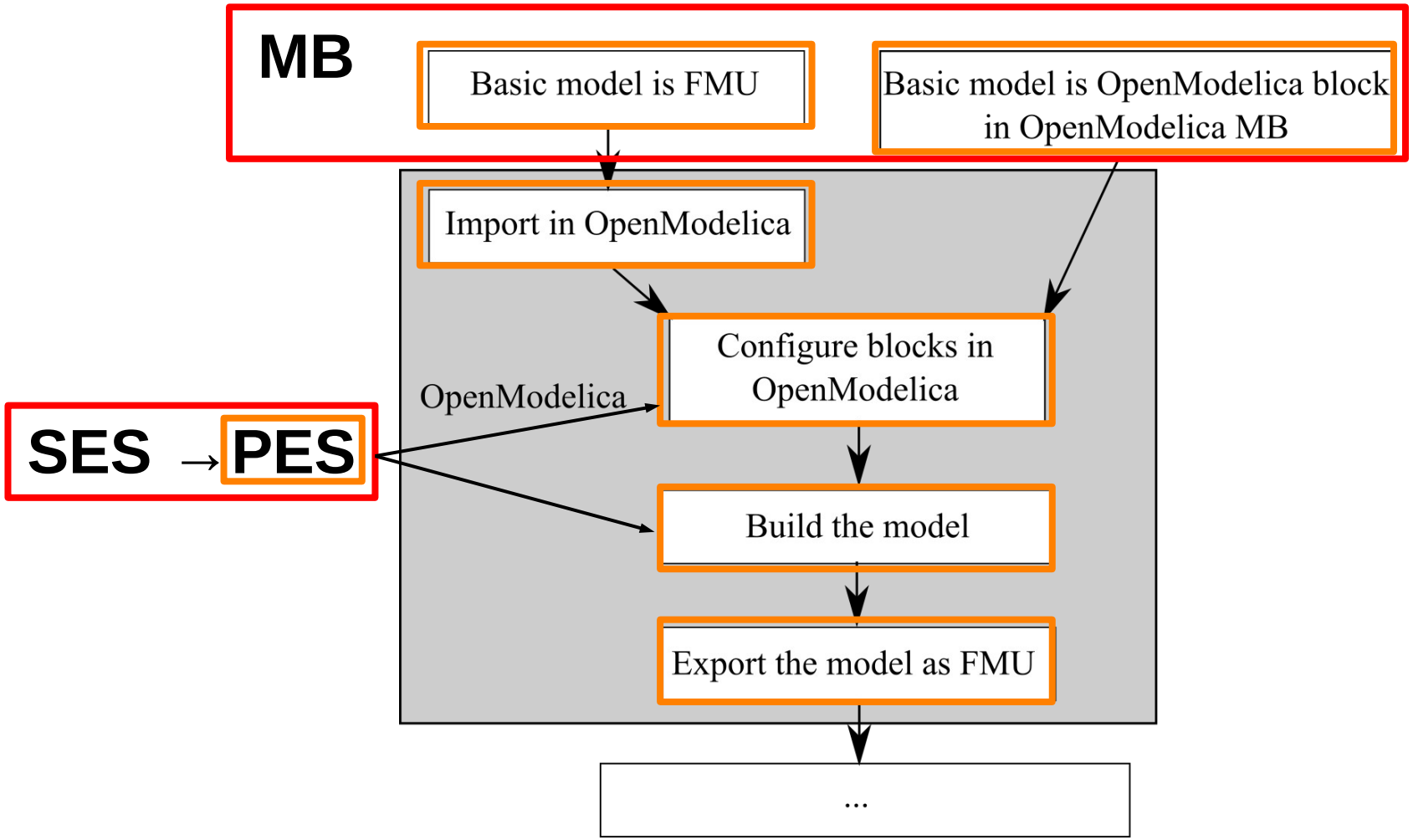
Model Building Using FMI (state of the art)



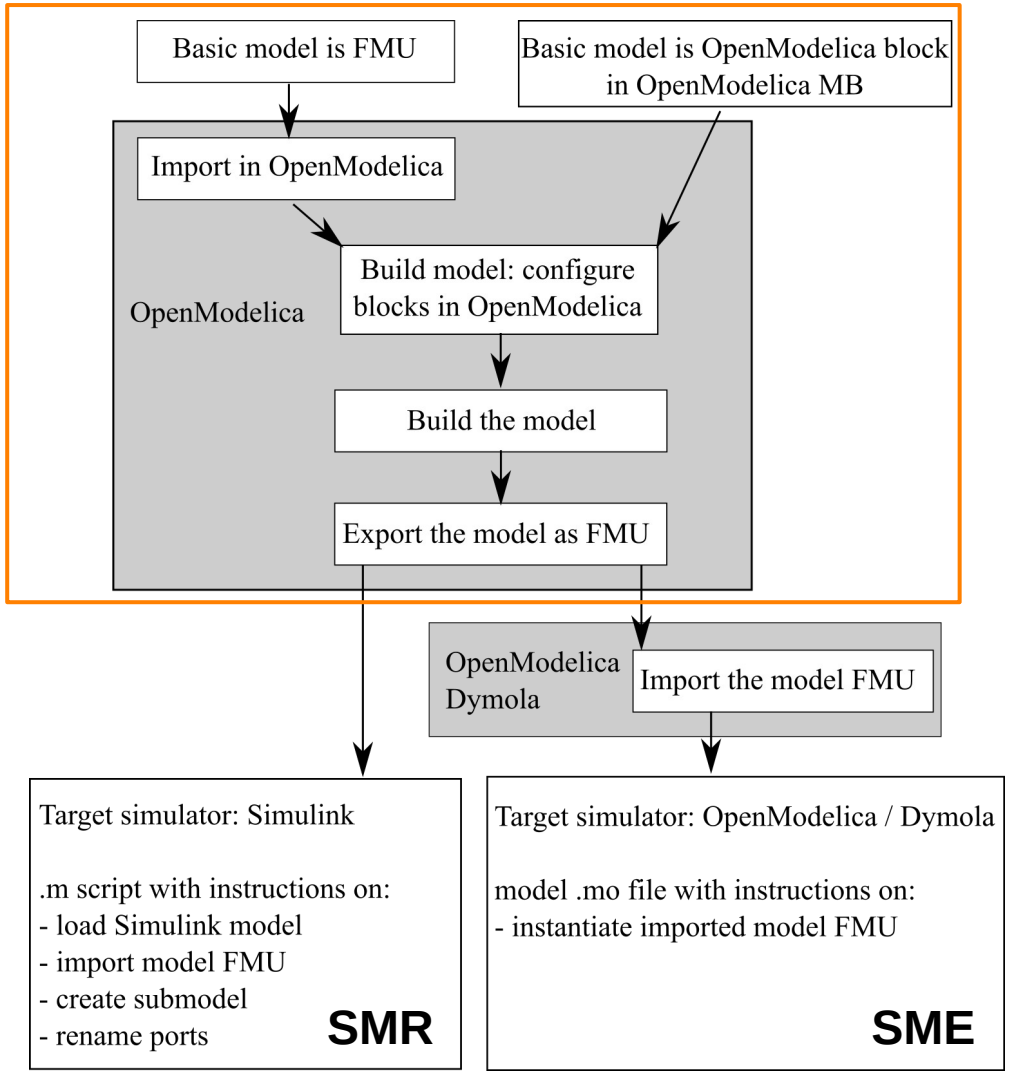
Model Building Using FMI (state of the art)



Model Building Using FMI (state of the art)

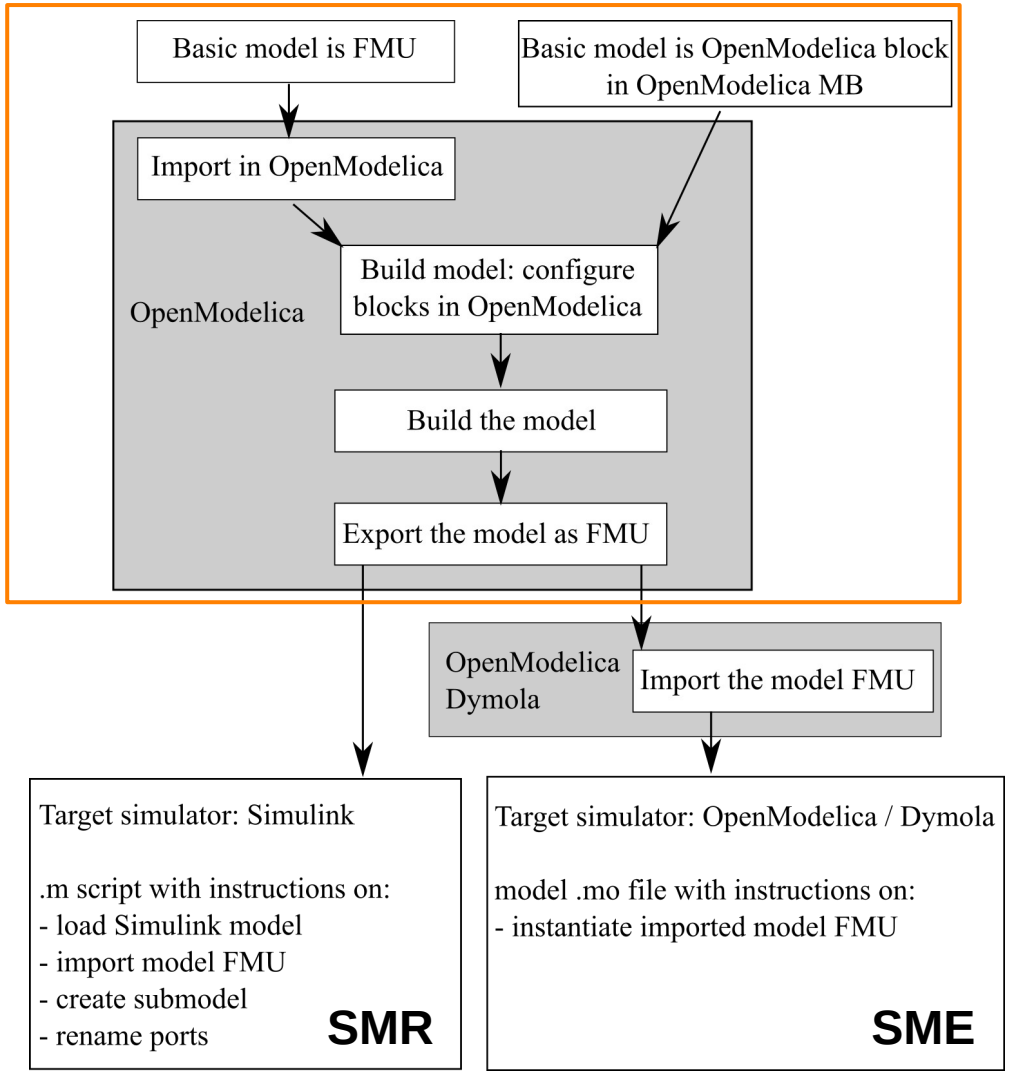


Model Building Using FMI (2) (state of the art)



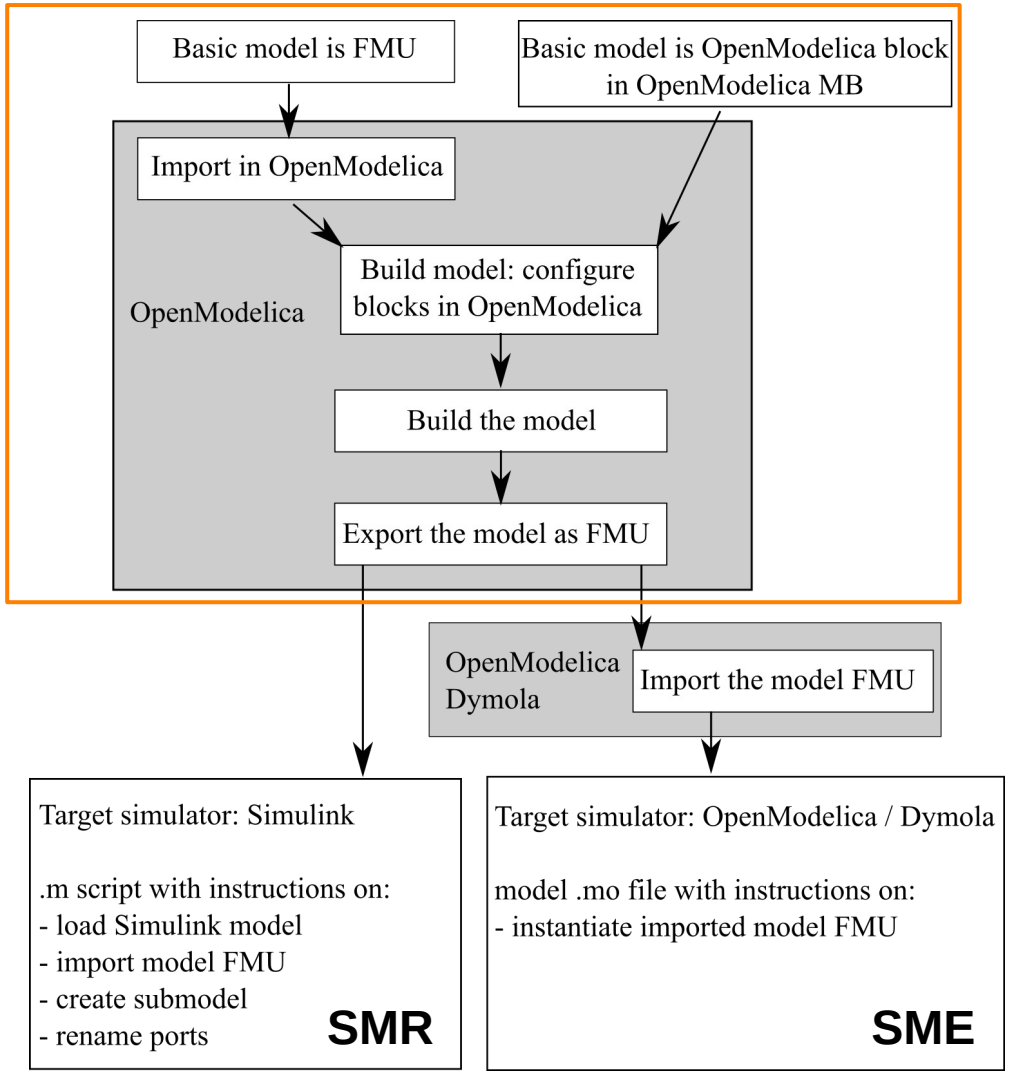
Model Building Using FMI (2) (state of the art)

- **Create simulator specific instructions on how to execute the model FMU**



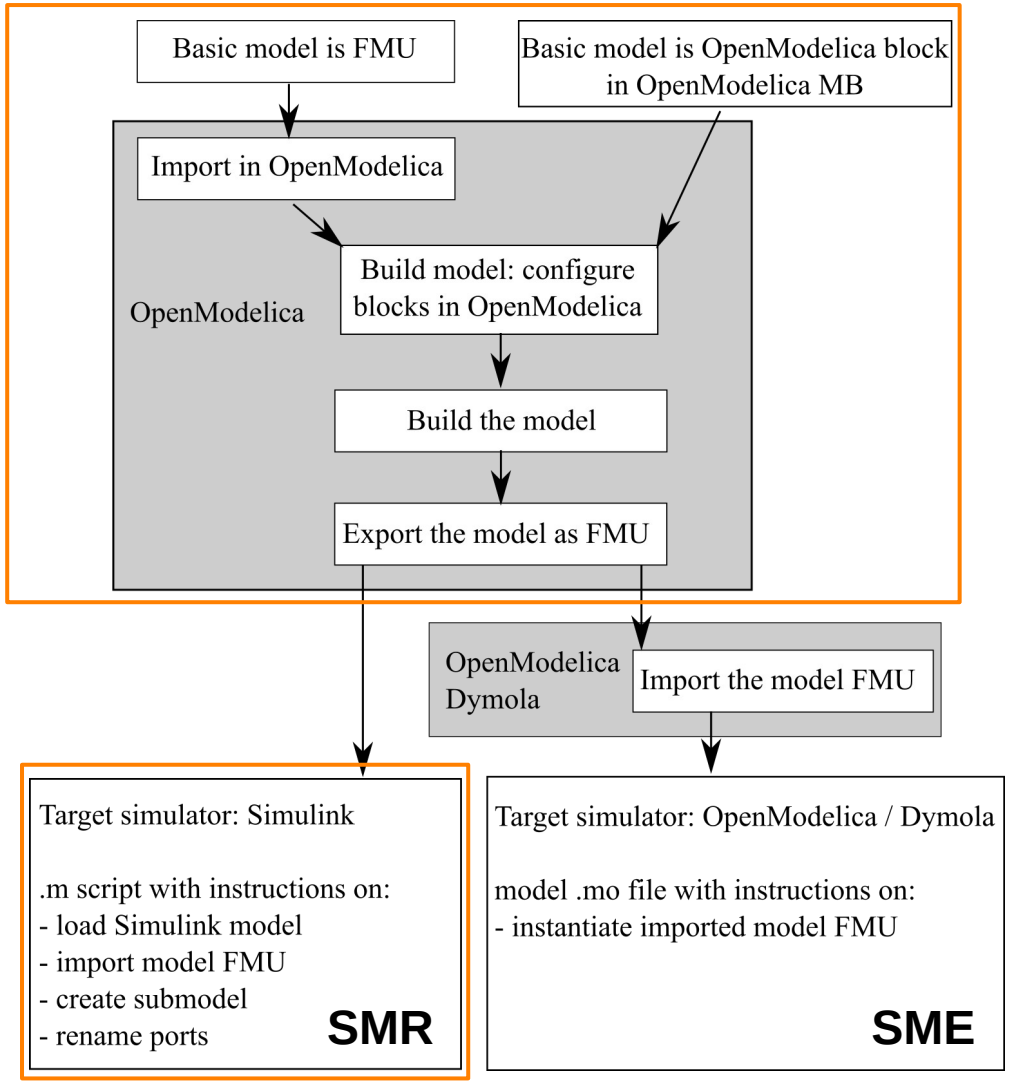
Model Building Using FMI (2) (state of the art)

- **Create simulator specific instructions on how to execute the model FMU**
- *Simulink* models can be created and manipulated with a Matlab script → **"Simulation Model Representation" (SMR)**



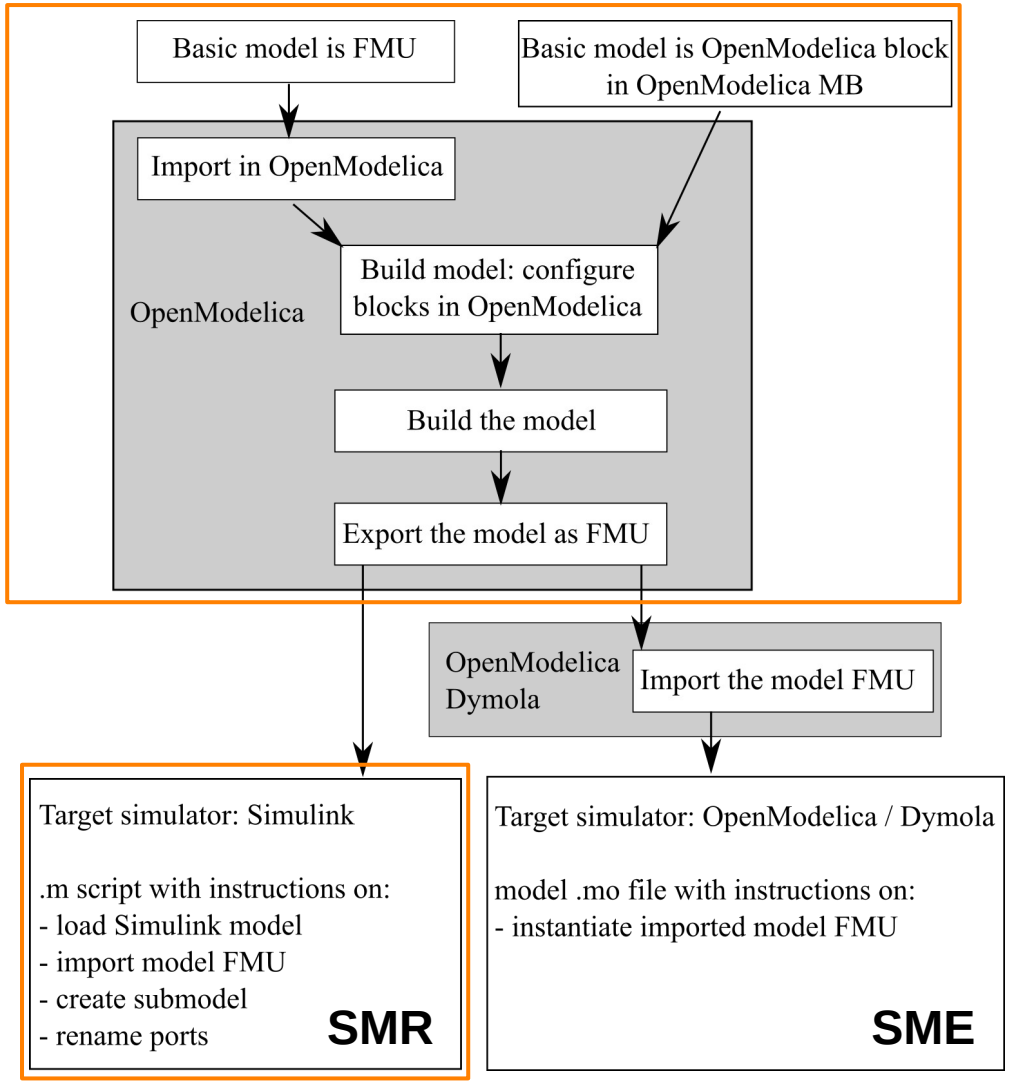
Model Building Using FMI (2) (state of the art)

- **Create simulator specific instructions on how to execute the model FMU**
- *Simulink* models can be created and manipulated with a Matlab script → **"Simulation Model Representation" (SMR)**



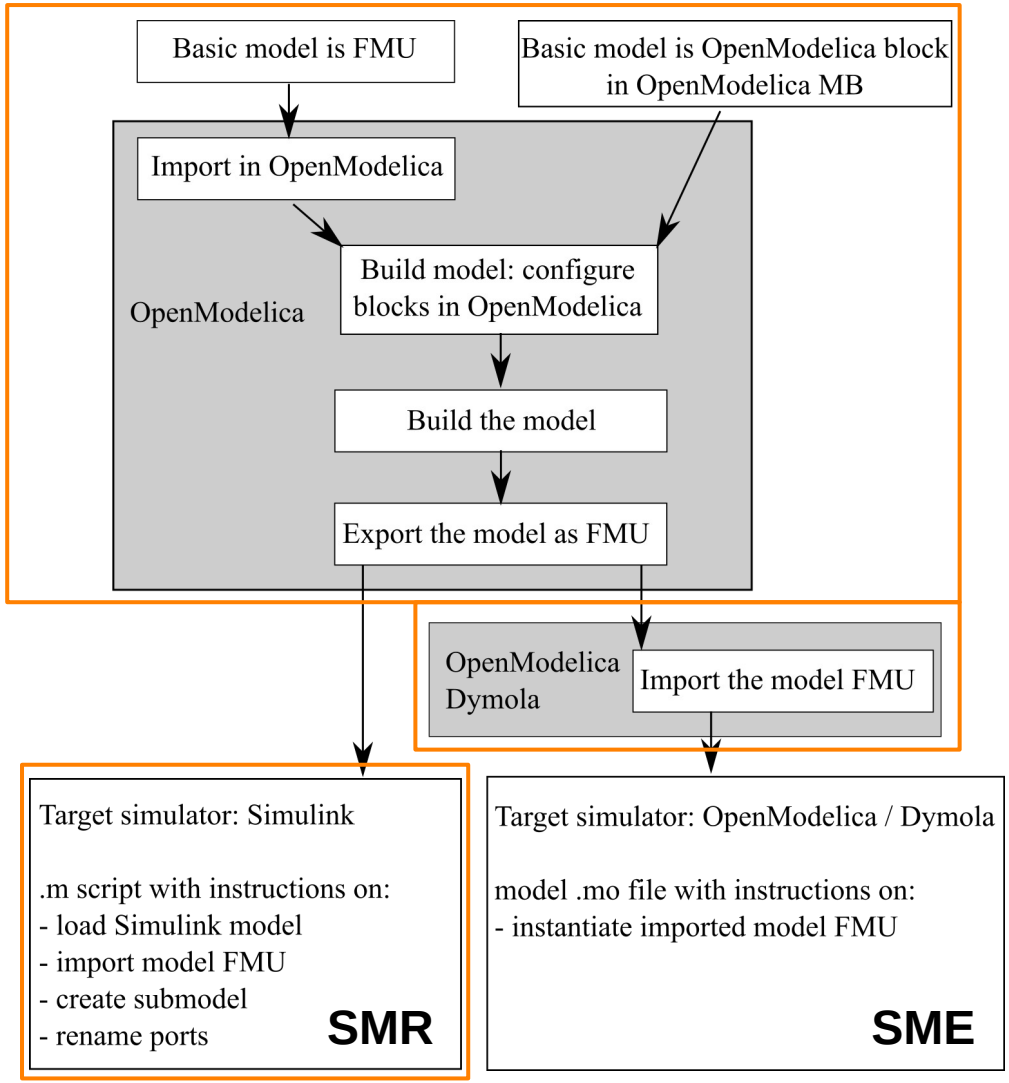
Model Building Using FMI (2) (state of the art)

- **Create simulator specific instructions on how to execute the model FMU**
- *Simulink* models can be created and manipulated with a Matlab script → **"Simulation Model Representation" (SMR)**
- *OpenModelica / Dymola* models are textfiles defining the executable model → **"Simulation Model Executable" (SME)**



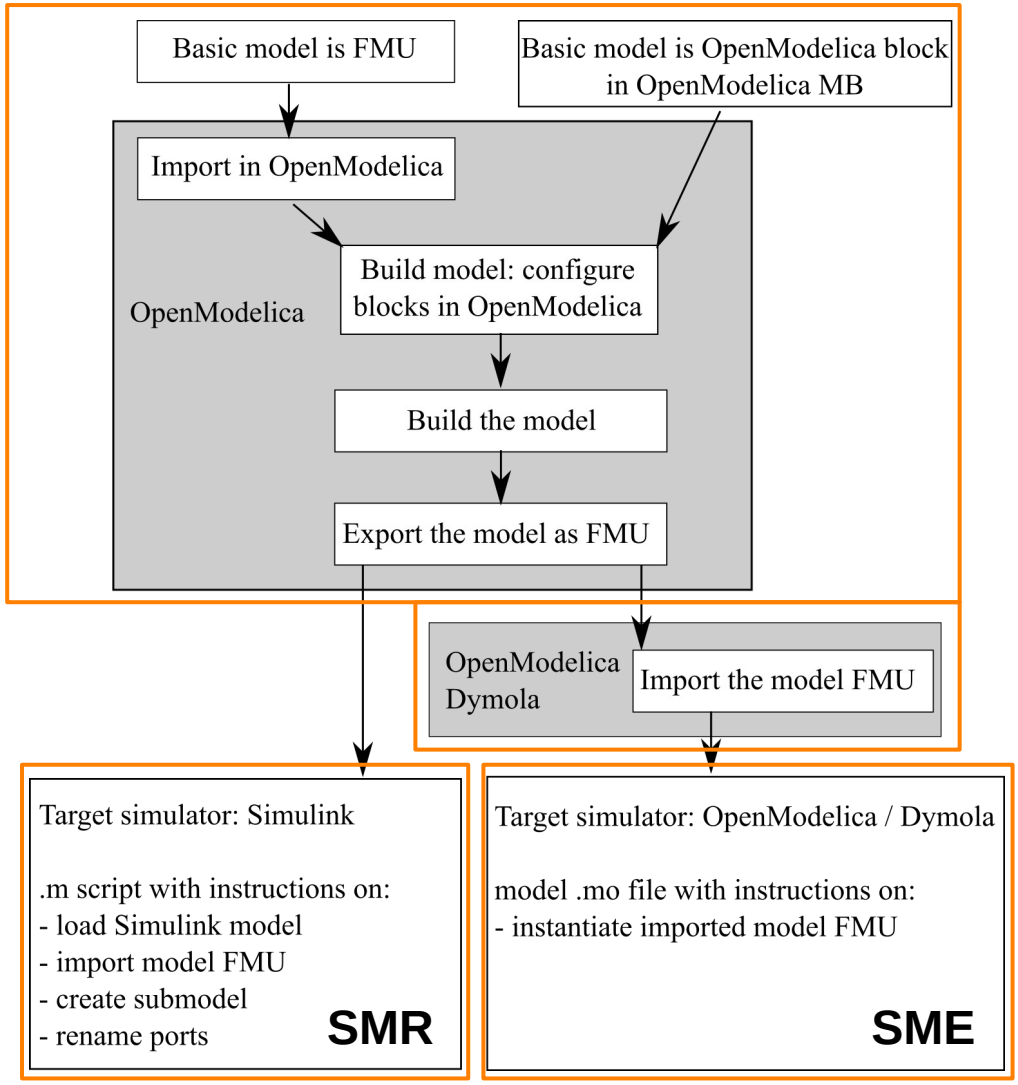
Model Building Using FMI (2) (state of the art)

- **Create simulator specific instructions on how to execute the model FMU**
- *Simulink* models can be created and manipulated with a Matlab script → **"Simulation Model Representation" (SMR)**
- *OpenModelica / Dymola* models are textfiles defining the executable model → **"Simulation Model Executable" (SME)**



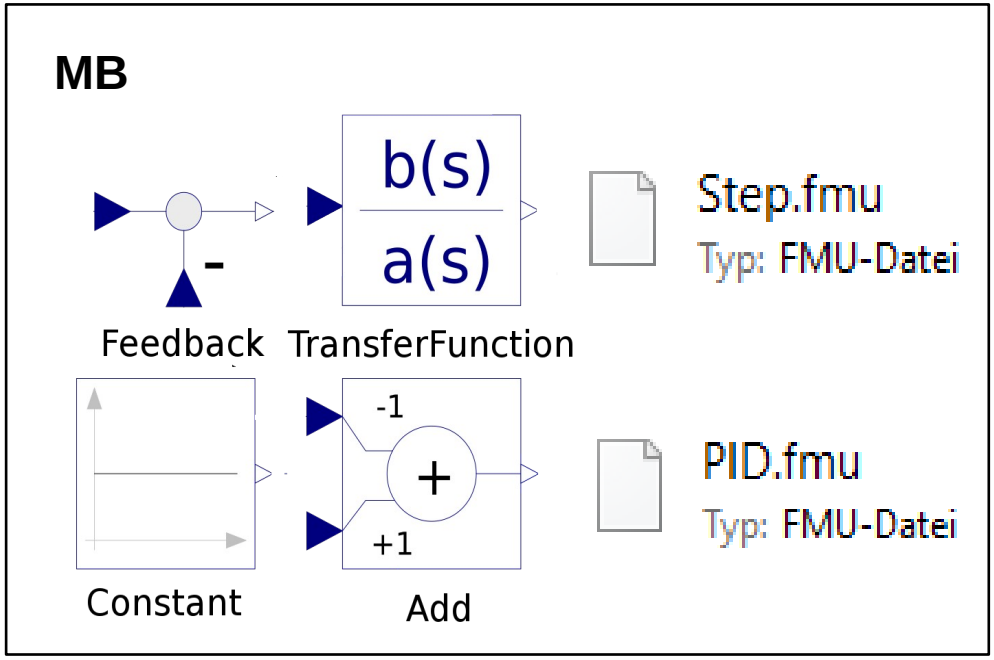
Model Building Using FMI (2) (state of the art)

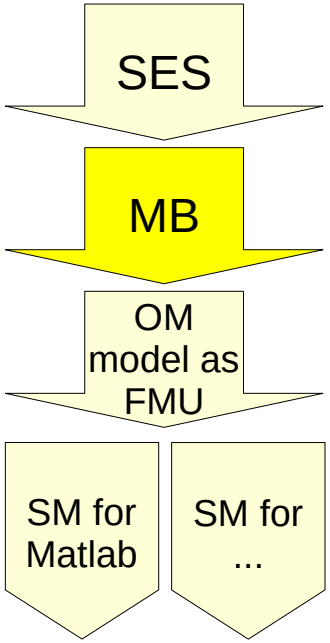
- **Create simulator specific instructions on how to execute the model FMU**
- *Simulink* models can be created and manipulated with a Matlab script → **"Simulation Model Representation" (SMR)**
- *OpenModelica / Dymola* models are textfiles defining the executable model → **"Simulation Model Executable" (SME)**



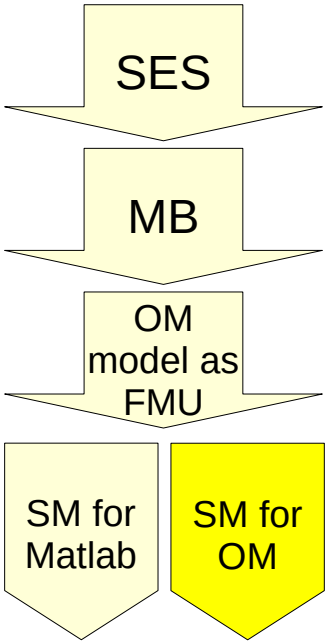
Demonstration of the SES and a Model Base with OpenModelica and FMI Components

- SES nearly like before, but interface is ‘FMI‘ and mb-attributes differ
  → prune, flatten
- Model generation takes a while → presented on the next slides





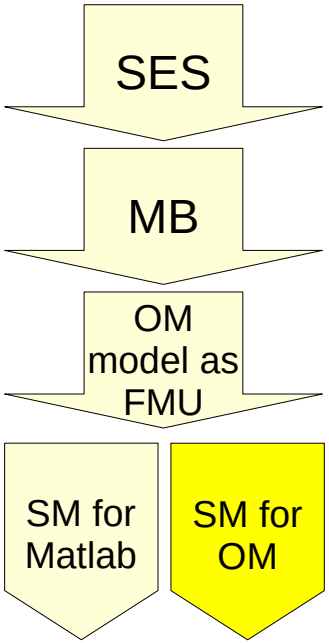
Case Study: FMU Imported in OpenModelica



Case Study: FMU Imported in OpenModelica

- Import the model FMU in the simulator

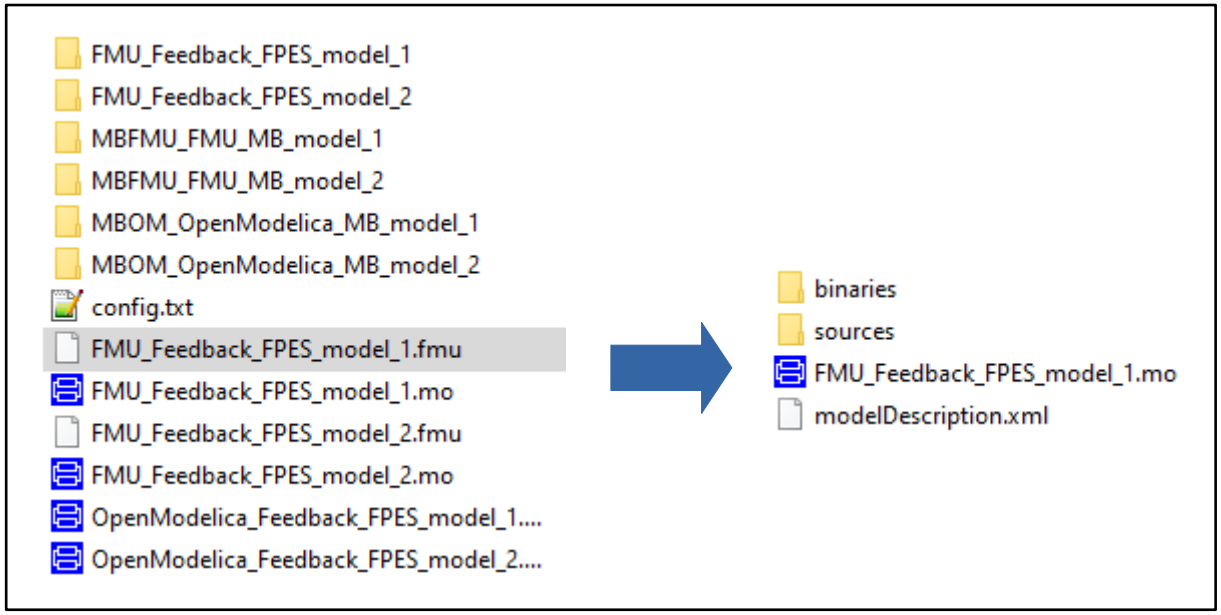
  → model (.mo file) is created

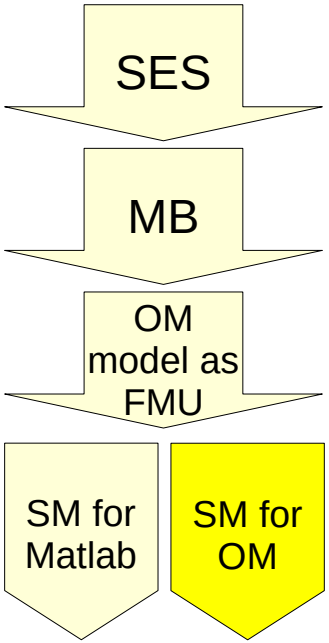


Case Study: FMU Imported in OpenModelica

- Import the model FMU in the simulator

  → model (.mo file) is created

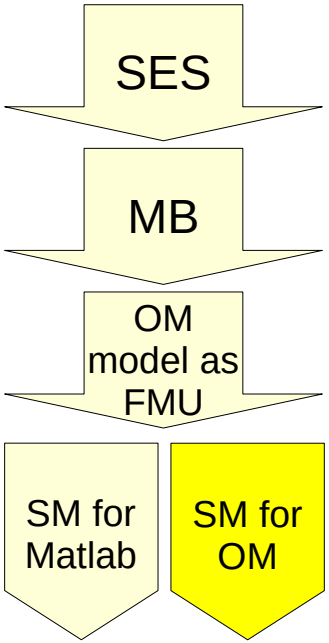




Case Study: FMU Imported in OpenModelica

- Import the model FMU in the simulator

  → model (.mo file) is created

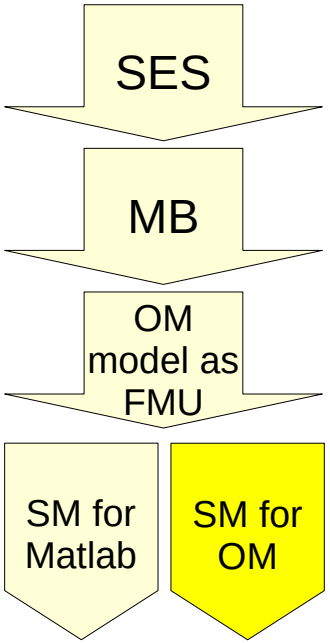


Case Study: FMU Imported in OpenModelica

- Import the model FMU in the simulator

  → model (.mo file) is created

- Create an OpenModelica model:
  - Instantiate imported FMU



Case Study: FMU Imported in OpenModelica

- Import the model FMU in the simulator

  → model (.mo file) is created

- Create an OpenModelica model:
  - Instantiate imported FMU

```
model OpenModelica_Feedback_FPES_model_1
  FMU_Feedback_FPES_model_1 FMU_Feedback_FPES_model_11();
equation
end OpenModelica_Feedback_FPES_model_1;
```