BLENDED MODELING FOR SOFTWARE ARCHITECTURES

TUTORIAL @ ICSA 2023

MARCH 13, 2023

TEAM

MALVINA LATIFAJ
Doctoral student @MDU

MUHAMMAD WASEEM ANWAR
Postdoctoral Researcher@MDU

FEDERICO CICCOZZI
Associate Professor @MDU

KOUSAR ASLAM
Postdoctoral Researcher@VU

IVANO MALAVOLTA
Associate Professor @VU

INTRODUCTION

- Tutorial originating from the work in ITEA4 BUMBLE project

- About BUMBLE settings
  - European industry-driven project (ITEA4 framework)
  - 4 countries (Austria, The Netherlands, Sweden, Turkey)
  - 22 partners (4 universities)
  - Entire market value chain
    - research
    - tool providers
    - tier 1
    - OEMs
    - end-users
  - Nov 2019 - May 2023

BUMBLE

- Blended modeling for Enhanced Software and Systems Engineering

- BUMBLE focuses specifically on industrial-grade innovations to support:
    - Automatic **generation** of blended graphical-textual modeling environments
    - Automatic **synchronization** between multiple modeling notations
    - **Collaborative** modeling
    - Evolution of modeling language definitions and **co-evolution** of blended models

BUMBLE PARTNERS

Austria

AVL List GmbH
Austria

EclipseSource Services GmbH
Austria

Johannes Kepler University Linz
Austria

TU Wien
Austria

WU Vienna
Austria

Netherlands

Canon Production Printing Netherlands B.V.
Netherlands

Modeling Value Group
Netherlands

Sioux Technologies BV
Netherlands

VU University Amsterdam
Netherlands

Sweden

Alten
Sweden

HCL Technologies Sweden AB
Sweden

Mälardalen University
Sweden

Pictor Consulting AB
Sweden

Saab AB
Sweden

Unibap AB
Sweden

University of Gothenburg
Sweden

Volvo Technology AB
Sweden

Türkiye

Ford Otosan
Türkiye

Hermes Iletisim
Türkiye

Mantis
Türkiye

Turkcell Teknoloji
Türkiye

UNIT Information Technologies R&D Ltd.
Türkiye

BUMBLE PROJECT OUTCOMES

1. Concept and open-source implementation for the generation of blended graphical and textual modeling environments

2. Concept and open-source implementation for the synchronization of graphical and textual models

3. Concept and open-source implementation for collaborative modeling

4. Concept and open-source implementation to support co-evolution of the blended modeling environment

eclipse

MPS

BUMBLE PROJECT OUTCOMES

1. Concept and open-source implementation for the **generation** of blended graphical and textual modeling environments

2. Concept and open-source implementation for the **synchronization** of graphical and textual models

3. Concept and open-source implementation for **collaborative** modeling

4. Concept and open-source implementation to support co-evolution of the blended modeling environment

eclipse

MPS

BUMBLE TECHNOLOGIES

Acceleo

Artop
AUTOSAR Tool Platform

HCL RTist

eclipse

capra

eclipse modeling PROJECT

MPS

EATOP
EAST-ADL Tool Platform

emf.

epsilon

CL

UNIFIED MODELING LANGUAGE™ UML®

Sirius

JavaCC™
The Java Parser Generator

OMG SYSTEMS MODELING LANGUAGE SYSML™

emf
ECLIPSE MODELING FRAMEWORK

Xtext

{S} spoofax

Xtend
A language made for Java developers.

ABOUT BLENDED MODELING

BLENDED MODELING

"interact seamlessly with a single model (i.e., abstract syntax) through multiple notations (i.e., concrete syntaxes).." [1]

[1] Ciccozzi, F., Tichy, M., Vangheluwe, H., & Weyns, D. (2019, September). Blended modelling-what, why and how. In 2019 ACM/IEEE 22nd International Conference on Model Driven Engineering Languages and Systems Companion (MODELS-C) (pp. 425-430). IEEE.

Orthogonal to ISO/IEC/IEEE 42010

- At first sight similar or overlapping with viewpoint/view/model of ISO/IEC/IEEE 42010 standard
- ISO/IEC/IEEE 42010 specifies requirements on architecture description frameworks by defining the viewpoint/view/model paradigm
- Blended modeling does *not* focus on identifying viewpoints and related view
- Blended modeling focuses on providing multiple blended editing and visualising notations to interact with a set of concepts

Blended modeling ***orthogonal*** *to ISO/IEC/IEEE 42010*

SOFTWARE ARCHITECTURE

- Blended modeling emerging trend in Model-Driven Engineering for complex software architectures

- Enables modeling of multiple architectural aspects
  - Multiple editing notations
  - Seamlessly, interchangeably, and collaboratively

- To manually architect and build a blended modeling environment is not trivial

- With our solutions we aid architects in designing and semi-automatically generating blended modeling environments for architecting software

STATE OF THE ART

- Existing approaches
  - are orthogonal to blended modeling, e.g. focusing on 42010
  - provide partial blended modeling solutions for specific architectural languages (e.g., OSATE for AADL)
  - do not target certain core blended modeling aspects
    - mechanisms to generate graphical and textual notations automatically
    - higher-order transformations for generalisability of synchronisation and migration aspects

Our goal

unified, automated and customizable solution applicable to multiple languages and application domains

TAKE-AWAY MESSAGES

*Blended modeling is **orthogonal** to ISO/IEC/IEEE 42010*

*Blended modeling is apt to **simplify** architecting of software systems*

*Architecting blended modeling environments is **not as hard** with the appropriate support*

WHAT WE WILL SHOW

- Architecting a blended modeling environment in Eclipse
  - Definition of mapping rules between graphical and textual notations
  - Generation of graphical and textual editors
  - Generation of the synchronisation mechanisms via higher-order transformations
  - Exploration of the resulting blended modeling environment

- Usage of the generated environment
  - modeling of software architectures via graphical and textual notations
  - Synchronisation of model changes across notations
  - Collaborative interactive model editing

AGENDA

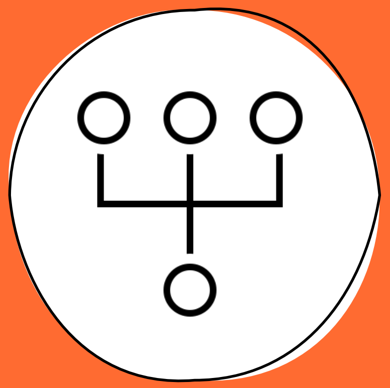
TIME	ACTIVITY
13:30 - 14:00	Introduction
14:00 - 14:50	Presentation on generation and use of blended modeling environments
14:50 - 15:00	Break
15:00 - 15:50	Demo on generation and use of blended modeling environments
15:50 - 16:20	Presentation on collaborative blended modeling
16:20 - 16:30	Demo on collaborative blended modeling
16:30 - 17:00	Break
17:00 - 17:45	Facilitated discussion (focus groups)

PART 1

ARCHITECTING A **BLENDED MODELING** ENVIRONMENT IN ECLIPSE

MOTIVATION

MUTUALLY EXCLUSIVE USE OF NOTATIONS* IN MODELING TOOLS



GRAPHICAL

TEXTUAL

Each notations has unique benefits.

Limits communication between stakeholders.

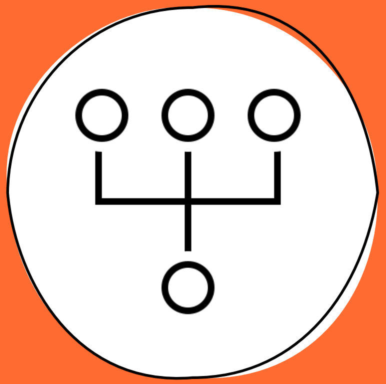
Different tasks require different notations.

Negatively affects the throughput of engineers.

* *"Notation" and "concrete syntax" are used interchangeably, as they both refer to the way in which a modeling language is represented visually or textually.*

MOTIVATION

SEAMLESS USE OF BOTH NOTATIONS



GRAPHICAL

TEXTUAL

Benefits of both graphical and textual notations.

Better intra- and inter-disciplinary communication.

Flexible separation of concerns.

Faster modeling activities.

DEFINITION

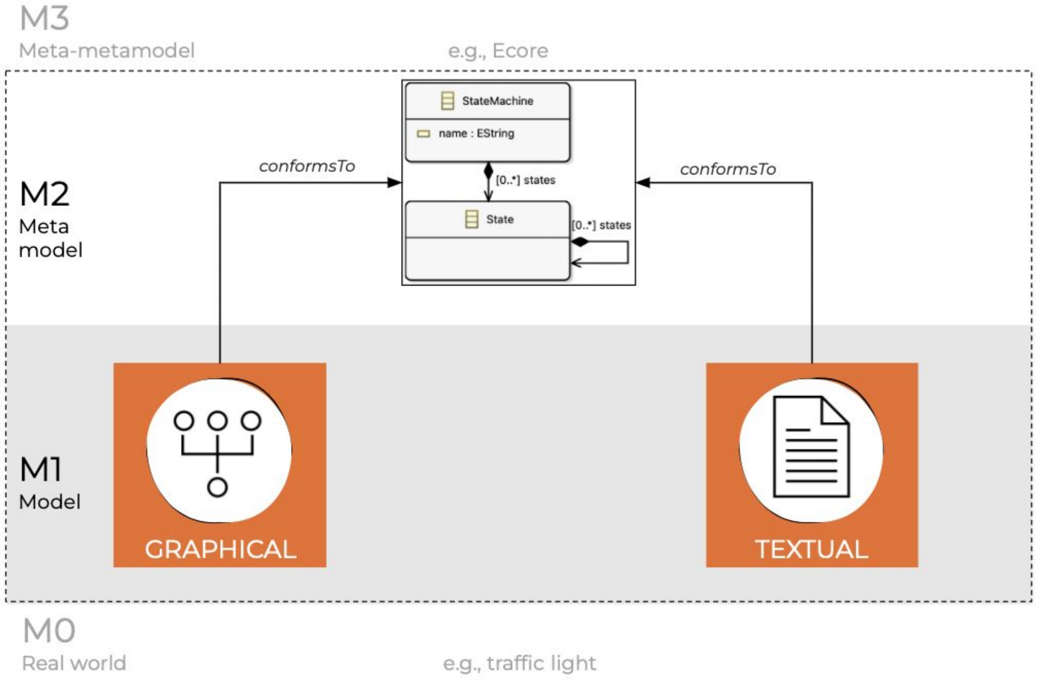
BLENDED MODELING

"interact seamlessly with a single model (i.e., abstract syntax) through multiple notations (i.e., concrete syntaxes).." [1]

[1] Ciccozzi, F., Tichy, M., Vangheluwe, H., & Weyns, D. (2019, September). Blended modelling-what, why and how. In 2019 ACM/IEEE 22nd International Conference on Model Driven Engineering Languages and Systems Companion (MODELS-C) (pp. 425-430). IEEE.

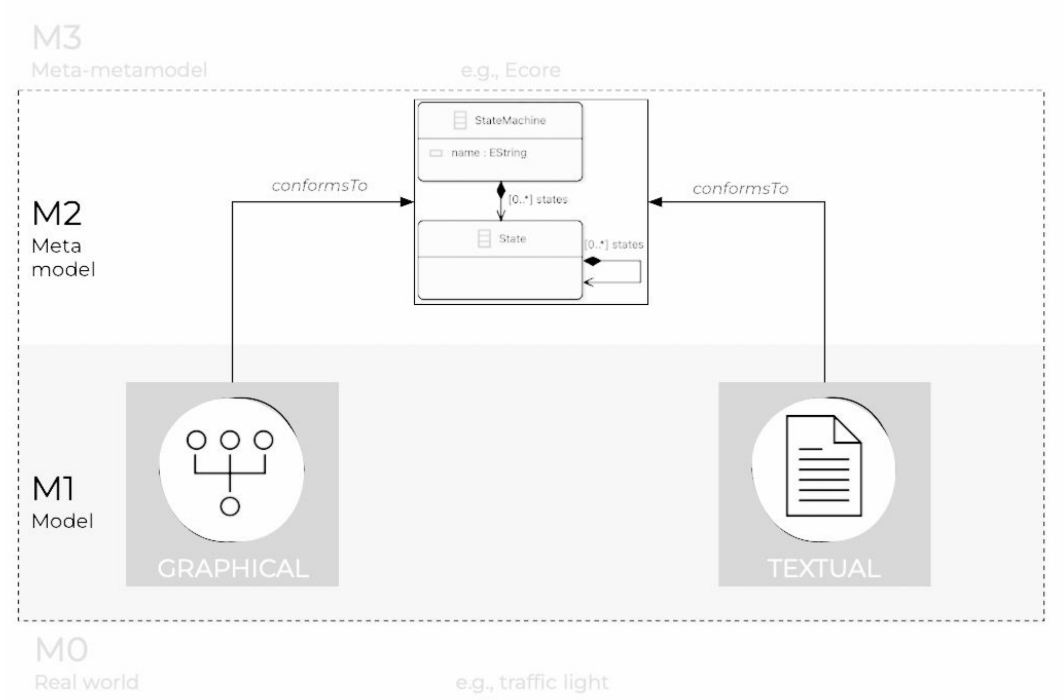
BLENDED MODELING

CLASSICAL BLENDED MODELING

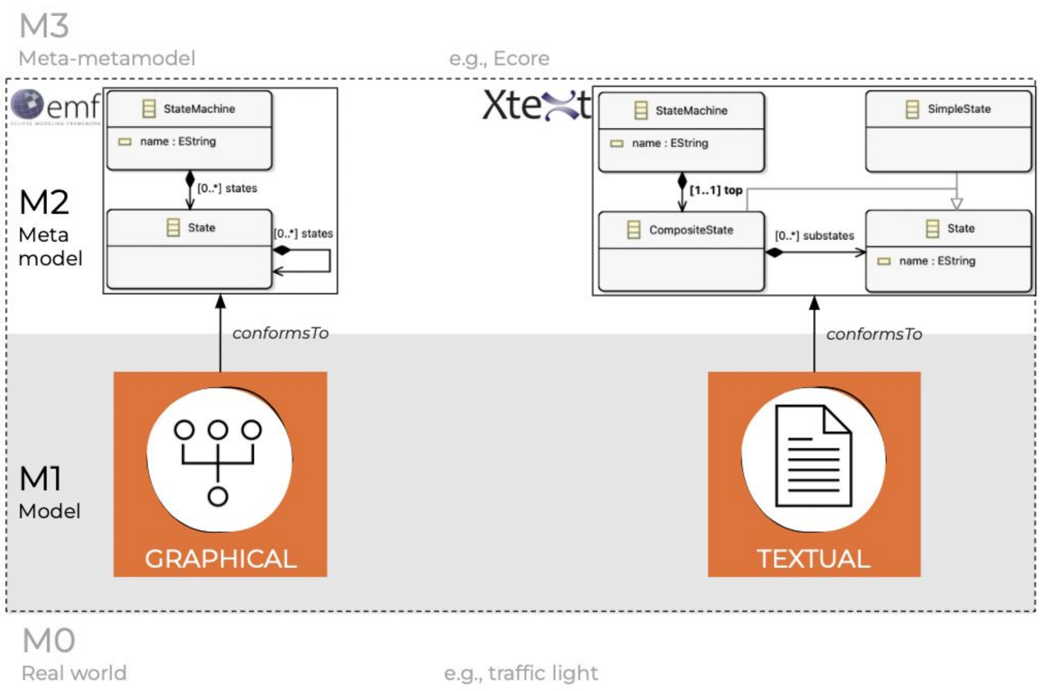


BLENDED MODELING

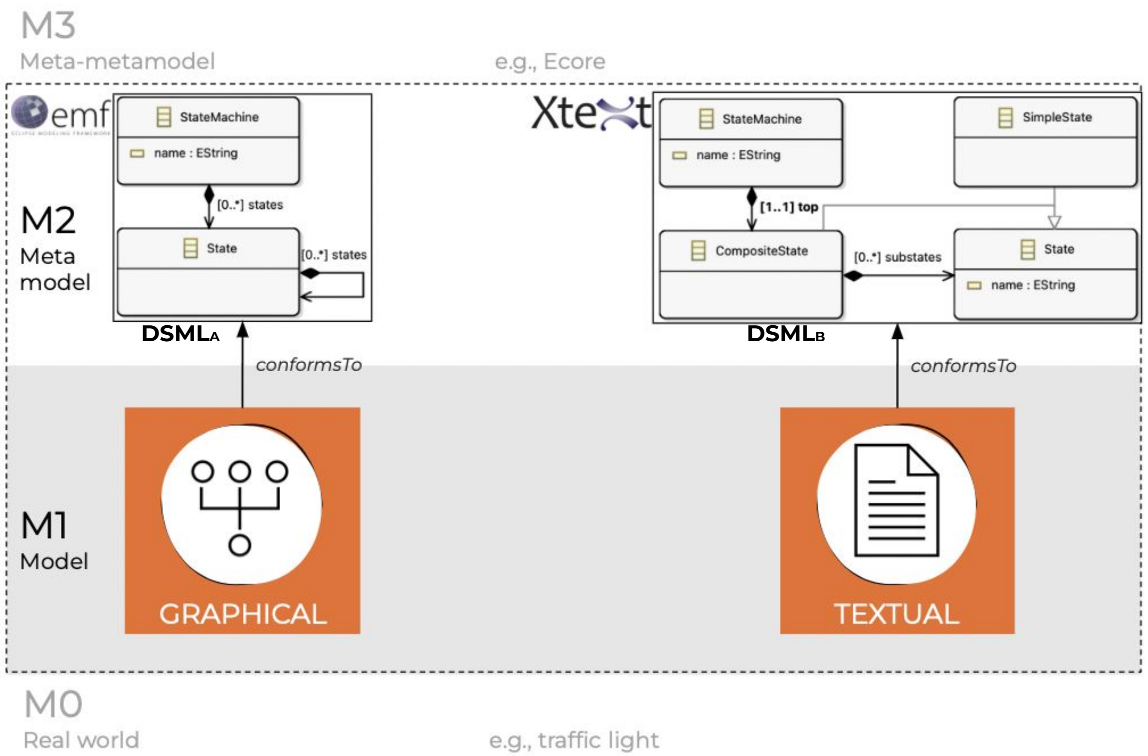
CLASSICAL BLENDED MODELING



EXTENDED BLENDED MODELING



EXTENDED BLENDED MODELING



"a single underlying language (set of concepts) formalized through ***multiple abstract syntaxes****"*

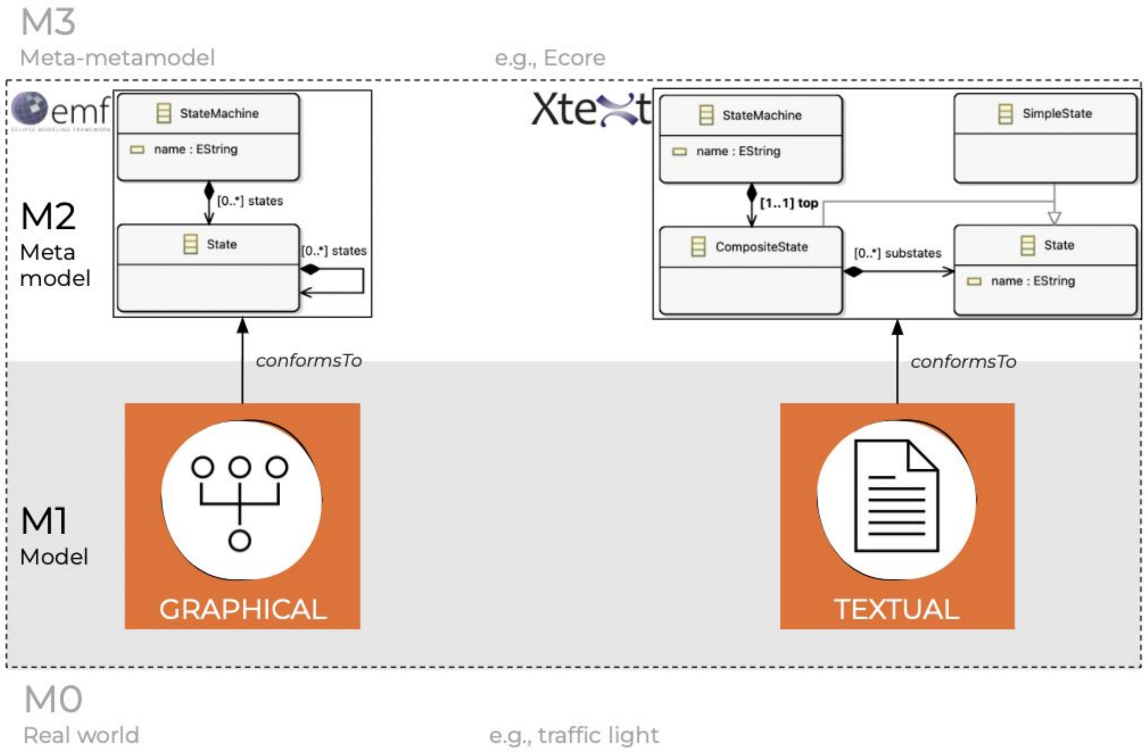
Scenario 1
DSMLA and DSMLB represent two notations of the same language (e.g., graphical and textual UML-RT state-machines), thus we support blended modeling across different notations of the same language.

Scenario 2
DSMLA and DSMLB are disjoint, thus we support blended modeling across different notations of different languages.

DEFINITION OF EDITORS

Sirius

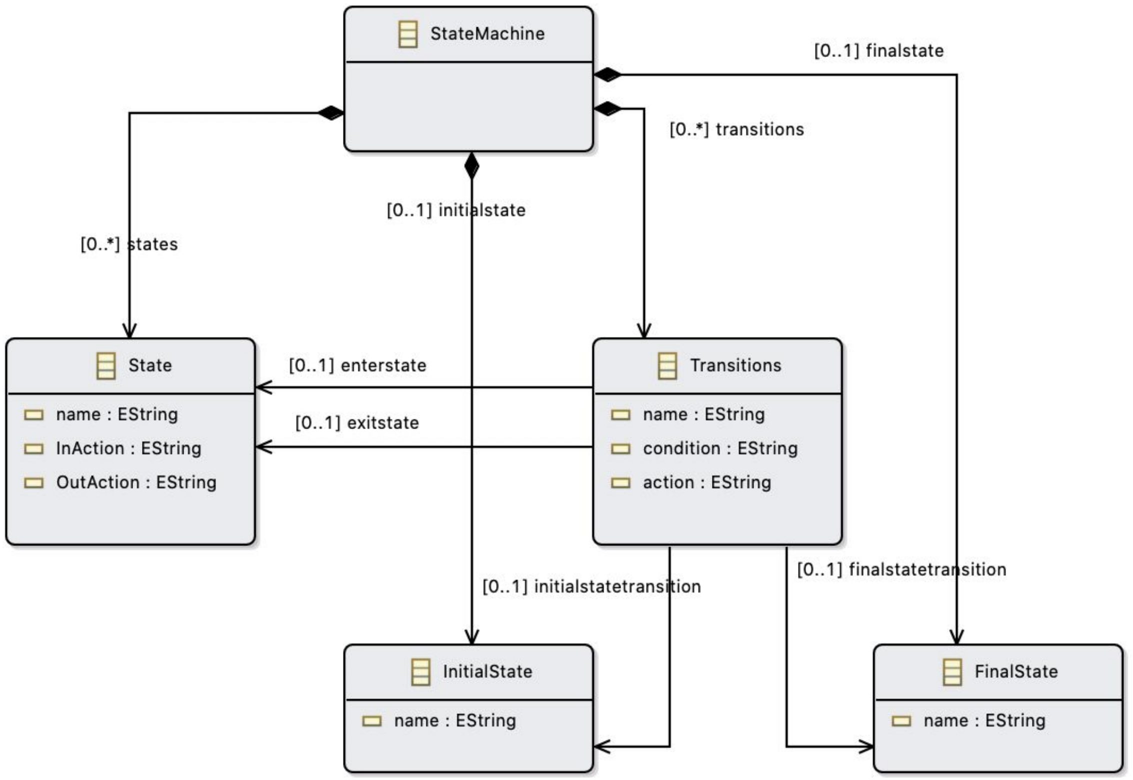
Open-source framework used to create graphical modeling workbenches for your DSL.

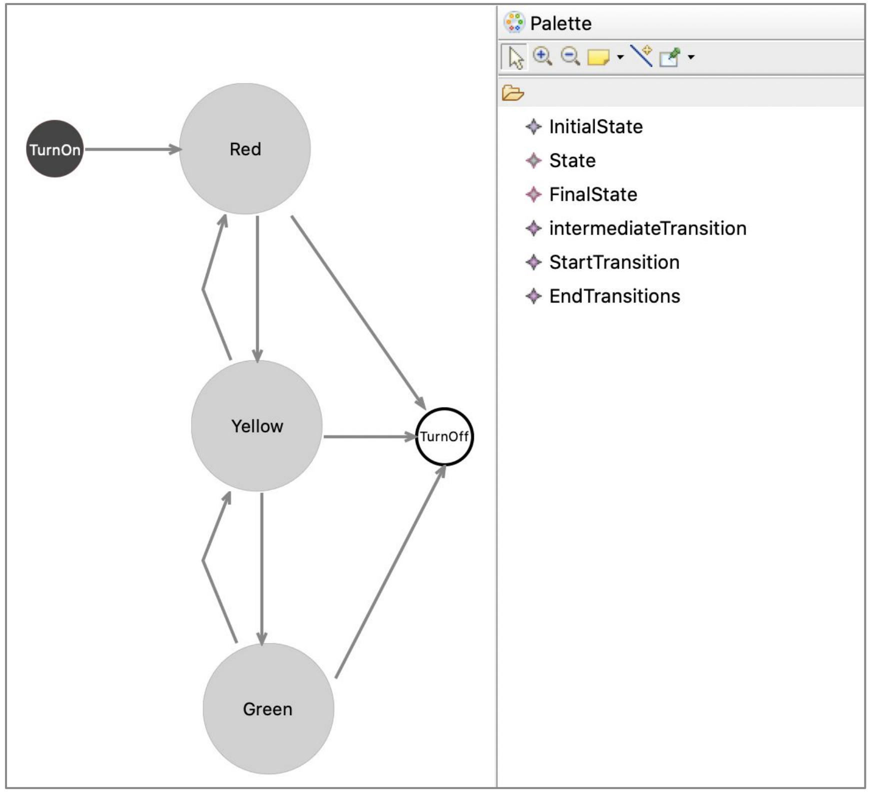


Xtext

Open-source framework for the development of textual DSLs.

DEFINITION OF GRAPHICAL EDITORS

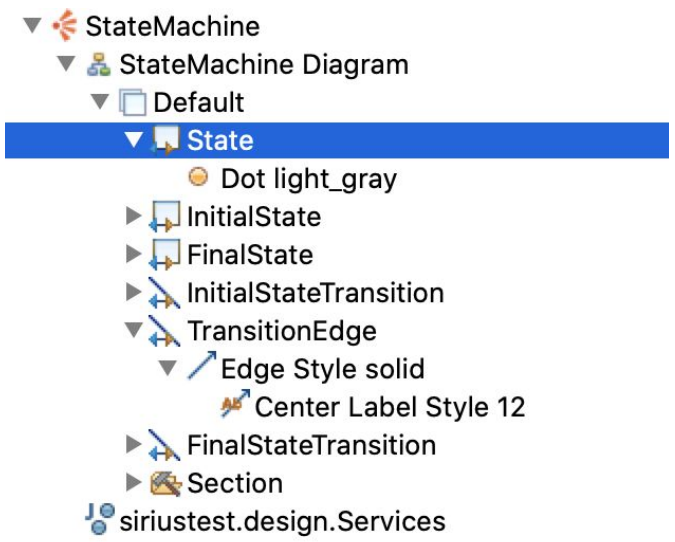


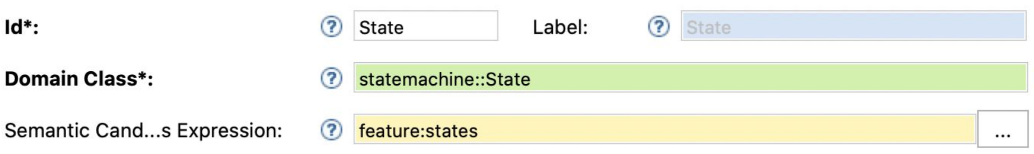


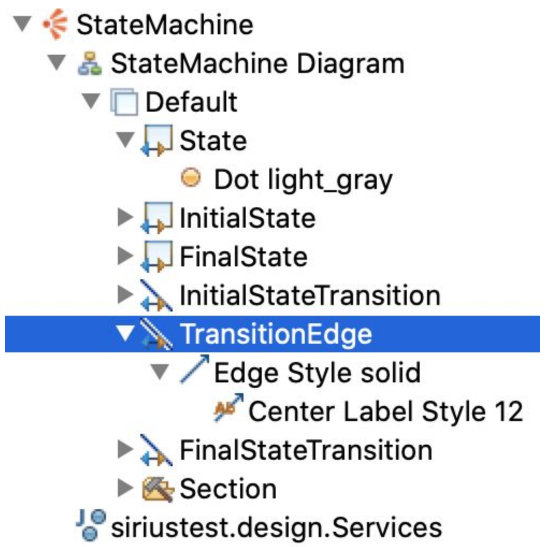
Sirius

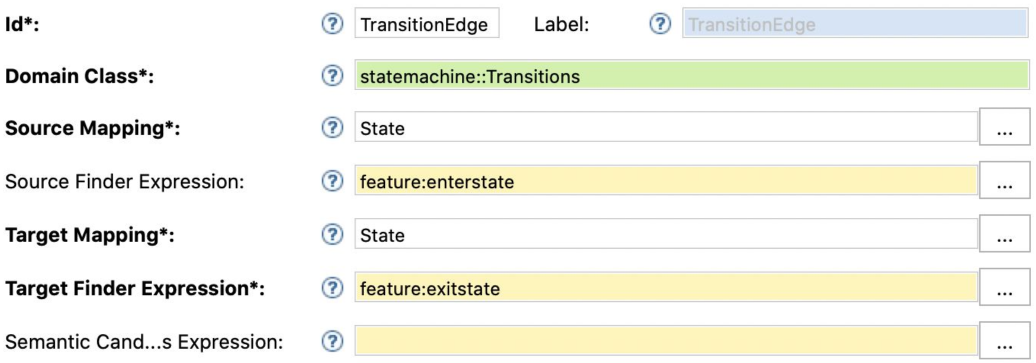
define graphical representations and map them to Ecore model elements

DEFINITION OF GRAPHICAL EDITORS









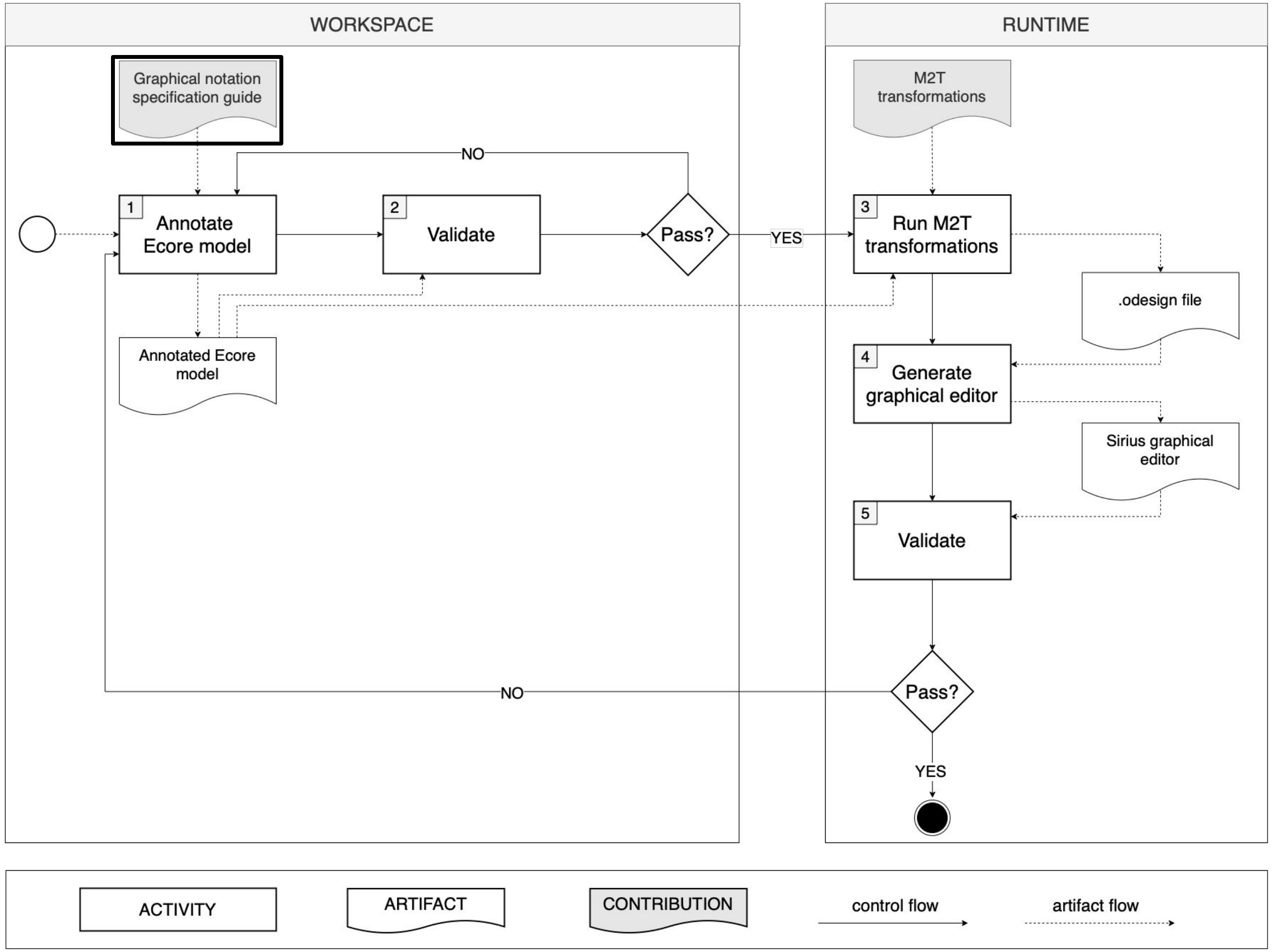
CHALLENGES

1. Error-prone and tedious process of creating the graphical editor.

2. Time-consuming navigation through menus.

3. Updating the configuration file after changes to the metamodel may require manual effort, including fixing errors and updating graphical representations, which can be time-consuming and tedious.

GENERATION OF GRAPHICAL EDITORS



GRAPHICAL NOTATION SPECIFICATION GUIDE

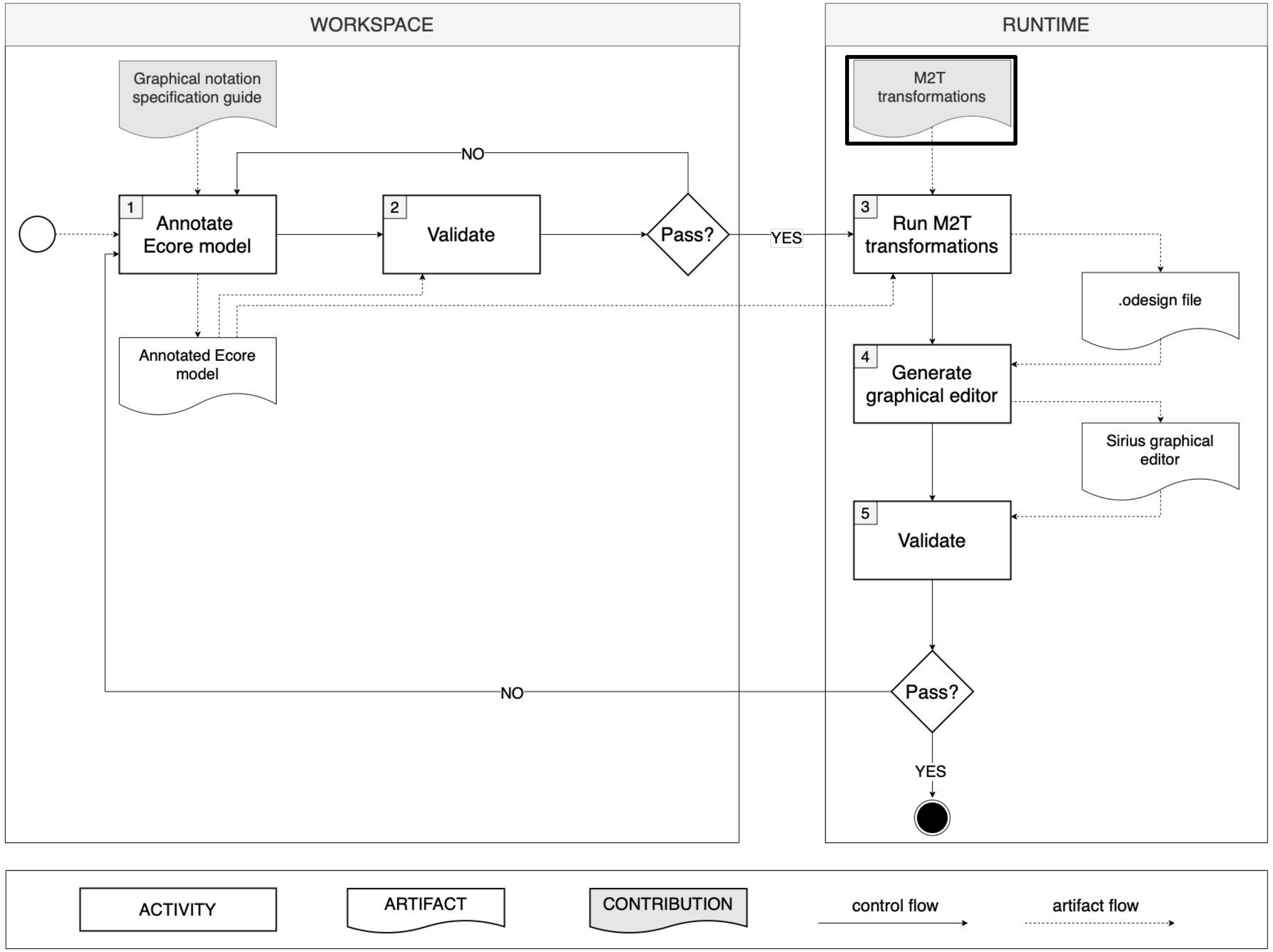
1. Identify a set of concepts that are frequently used during the creation of graphical editors.

2. Define a graphical notation specification guide that allows you to define these concepts in the form of EAnnotations.

```
class StateMachine {
   attr String name;
   @sirius.container(name="State",expression=
"feature:states", background="purple", foreground="white",
container="default", domain="State", icon="false")
   @sirius.tool(name="CreateState",
type="ContainerCreationDescription", mappings="State,
ContainedState:State", reference="states",
belongsto="State")
   val State[*] states;
```

Implementation of the approach can be found in our repository.

GENERATION OF GRAPHICAL EDITORS



MODEL TO TEXT TRANSFORMATIONS

1. Define model to text transformation that take as input the annotated metamodel.
2. Generate the xml version of the .odesign file.

Implementation of the approach can be found in our repository.

ADVANTAGES

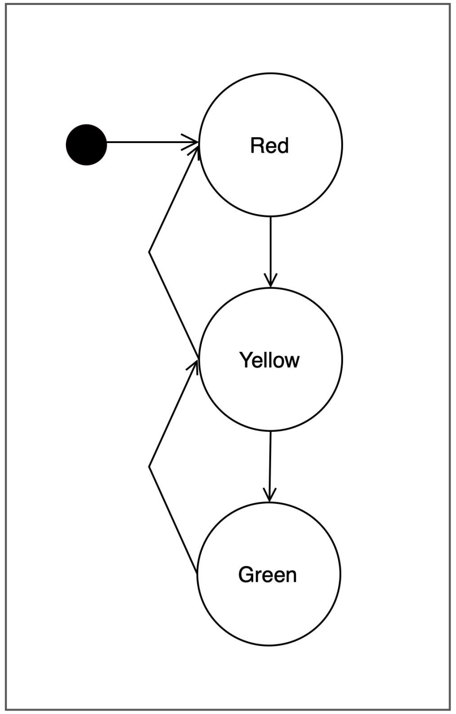
1. Provides a complete overview of the definition of both the abstract and concrete syntax in one single file.

2. Facilitates co-evolution of the concrete syntax in response to changes made to the abstract syntax.

3. It can facilitate the creation of graphical editors for both inexperienced and seasoned developers.

SYNCHRONIZATION

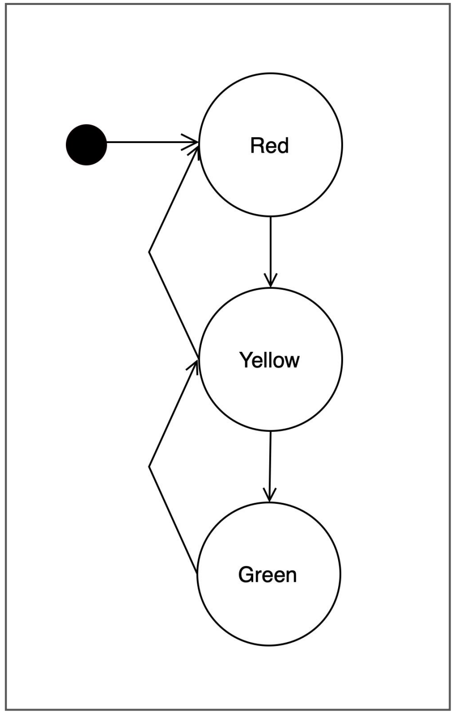
STATE MACHINES – TRAFFIC LIGHT EXAMPLE




```
StateMachine Traffic Light {
  InitialState Start,
  IntermediateState Red,
  IntermediateState Yellow,
  IntermediateState Green,

  Transition "startToRed"
  when "timer=30"
  Start to Red,
  Transition "redToYellow"
  when "timer=30"
  Red to Yellow,
  Transition "yellowToGreen"
  when "timer=30"
  Yellow to Green,
  Transition "greenToYellow"
  when "timer=30"
  Green to Yellow,
  Transition "yellowToRed"
  when "timer=30"
  Yellow to Red
}
```

SYNCHRONIZATION

STATE MACHINES – TRAFFIC LIGHT EXAMPLE




```
StateMachine Traffic Light {
  InitialState Start,
  IntermediateState Red,
  IntermediateState Yellow,
  IntermediateState Green,

  Transition "startToRed"
  when "timer=30"
  Start to Red,
  Transition "redToYellow"
  when "timer=30"
  Red to Yellow,
  Transition "yellowToGreen"
  when "timer=30"
  Yellow to Green,
  Transition "greenToYellow"
  when "timer=30"
  Green to Yellow,
  Transition "yellowToRed"
  when "timer=30"
  Yellow to Red
}
```

SYNCHRONIZATION

TASK FOR ENGINEERS

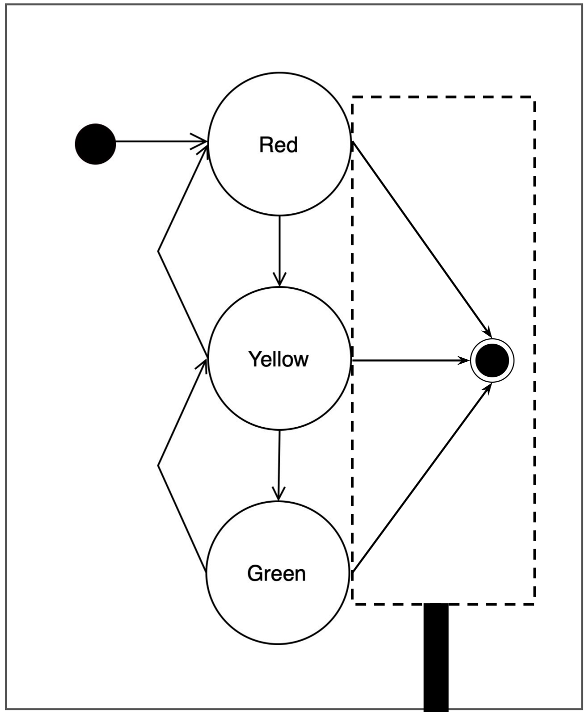
Define model transformations using transformation languages.

SYNCHRONIZATION

```
modeltype hclScope uses 'http://www.xtext.org/example/hclscope/HclScope';
modeltype statemach uses 'http://www.eclipse.org/papyrusrt/xtumlrt/statemach';

transformation Textual2Graphical( in hclScopeModel:hclScope, out statemachModel:statemach);
main() {
hclScopeModel.rootObjects()[hclScope::StateMachine] -> map StateMachine2StateMachine();
}
mapping hclScope :: StateMachine :: StateMachine2StateMachine() : statemach :: StateMachine
{
    if (self.states -> size() = 1){
        result.top := self.states -> first().map State2CompositeState();
        result.name := self.name;
    };
    if (self.states -> size() != 1){
        var CompositeStateObject := object statemach :: CompositeState{
        substates := self.states -> map State2StateDisjunct();
    };
    top := CompositeStateObject;
        result.name := self.name;
    };
}
mapping hclScope :: State :: State2CompositeState() : statemach :: CompositeState
inherits hclScope :: State :: AbstractState2State
when {not(self.states -> isEmpty())}
{
    result.substates := self.states -> map State2StateDisjunct();
}
mapping hclScope :: State :: State2SimpleState() : statemach :: SimpleState
inherits hclScope :: State :: AbstractState2State
when {self.states -> isEmpty() }
{
}

mapping hclScope :: State :: State2StateDisjunct() : statemach :: State
disjuncts hclScope :: State :: State2CompositeState, hclScope :: State :: State2SimpleState{}

abstract mapping hclScope :: State :: AbstractState2State() : statemach :: State
{
    result.name := self.name;
}
```

SYNCHRONIZATION

STATE MACHINE METAMODEL EVOLUTION – TRAFFIC LIGHT EXAMPLE



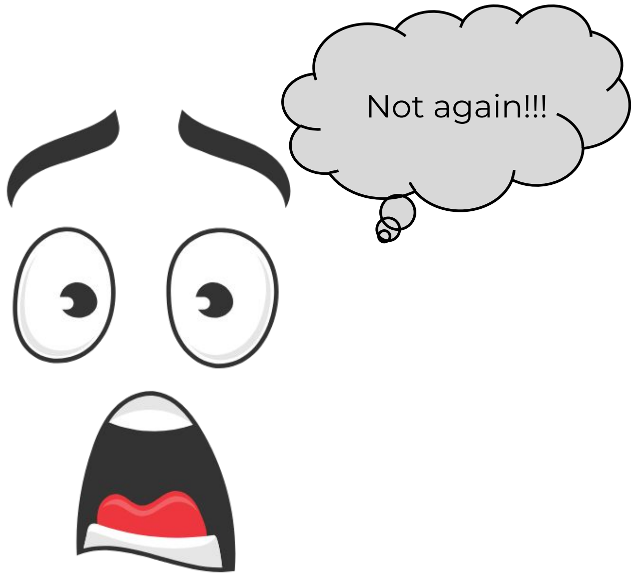

```
StateMachine Traffic Light {
  InitialState Start,
  IntermediateState Red,
  IntermediateState Yellow,
  IntermediateState Green,

  Transition "startToRed"
  when "timer=30"
  Start to Red,
  Transition "redToYellow"
  when "timer=30"
  Red to Yellow,
  Transition "yellowToGreen"
  when "timer=30"
  Yellow to Green,
  Transition "greenToYellow"
  when "timer=30"
  Green to Yellow,
  Transition "yellowToRed"
  when "timer=30"
  Yellow to Red
}
```

New concepts added to the underlying metamodel (metamodel evolution)

SYNCHRONIZATION

TASK FOR ENGINEERS

Continuosly evolve the model transformations according to metamodel changes.



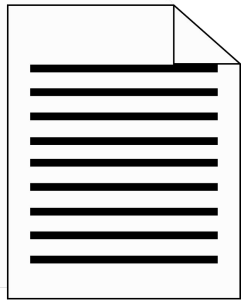
GOAL

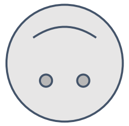
automate the engineering of synchronization transformations across multiple notations for blended modeling

REQUIRED INGREDIENTS

LEVEL OF
ABSTRACTION

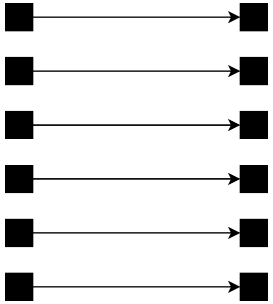
MODEL 2 MODEL
TRANSFORMATIONS





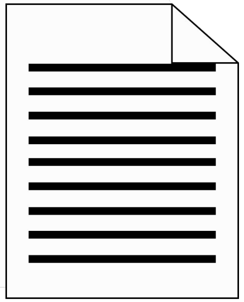
REQUIRED INGREDIENTS

MAPPING

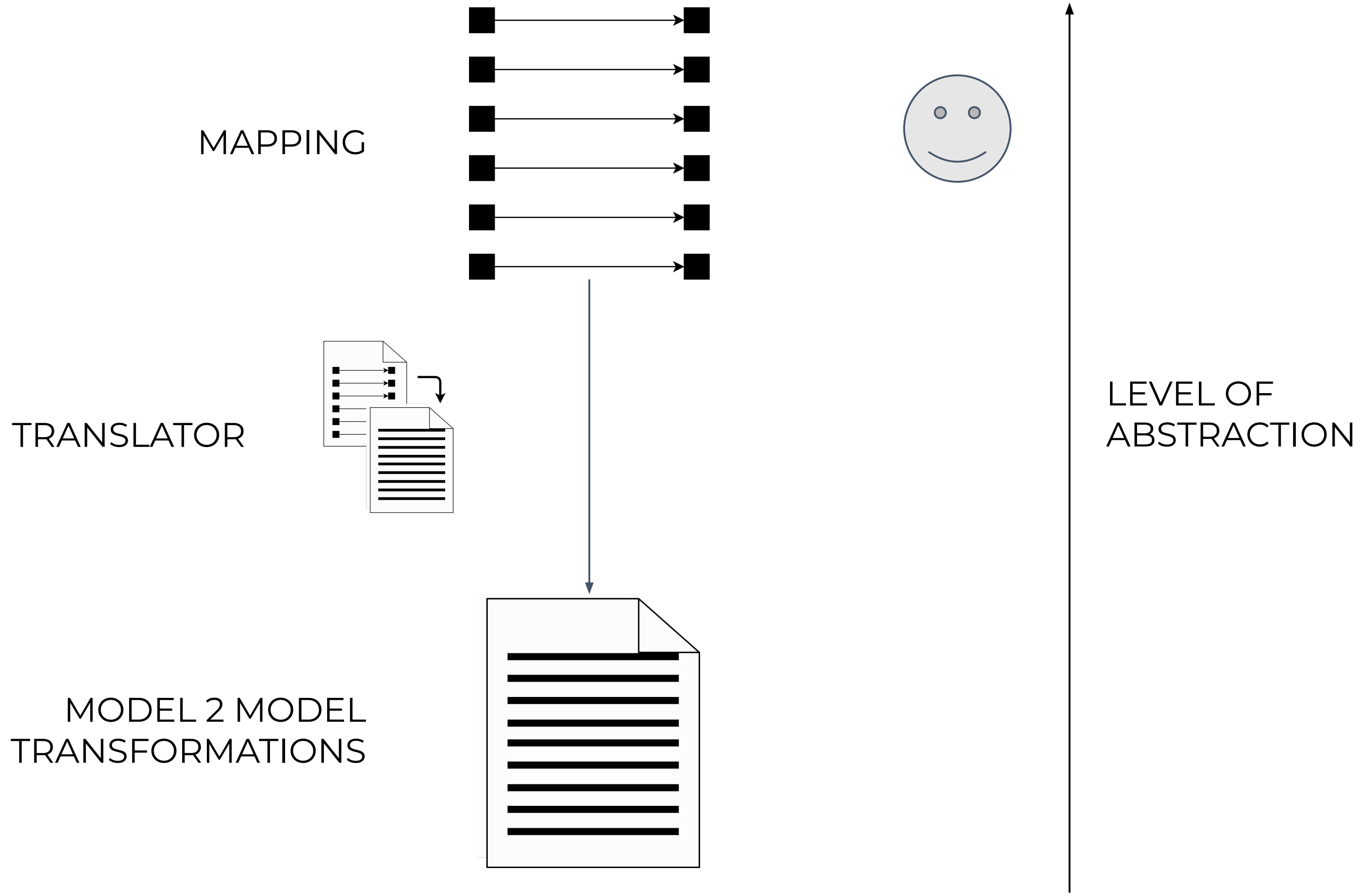


LEVEL OF
ABSTRACTION

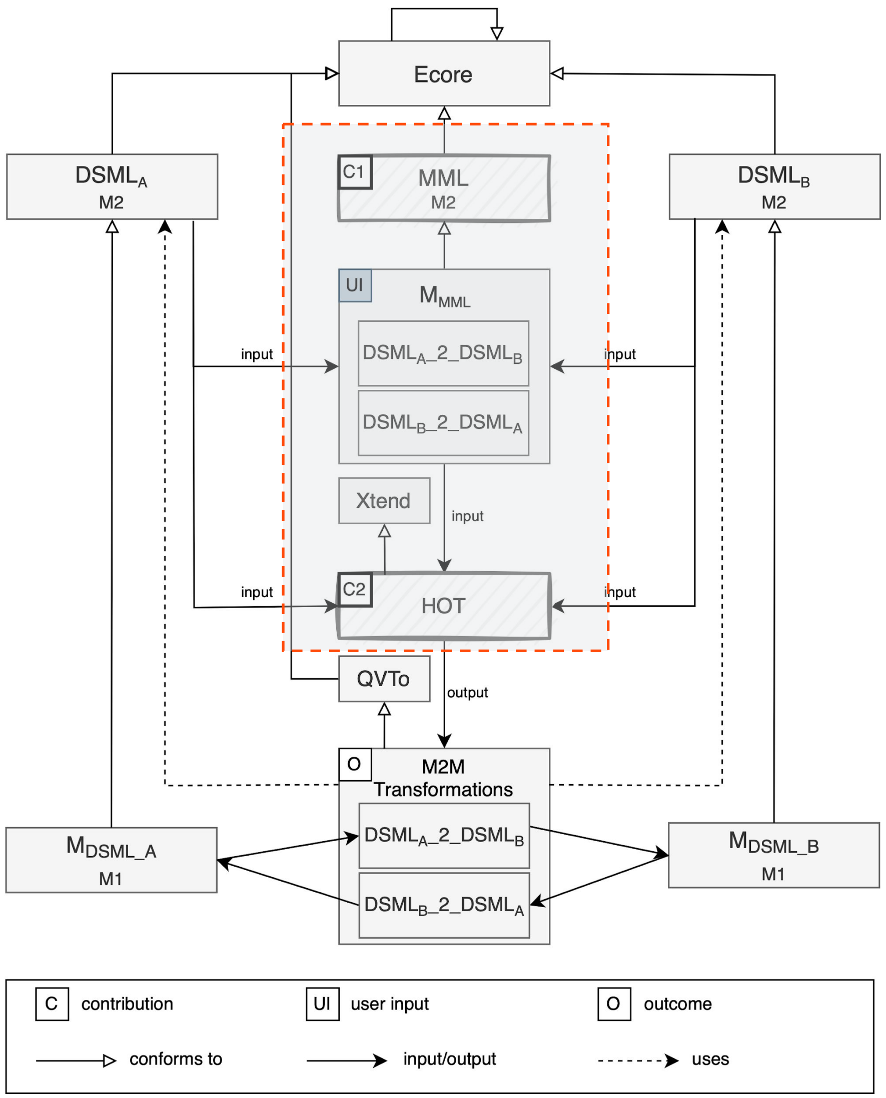
MODEL 2 MODEL
TRANSFORMATIONS



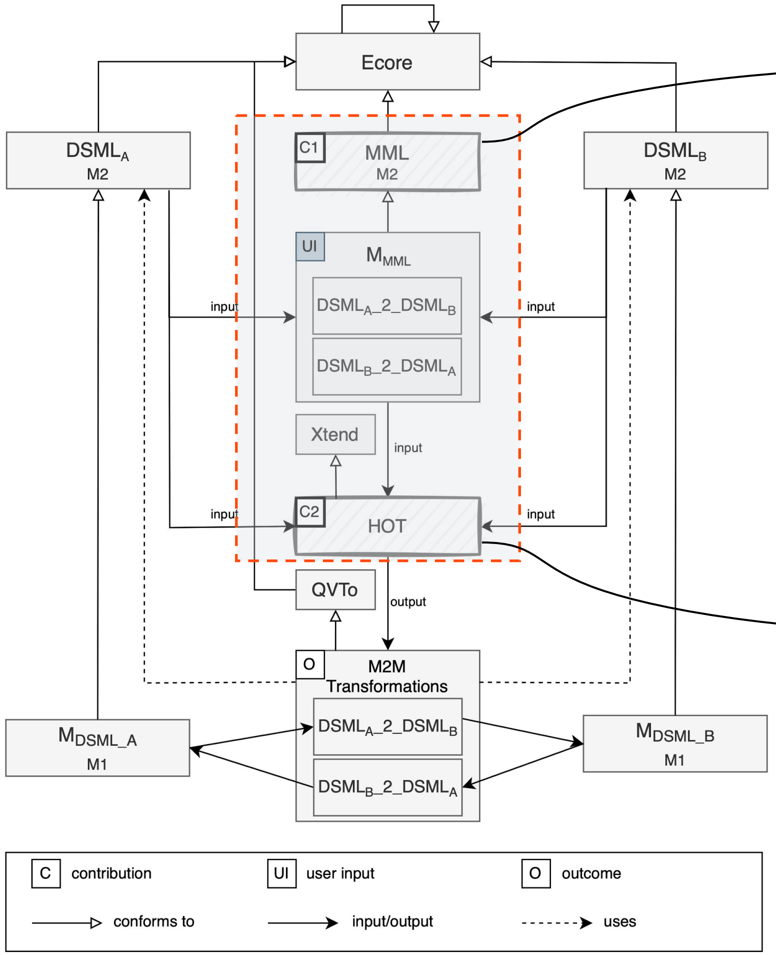
REQUIRED INGREDIENTS

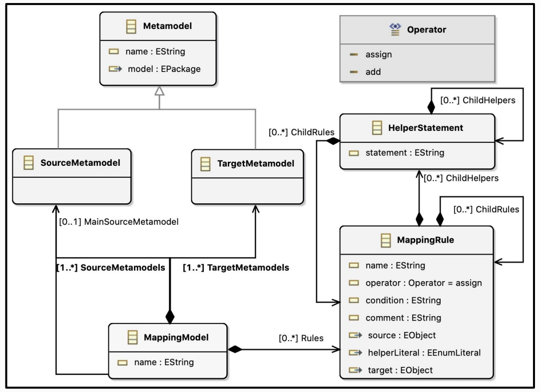


OVERALL ARCHITECTURE



APPROACH





MAPPING MODELING LANGUAGE (MML)

Textual and tree-based editors.

Syntactical resemblance to object oriented programming languages.

Minimum set of concepts required for defining deterministic mappings.

Focus on domain logic rather than lower level model transformations.

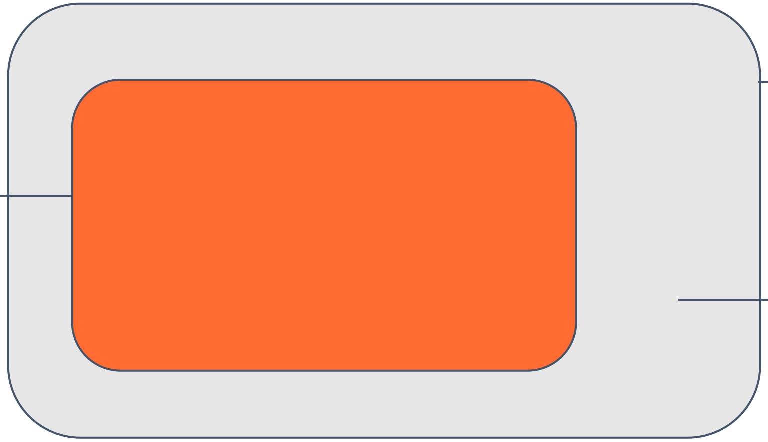
HIGHER-ORDER TRANSFORMATIONS (HOTs)

Model transformation that take as input a high-level mapping between two DSMLs and generate QVTo model transformations.

Implementation of the approach can be found in our repository.

DEFINING MML

1. Identify the required input.
2. Identify the maximum set of information that could be automatically retrieved from DSMLs.
3. Identify the information that should be manually inputted by the user.



mapping_input

automatic_input

automatic_input ⊂ mapping_input

{abstract/non-abstract, disjunctive/disjuncted, invocation of mapping rules, assignment operator, , inheritance, navigation operators, navigation path, mapping rule name }

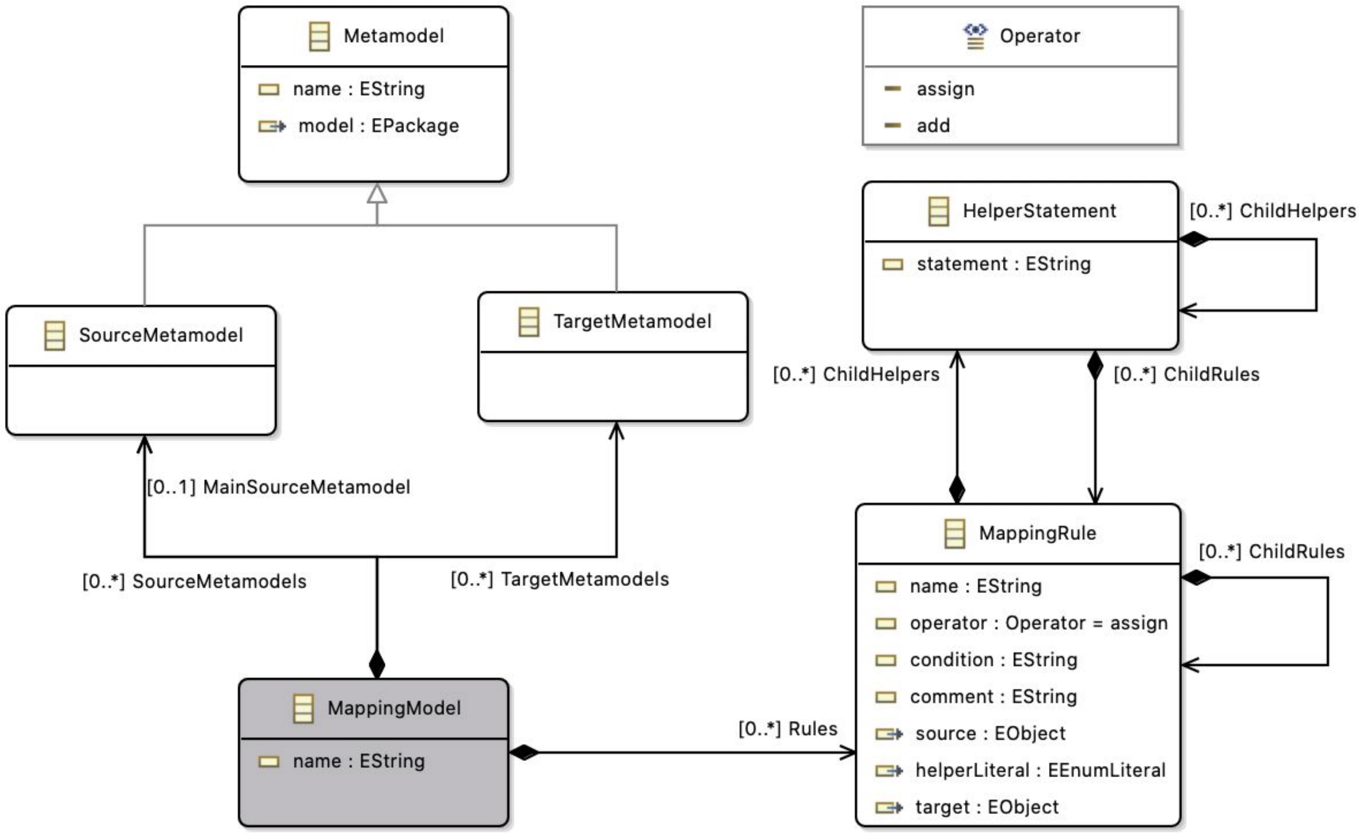
manual_input

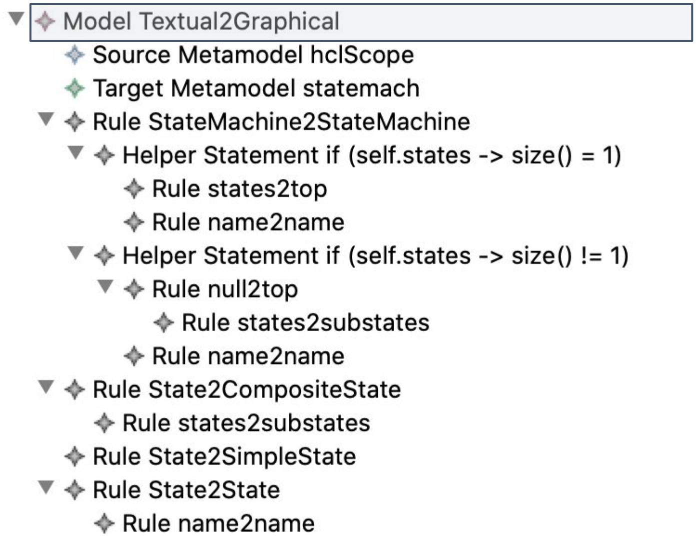
manual_input ⊂ mapping_input

{involved DSMLs, source/target elements, conditions, operator, comment, helper literal}

manual_input ∪ automatic_input = mapping_input

MAPPING METAMODEL





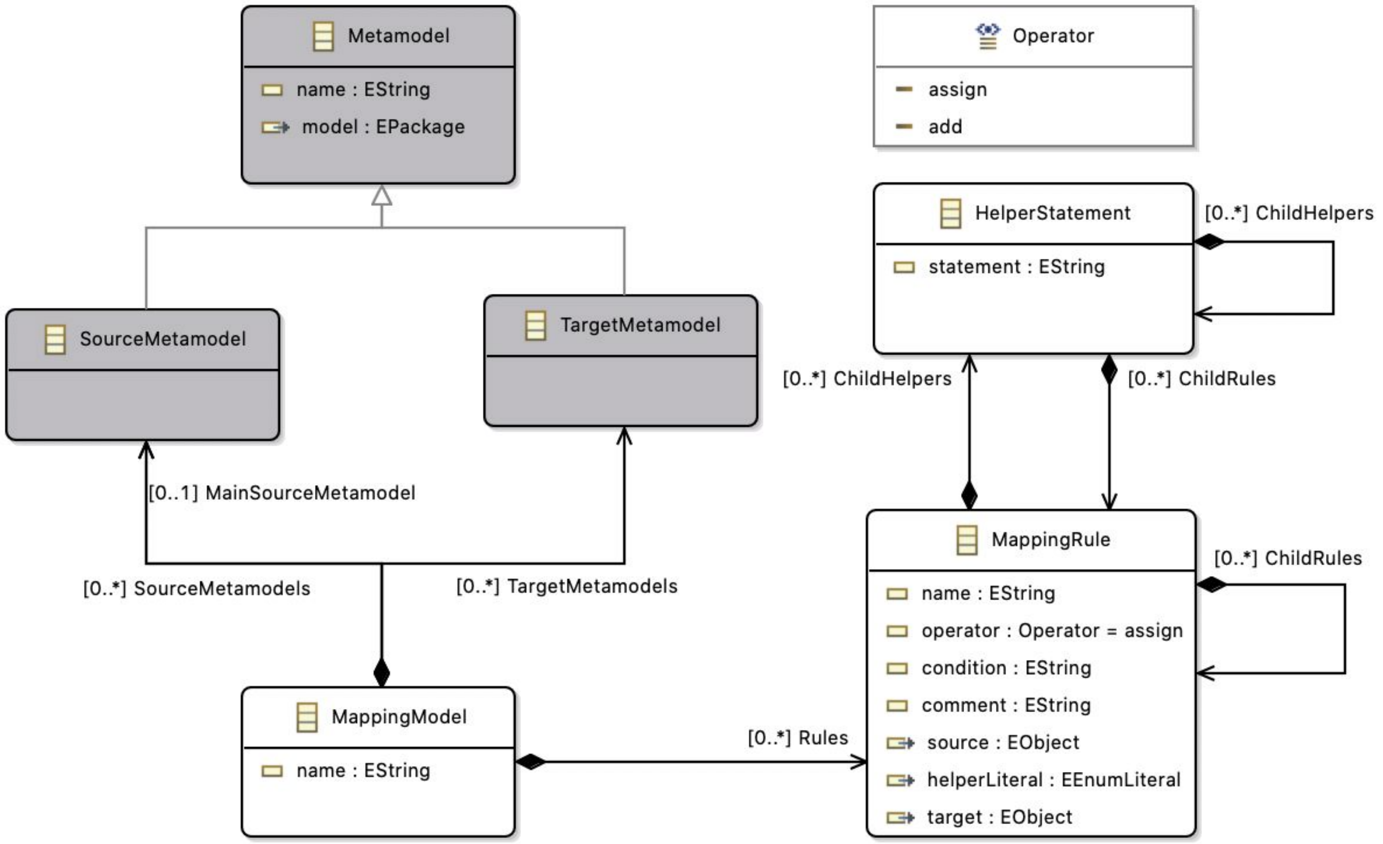
Property	Value
Main Source Metamodel	
Name	Textual2Graphical

HIGHER-ORDER TRANSFORMATIONS

- Model Textual2Graphical
  - Source Metamodel hclScope
  - Target Metamodel statemach
  - Rule StateMachine2StateMachine
    - Helper Statement if (self.states -> size() = 1)
      - Rule states2top
      - Rule name2name
    - Helper Statement if (self.states -> size() != 1)
      - Rule null2top
        - Rule states2substates
      - Rule name2name
  - Rule State2CompositeState
    - Rule states2substates
  - Rule State2SimpleState
  - Rule State2State
    - Rule name2name

```
modeltype hclScope uses 'http://www.xtext.org/example/hclscope/HclScope';
modeltype statemach uses 'http://www.eclipse.org/papyrusrt/xtumlrt/statemach';

transformation Textual2Graphical( in hclScopeModel:hclScope, out statemachModel:statemach);
main() {
hclScopeModel.rootObjects()[hclScope::StateMachine] -> map StateMachine2StateMachine();
}
mapping hclScope :: StateMachine :: StateMachine2StateMachine() : statemach :: StateMachine
{
    if (self.states -> size() = 1){
        result.top := self.states -> first().map State2CompositeState();
        result.name := self.name;
    };
    if (self.states -> size() != 1){
        var CompositeStateObject := object statemach :: CompositeState{
        substates := self.states -> map State2StateDisjunct();
    };
    top := CompositeStateObject;
        result.name := self.name;
    };
}
mapping hclScope :: State :: State2CompositeState() : statemach :: CompositeState
inherits hclScope :: State :: AbstractState2State
when {not(self.states -> isEmpty())}
{
    result.substates := self.states -> map State2StateDisjunct();
}
mapping hclScope :: State :: State2SimpleState() : statemach :: SimpleState
inherits hclScope :: State :: AbstractState2State
when {self.states -> isEmpty() }
{
}

mapping hclScope :: State :: State2StateDisjunct() : statemach :: State
disjuncts hclScope :: State :: State2CompositeState, hclScope :: State :: State2SimpleState{}

abstract mapping hclScope :: State :: AbstractState2State() : statemach :: State
{
    result.name := self.name;
}
```

MAPPING METAMODEL



- Model Textual2Graphical
  - Source Metamodel hclScope
  - Target Metamodel statemach
  - Rule StateMachine2StateMachine
    - Helper Statement if (self.states -> size() = 1)
      - Rule states2top
      - Rule name2name
    - Helper Statement if (self.states -> size() != 1)
      - Rule null2top
        - Rule states2substates
      - Rule name2name
  - Rule State2CompositeState
    - Rule states2substates
  - Rule State2SimpleState
  - Rule State2State
    - Rule name2name

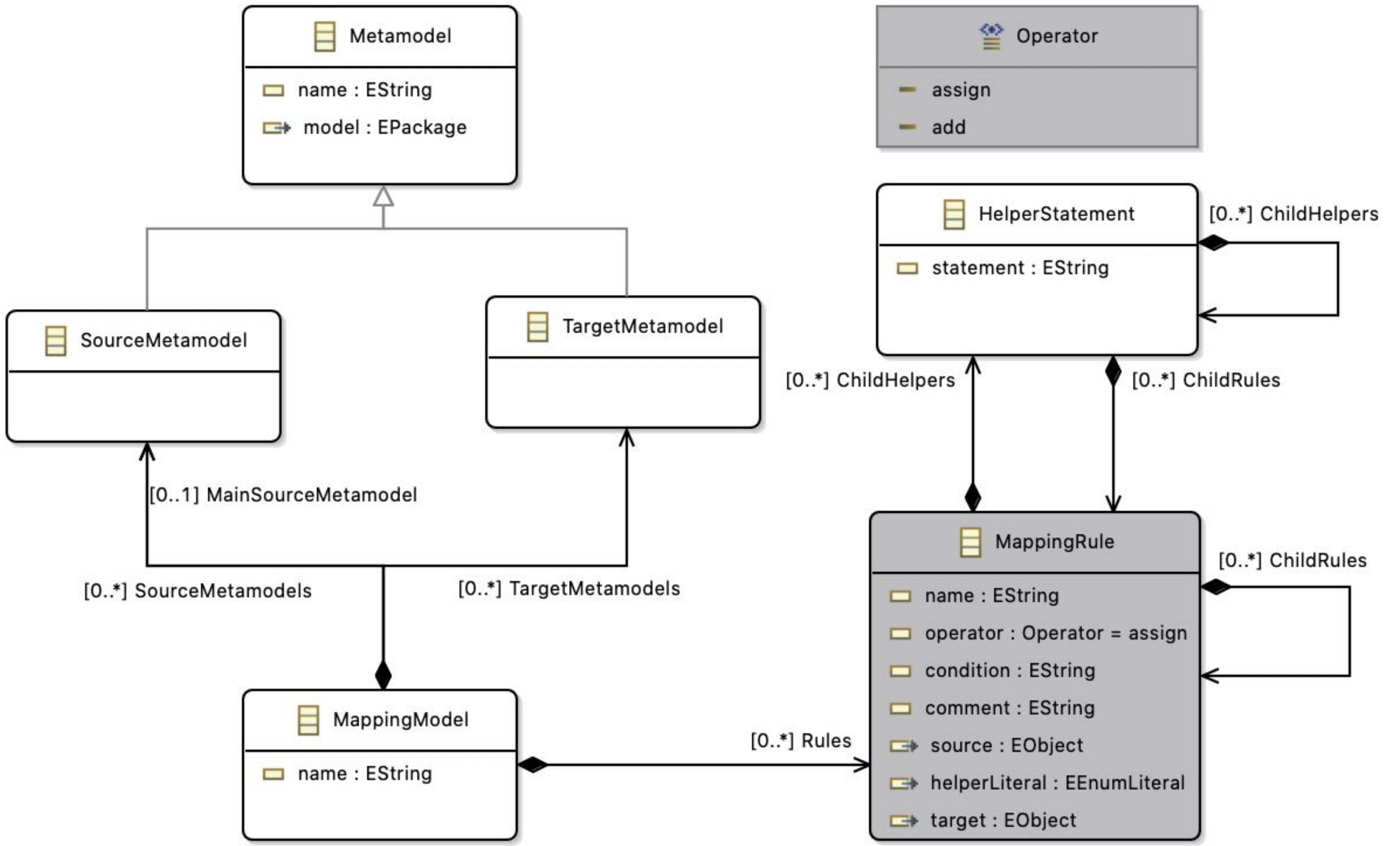
Property	Value
Model	hclScope
Name	hclScope

HIGHER-ORDER TRANSFORMATIONS

- Model Textual2Graphical
  - Source Metamodel hclScope
  - Target Metamodel statemach
  - Rule StateMachine2StateMachine
    - Helper Statement if (self.states -> size() = 1)
      - Rule states2top
      - Rule name2name
    - Helper Statement if (self.states -> size() != 1)
      - Rule null2top
        - Rule states2substates
      - Rule name2name
  - Rule State2CompositeState
    - Rule states2substates
  - Rule State2SimpleState
  - Rule State2State
    - Rule name2name

```
modeltype hclScope uses 'http://www.xtext.org/example/hclscope/HclScope';
modeltype statemach uses 'http://www.eclipse.org/papyrusrt/xtumlrt/statemach';

transformation Textual2Graphical( in hclScopeModel:hclScope, out statemachModel:statemach);
main() {
hclScopeModel.rootObjects()[hclScope::StateMachine] -> map StateMachine2StateMachine();
}
mapping hclScope :: StateMachine :: StateMachine2StateMachine() : statemach :: StateMachine
{
    if (self.states -> size() = 1){
        result.top := self.states -> first().map State2CompositeState();
        result.name := self.name;
    };
    if (self.states -> size() != 1){
        var CompositeStateObject := object statemach :: CompositeState{
        substates := self.states -> map State2StateDisjunct();
    };
    top := CompositeStateObject;
        result.name := self.name;
    };
}
mapping hclScope :: State :: State2CompositeState() : statemach :: CompositeState
inherits hclScope :: State :: AbstractState2State
when {not(self.states -> isEmpty())}
{
    result.substates := self.states -> map State2StateDisjunct();
}
mapping hclScope :: State :: State2SimpleState() : statemach :: SimpleState
inherits hclScope :: State :: AbstractState2State
when {self.states -> isEmpty() }
{
}

mapping hclScope :: State :: State2StateDisjunct() : statemach :: State
disjuncts hclScope :: State :: State2CompositeState, hclScope :: State :: State2SimpleState{}

abstract mapping hclScope :: State :: AbstractState2State() : statemach :: State
{
    result.name := self.name;
}
```

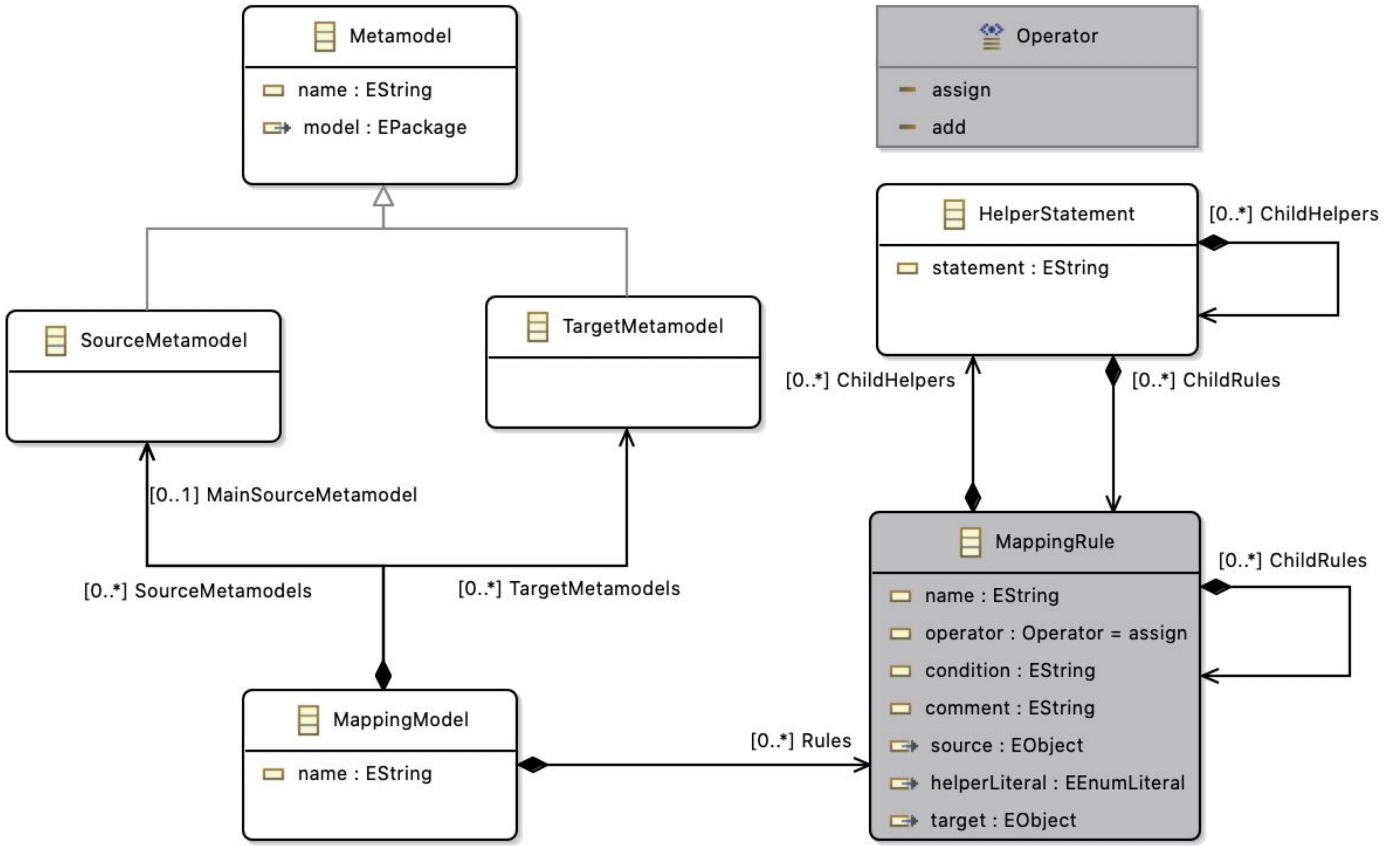
MAPPING METAMODEL



- Model Textual2Graphical
  - Source Metamodel hclScope
  - Target Metamodel statemach
  - Rule StateMachine2StateMachine
    - Helper Statement if (self.states -> size() = 1)
      - Rule states2top
      - Rule name2name
    - Helper Statement if (self.states -> size() != 1)
      - Rule null2top
        - Rule states2substates
      - Rule name2name
  - Rule State2CompositeState
    - Rule states2substates
  - Rule State2SimpleState
  - Rule State2State
    - Rule name2name

Property	Value
Comment	
Condition	not(self.states -> isEmpty())
Helper Literal	
Name	State2CompositeState
Operator	assign
Source	State -> Vertex • hclScope/State -> Vertex
Target	CompositeState -> State • statemach/CompositeState -> State

MAPPING METAMODEL

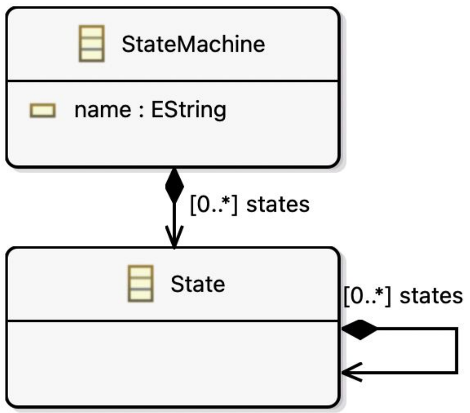


Scenario 1

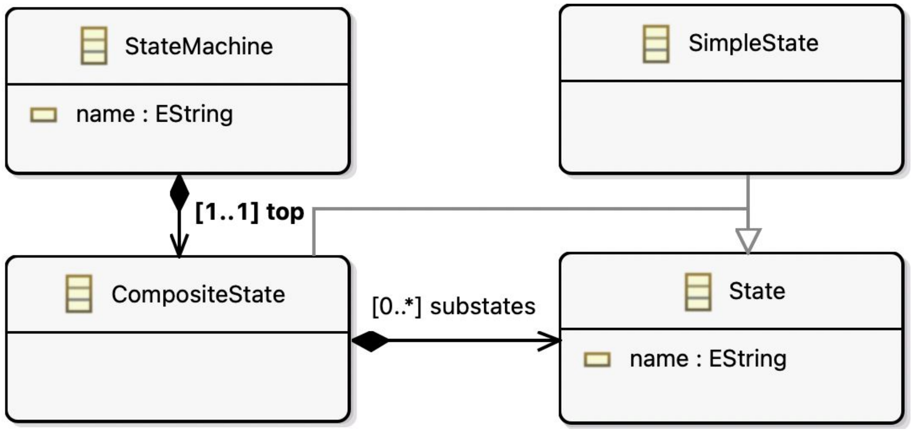
source!=null **and** target!=null

Non-empty set of input elements in the source model are transformed into a non-empty set of output elements in the target model.

SCENARIO 1

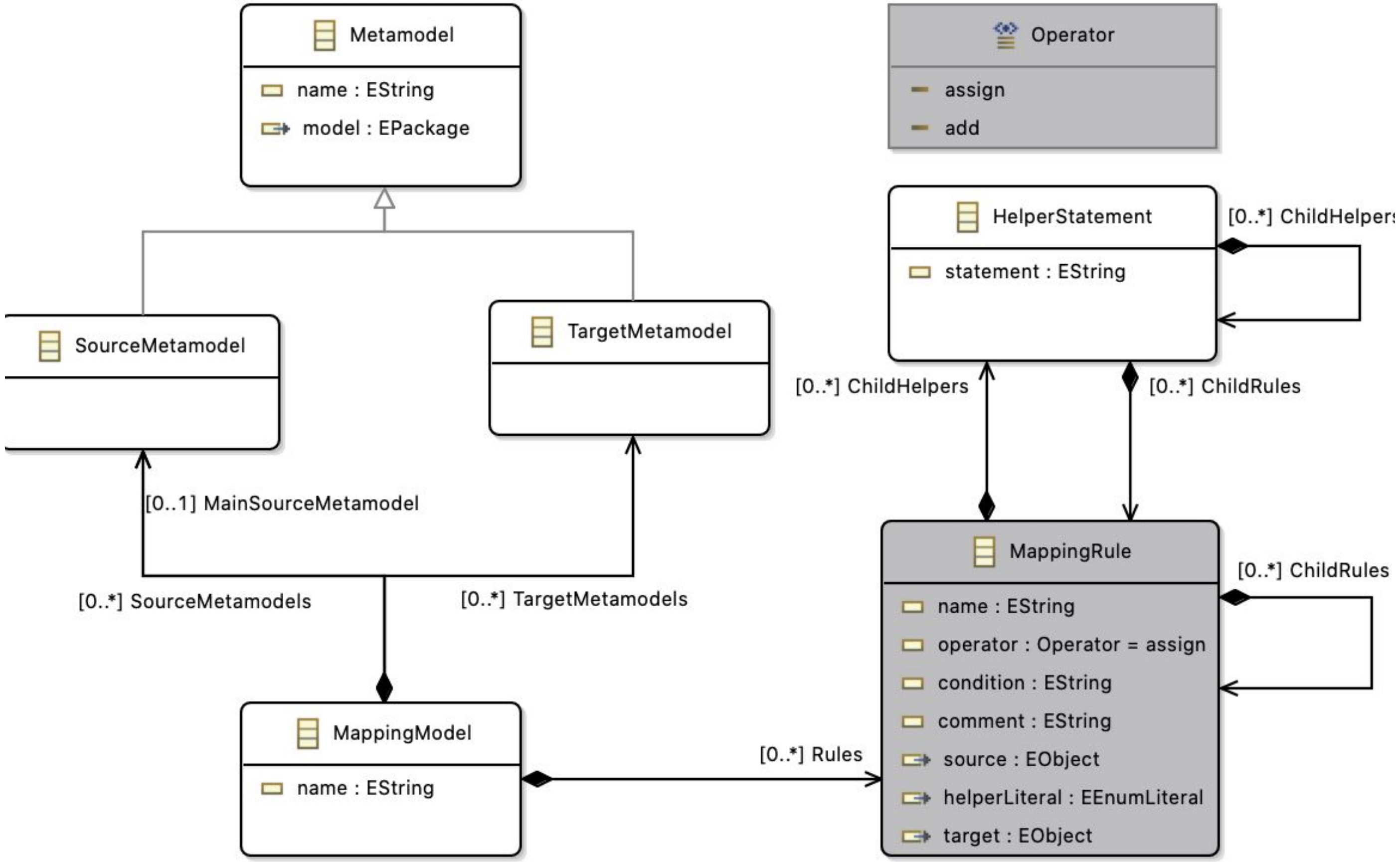


- Model Textual2Graphical
  - Source Metamodel hclScope
  - Target Metamodel statemach
  - Rule StateMachine2StateMachine
    - Helper Statement if (self.states -> size() = 1)
      - Rule states2top
      - Rule name2name
    - Helper Statement if (self.states -> size() != 1)
      - Rule null2top
        - Rule states2substates
      - Rule name2name
  - Rule State2CompositeState
    - Rule states2substates
  - Rule State2SimpleState
  - Rule State2State
    - Rule name2name



Property	Value
Comment	
Condition	
Helper Literal	
Name	StateMachine2StateMachine
Operator	assign
Source	StateMachine • hclScope/StateMachine
Target	StateMachine -> Behaviour • statemach/StateMachine

MAPPING METAMODEL



Scenario 1

source!=null **and** target!=null

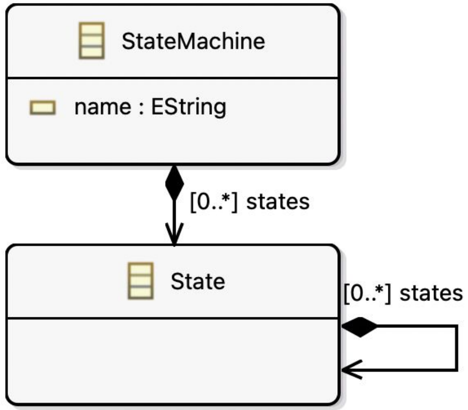
Non-empty set of input elements in the source model are transformed into a non-empty set of output elements in the target model.

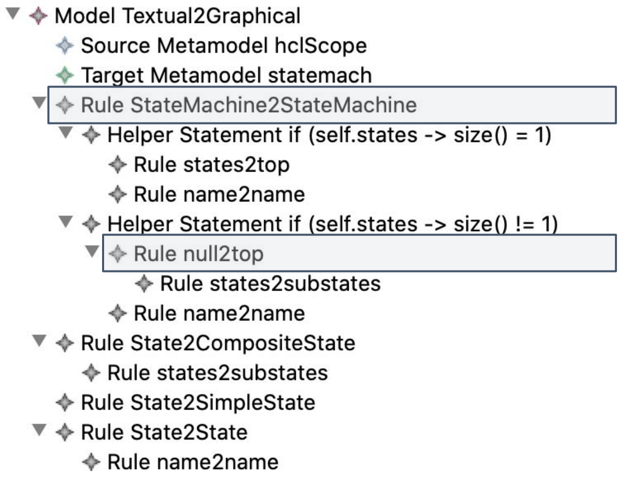
Scenario 2

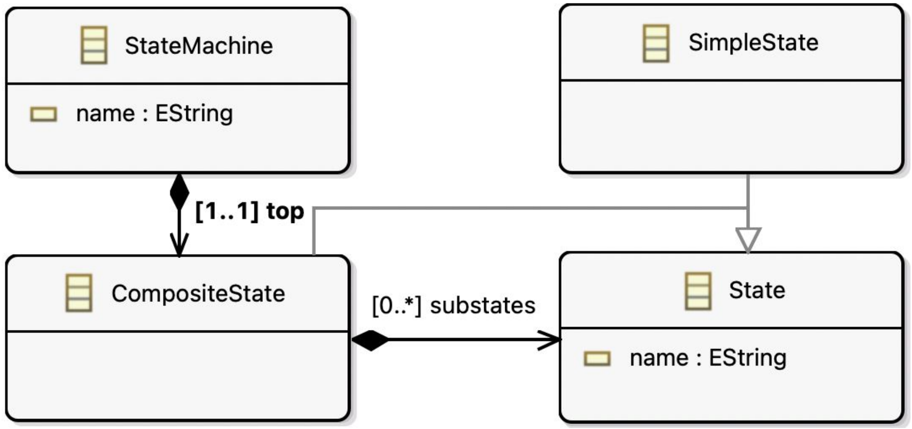
source==null **and** target!=null

Non-empty set of output elements are added to the target model.

SCENARIO 2

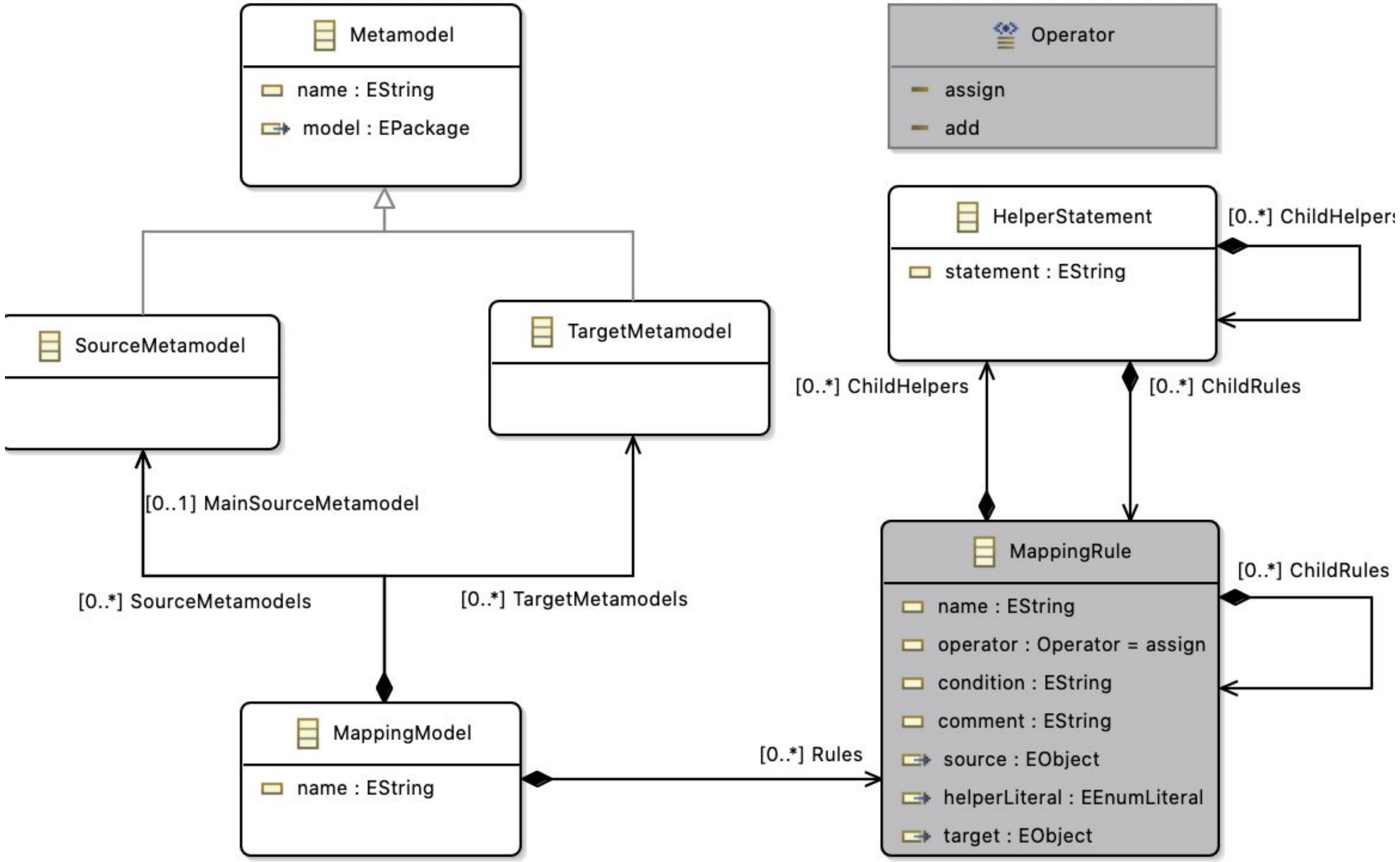






Property	Value
Comment	
Condition	
Helper Literal	
Name	null2top
Operator	assign
Source	
Target	top : CompositeState • statemach/StateMachine/top

MAPPING METAMODEL



Scenario 1

source!=null **and** target!=null

Non-empty set of input elements in the source model are transformed into a non-empty set of output elements in the target model.

Scenario 2

source==null **and** target!=null

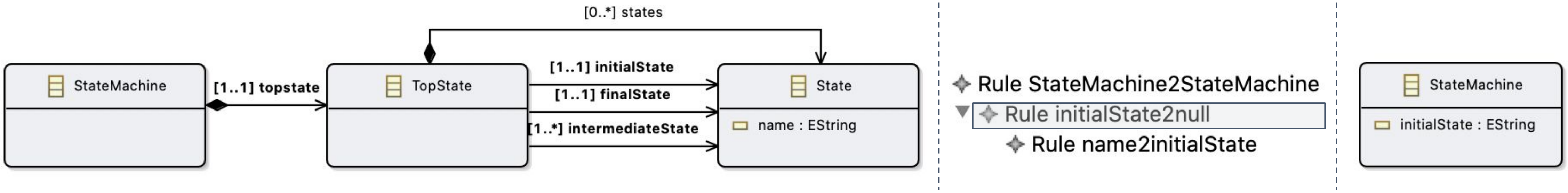
Non-empty set of output elements are added to the target model.

Scenario 3

source!=null **and** target==null

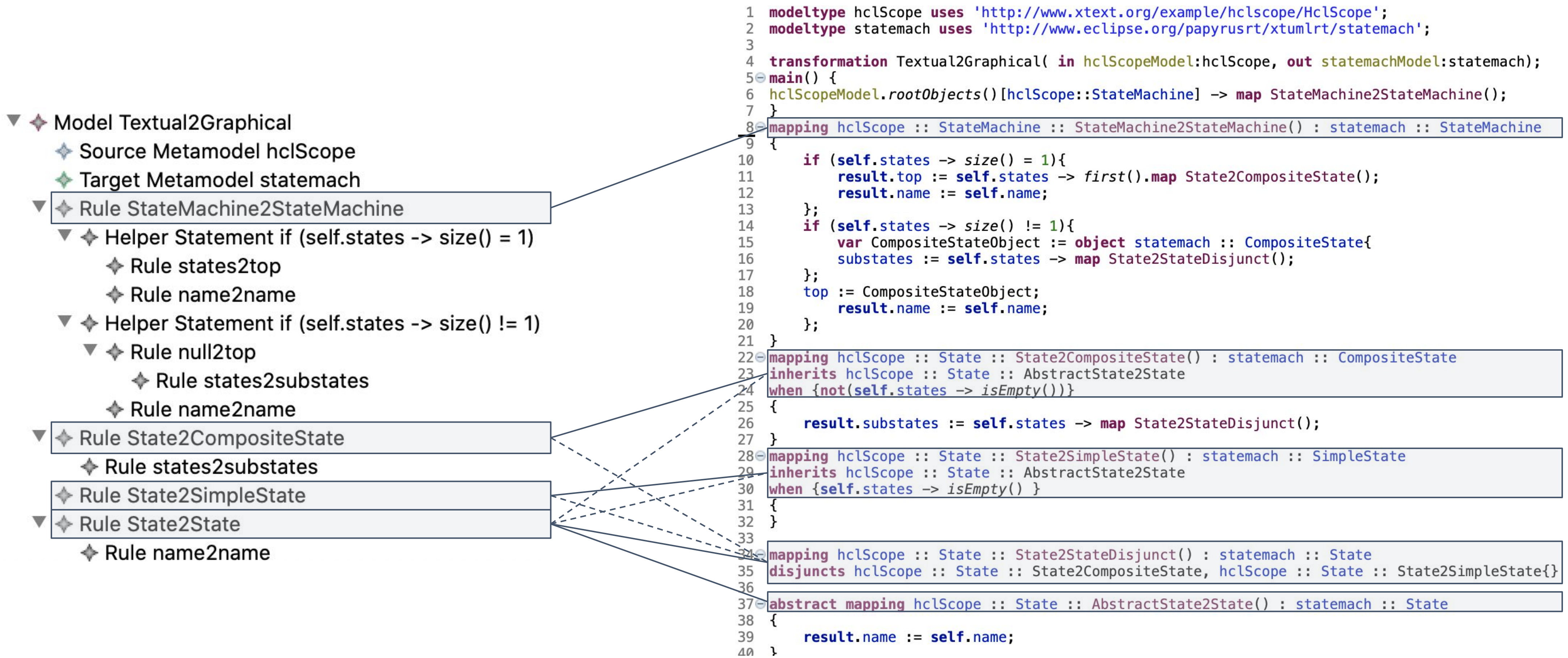
Non-empty set of input elements in the source model facilitates the navigation of model elements in the source model.

SCENARIO 3



Property	Value
Comment	
Condition	
Helper Literal	
Name	initialState2null
Operator	assign
Source	initialState : State • statemach/TopState/initialState
Target	

HIGHER-ORDER TRANSFORMATIONS



HIGHER-ORDER TRANSFORMATIONS

- Model Textual2Graphical
  - Source Metamodel hclScope
  - Target Metamodel statemach
  - Rule StateMachine2StateMachine
    - Helper Statement if (self.states -> size() = 1)
      - Rule states2top
      - Rule name2name
    - Helper Statement if (self.states -> size() != 1)
      - Rule null2top
        - Rule states2substates
      - Rule name2name
  - Rule State2CompositeState
    - Rule states2substates
  - Rule State2SimpleState
  - Rule State2State
    - Rule name2name

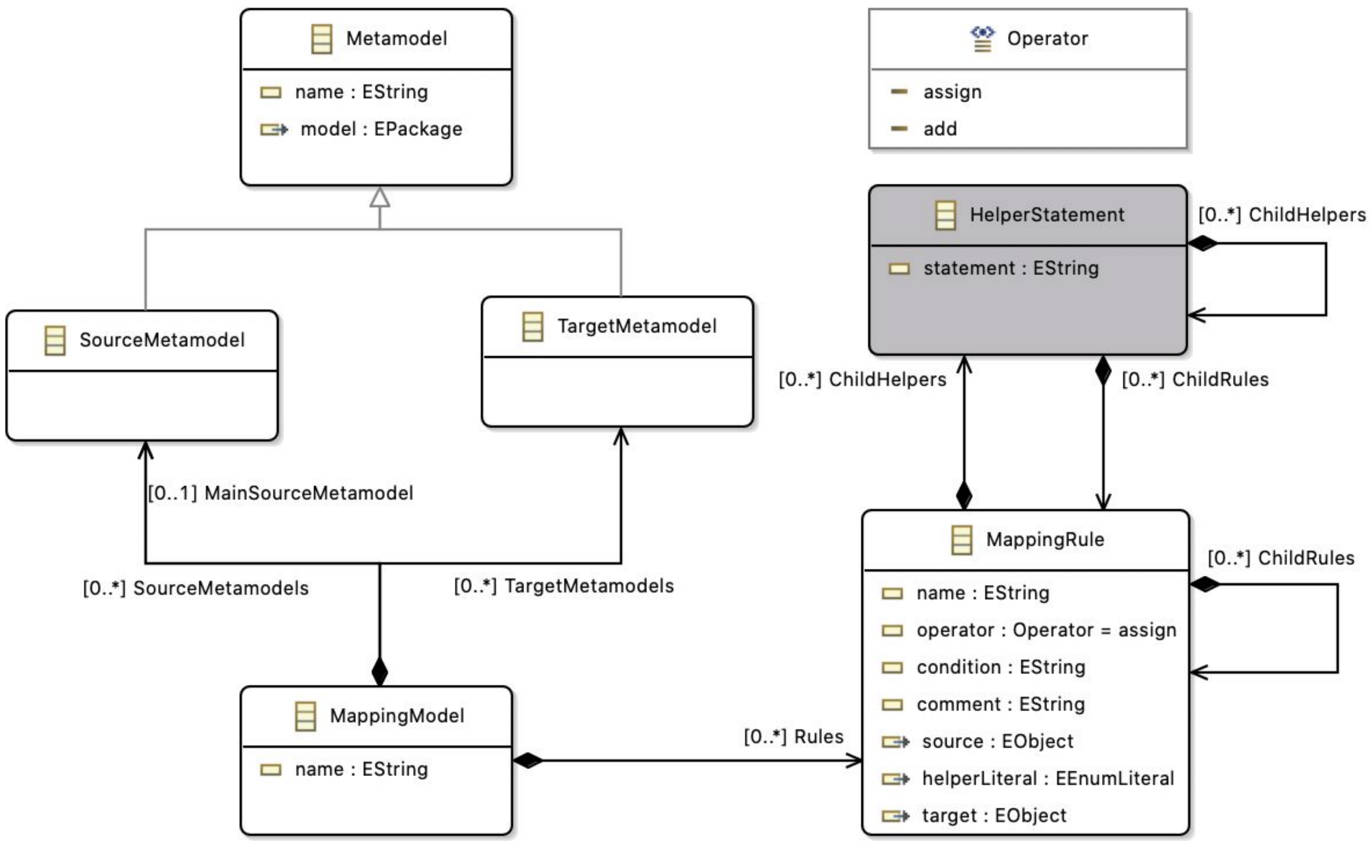
```
modeltype hclScope uses 'http://www.xtext.org/example/hclscope/HclScope';
modeltype statemach uses 'http://www.eclipse.org/papyrusrt/xtumlrt/statemach';

transformation Textual2Graphical( in hclScopeModel:hclScope, out statemachModel:statemach);
main() {
hclScopeModel.rootObjects()[hclScope::StateMachine] -> map StateMachine2StateMachine();
}
mapping hclScope :: StateMachine :: StateMachine2StateMachine() : statemach :: StateMachine
{
    if (self.states -> size() = 1){
        result.top := self.states -> first().map State2CompositeState();
        result.name := self.name;
    };
    if (self.states -> size() != 1){
        var CompositeStateObject := object statemach :: CompositeState{
        substates := self.states -> map State2StateDisjunct();
    };
    top := CompositeStateObject;
        result.name := self.name;
    };
}
mapping hclScope :: State :: State2CompositeState() : statemach :: CompositeState
inherits hclScope :: State :: AbstractState2State
when {not(self.states -> isEmpty())}
{
    result.substates := self.states -> map State2StateDisjunct();
}
mapping hclScope :: State :: State2SimpleState() : statemach :: SimpleState
inherits hclScope :: State :: AbstractState2State
when {self.states -> isEmpty() }
{
}

mapping hclScope :: State :: State2StateDisjunct() : statemach :: State
disjuncts hclScope :: State :: State2CompositeState, hclScope :: State :: State2SimpleState{}

abstract mapping hclScope :: State :: AbstractState2State() : statemach :: State
{
    result.name := self.name;
}
```

HIGHER-ORDER TRANSFORMATIONS

- Model Textual2Graphical
  - Source Metamodel hclScope
  - Target Metamodel statemach
  - Rule StateMachine2StateMachine
    - Helper Statement if (self.states -> size() = 1)
      - Rule states2top
      - Rule name2name
    - Helper Statement if (self.states -> size() != 1)
      - Rule null2top
        - Rule states2substates
      - Rule name2name
  - Rule State2CompositeState
    - Rule states2substates
  - Rule State2SimpleState
  - Rule State2State
    - Rule name2name

```
modeltype hclScope uses 'http://www.xtext.org/example/hclscope/HclScope';
modeltype statemach uses 'http://www.eclipse.org/papyrusrt/xtumlrt/statemach';

transformation Textual2Graphical( in hclScopeModel:hclScope, out statemachModel:statemach);
main() {
hclScopeModel.rootObjects()[hclScope::StateMachine] -> map StateMachine2StateMachine();
}
mapping hclScope :: StateMachine :: StateMachine2StateMachine() : statemach :: StateMachine
{
    if (self.states -> size() = 1){
        result.top := self.states -> first().map State2CompositeState();
        result.name := self.name;
    };
    if (self.states -> size() != 1){
        var CompositeStateObject := object statemach :: CompositeState{
        substates := self.states -> map State2StateDisjunct();
    };
    top := CompositeStateObject;
        result.name := self.name;
    };
}
mapping hclScope :: State :: State2CompositeState() : statemach :: CompositeState
inherits hclScope :: State :: AbstractState2State
when {not(self.states -> isEmpty())}
{
    result.substates := self.states -> map State2StateDisjunct();
}
mapping hclScope :: State :: State2SimpleState() : statemach :: SimpleState
inherits hclScope :: State :: AbstractState2State
when {self.states -> isEmpty() }
{
}

mapping hclScope :: State :: State2StateDisjunct() : statemach :: State
disjuncts hclScope :: State :: State2CompositeState, hclScope :: State :: State2SimpleState{}

abstract mapping hclScope :: State :: AbstractState2State() : statemach :: State
{
    result.name := self.name;
}
```

MAPPING METAMODEL



- Model Textual2Graphical
  - Source Metamodel hclScope
  - Target Metamodel statemach
  - Rule StateMachine2StateMachine
    - Helper Statement if (self.states -> size() = 1)
      - Rule states2top
      - Rule name2name
    - Helper Statement if (self.states -> size() != 1)
      - Rule null2top
        - Rule states2substates
      - Rule name2name
  - Rule State2CompositeState
    - Rule states2substates
  - Rule State2SimpleState
  - Rule State2State
    - Rule name2name

Property	Value
Statement	if (self.states -> size() = 1)

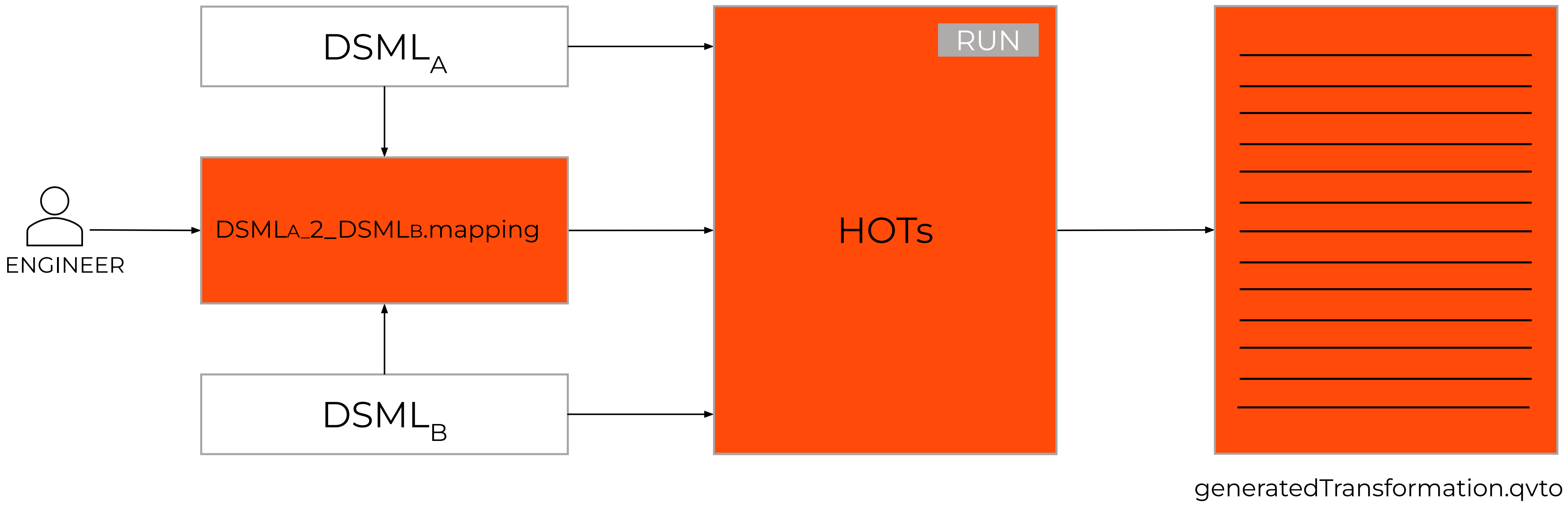
HIGHER-ORDER TRANSFORMATIONS

- Model Textual2Graphical
  - Source Metamodel hclScope
  - Target Metamodel statemach
  - Rule StateMachine2StateMachine
    - Helper Statement if (self.states -> size() = 1)
      - Rule states2top
      - Rule name2name
    - Helper Statement if (self.states -> size() != 1)
      - Rule null2top
        - Rule states2substates
      - Rule name2name
  - Rule State2CompositeState
    - Rule states2substates
  - Rule State2SimpleState
  - Rule State2State
    - Rule name2name

```
modeltype hclScope uses 'http://www.xtext.org/example/hclscope/HclScope';
modeltype statemach uses 'http://www.eclipse.org/papyrusrt/xtumlrt/statemach';

transformation Textual2Graphical( in hclScopeModel:hclScope, out statemachModel:statemach);
main() {
hclScopeModel.rootObjects()[hclScope::StateMachine] -> map StateMachine2StateMachine();
}
mapping hclScope :: StateMachine :: StateMachine2StateMachine() : statemach :: StateMachine
{
    if (self.states -> size() = 1){
        result.top := self.states -> first().map State2CompositeState();
        result.name := self.name;
    };
    if (self.states -> size() != 1){
        var CompositeStateObject := object statemach :: CompositeState{
        substates := self.states -> map State2StateDisjunct();
    };
    top := CompositeStateObject;
        result.name := self.name;
    };
}
mapping hclScope :: State :: State2CompositeState() : statemach :: CompositeState
inherits hclScope :: State :: AbstractState2State
when {not(self.states -> isEmpty())}
{
    result.substates := self.states -> map State2StateDisjunct();
}
mapping hclScope :: State :: State2SimpleState() : statemach :: SimpleState
inherits hclScope :: State :: AbstractState2State
when {self.states -> isEmpty() }
{
}

mapping hclScope :: State :: State2StateDisjunct() : statemach :: State
disjuncts hclScope :: State :: State2CompositeState, hclScope :: State :: State2SimpleState{}

abstract mapping hclScope :: State :: AbstractState2State() : statemach :: State
{
    result.name := self.name;
}
```

HIGHER-ORDER TRANSFORMATIONS

Implemented in Xtend, a flexible dialect of Java, which compiles into readable Java 8 compatible source code and is particularly suitable for the generation of pretty-printed textual artefacts.



HIGHER-ORDER TRANSFORMATIONS

1. FULLY QUALIFIED NAMES

```
modeltype hclScope uses 'http://www.xtext.org/example/hclscope/HclScope';
modeltype statemach uses 'http://www.eclipse.org/papyrusrt/xtumlrt/statemach';

transformation Textual2Graphical( in hclScopeModel:hclScope, out statemachModel:statemach);
main() {
hclScopeModel.rootObjects()[hclScope::StateMachine] -> map StateMachine2StateMachine();
}
mapping hclScope :: StateMachine :: StateMachine2StateMachine() : statemach :: StateMachine
{
    if (self.states -> size() = 1){
        result.top := self.states -> first().map State2CompositeState();
        result.name := self.name;
    };
    if (self.states -> size() != 1){
        var CompositeStateObject := object statemach :: CompositeState{
        substates := self.states -> map State2StateDisjunct();
    };
    top := CompositeStateObject;
        result.name := self.name;
    };
}
mapping hclScope :: State :: State2CompositeState() : statemach :: CompositeState
inherits hclScope :: State :: AbstractState2State
when {not(self.states -> isEmpty())}
{
    result.substates := self.states -> map State2StateDisjunct();
}
mapping hclScope :: State :: State2SimpleState() : statemach :: SimpleState
inherits hclScope :: State :: AbstractState2State
when {self.states -> isEmpty() }
{
}

mapping hclScope :: State :: State2StateDisjunct() : statemach :: State
disjuncts hclScope :: State :: State2CompositeState, hclScope :: State :: State2SimpleState{}

abstract mapping hclScope :: State :: AbstractState2State() : statemach :: State
{
    result.name := self.name;
}
```

HIGHER-ORDER TRANSFORMATIONS

1. FULLY QUALIFIED NAMES

2. NAMING CONVENTIONS FOR MAPPING RULES

```
modeltype hclScope uses 'http://www.xtext.org/example/hclscope/HclScope';
modeltype statemach uses 'http://www.eclipse.org/papyrusrt/xtumlrt/statemach';

transformation Textual2Graphical( in hclScopeModel:hclScope, out statemachModel:statemach);
main() {
hclScopeModel.rootObjects()[hclScope::StateMachine] -> map StateMachine2StateMachine();
}
mapping hclScope :: StateMachine :: StateMachine2StateMachine() : statemach :: StateMachine
{
    if (self.states -> size() = 1){
        result.top := self.states -> first().map State2CompositeState();
        result.name := self.name;
    };
    if (self.states -> size() != 1){
        var CompositeStateObject := object statemach :: CompositeState{
        substates := self.states -> map State2StateDisjunct();
    };
    top := CompositeStateObject;
        result.name := self.name;
    };
}
mapping hclScope :: State :: State2CompositeState() : statemach :: CompositeState
inherits hclScope :: State :: AbstractState2State
when {not(self.states -> isEmpty())}
{
    result.substates := self.states -> map State2StateDisjunct();
}
mapping hclScope :: State :: State2SimpleState() : statemach :: SimpleState
inherits hclScope :: State :: AbstractState2State
when {self.states -> isEmpty() }
{
}

mapping hclScope :: State :: State2StateDisjunct(): statemach :: State
disjuncts hclScope :: State :: State2CompositeState, hclScope :: State :: State2SimpleState{}

abstract mapping hclScope :: State :: AbstractState2State() : statemach :: State
{
    result.name := self.name;
}
```

HIGHER-ORDER TRANSFORMATIONS

1. FULLY QUALIFIED NAMES
2. NAMING CONVENTIONS FOR MAPPING RULES
3. **AUTOMATIC RULE INVOCATION**

```
modeltype hclScope uses 'http://www.xtext.org/example/hclscope/HclScope';
modeltype statemach uses 'http://www.eclipse.org/papyrusrt/xtumlrt/statemach';

transformation Textual2Graphical( in hclScopeModel:hclScope, out statemachModel:statemach);
main() {
hclScopeModel.rootObjects()[hclScope::StateMachine] -> map StateMachine2StateMachine();
}
mapping hclScope :: StateMachine :: StateMachine2StateMachine() : statemach :: StateMachine
{
    if (self.states -> size() = 1){
        result.top := self.states -> first().map State2CompositeState();
        result.name := self.name;
    };
    if (self.states -> size() != 1){
        var CompositeStateObject := object statemach :: CompositeState{
        substates := self.states -> map State2StateDisjunct();
    };
    top := CompositeStateObject;
        result.name := self.name;
    };
}
mapping hclScope :: State :: State2CompositeState() : statemach :: CompositeState
inherits hclScope :: State :: AbstractState2State
when {not(self.states -> isEmpty())}
{
    result.substates := self.states -> map State2StateDisjunct();
}
mapping hclScope :: State :: State2SimpleState() : statemach :: SimpleState
inherits hclScope :: State :: AbstractState2State
when {self.states -> isEmpty() }
{
}

mapping hclScope :: State :: State2StateDisjunct() : statemach :: State
disjuncts hclScope :: State :: State2CompositeState, hclScope :: State :: State2SimpleState{}

abstract mapping hclScope :: State :: AbstractState2State() : statemach :: State
{
    result.name := self.name;
}
```

HIGHER-ORDER TRANSFORMATIONS

1. FULLY QUALIFIED NAMES
2. NAMING CONVENTIONS FOR MAPPING RULES
3. AUTOMATIC RULE INVOCATION
4. **ABSTRACT MAPPING RULES**

```
modeltype hclScope uses 'http://www.xtext.org/example/hclscope/HclScope';
modeltype statemach uses 'http://www.eclipse.org/papyrusrt/xtumlrt/statemach';

transformation Textual2Graphical( in hclScopeModel:hclScope, out statemachModel:statemach);
main() {
hclScopeModel.rootObjects()[hclScope::StateMachine] -> map StateMachine2StateMachine();
}
mapping hclScope :: StateMachine :: StateMachine2StateMachine() : statemach :: StateMachine
{
    if (self.states -> size() = 1){
        result.top := self.states -> first().map State2CompositeState();
        result.name := self.name;
    };
    if (self.states -> size() != 1){
        var CompositeStateObject := object statemach :: CompositeState{
        substates := self.states -> map State2StateDisjunct();
    };
    top := CompositeStateObject;
        result.name := self.name;
    };
}
mapping hclScope :: State :: State2CompositeState() : statemach :: CompositeState
inherits hclScope :: State :: AbstractState2State
when {not(self.states -> isEmpty())}
{
    result.substates := self.states -> map State2StateDisjunct();
}
mapping hclScope :: State :: State2SimpleState() : statemach :: SimpleState
inherits hclScope :: State :: AbstractState2State
when {self.states -> isEmpty() }
{
}

mapping hclScope :: State :: State2StateDisjunct() : statemach :: State
disjuncts hclScope :: State :: State2CompositeState, hclScope :: State :: State2SimpleState{}

abstract mapping hclScope :: State :: AbstractState2State() : statemach :: State
{
    result.name := self.name;
}
```

HIGHER-ORDER TRANSFORMATIONS

1. FULLY QUALIFIED NAMES
2. NAMING CONVENTIONS FOR MAPPING RULES
3. AUTOMATIC RULE INVOCATION
4. ABSTRACT MAPPING RULES
5. **DISJUNCTION**

```
modeltype hclScope uses 'http://www.xtext.org/example/hclscope/HclScope';
modeltype statemach uses 'http://www.eclipse.org/papyrusrt/xtumlrt/statemach';

transformation Textual2Graphical( in hclScopeModel:hclScope, out statemachModel:statemach);
main() {
hclScopeModel.rootObjects()[hclScope::StateMachine] -> map StateMachine2StateMachine();
}
mapping hclScope :: StateMachine :: StateMachine2StateMachine() : statemach :: StateMachine
{
    if (self.states -> size() = 1){
        result.top := self.states -> first().map State2CompositeState();
        result.name := self.name;
    };
    if (self.states -> size() != 1){
        var CompositeStateObject := object statemach :: CompositeState{
        substates := self.states -> map State2StateDisjunct();
    };
    top := CompositeStateObject;
        result.name := self.name;
    };
}
mapping hclScope :: State :: State2CompositeState() : statemach :: CompositeState
inherits hclScope :: State :: AbstractState2State
when {not(self.states -> isEmpty())}
{
    result.substates := self.states -> map State2StateDisjunct();
}
mapping hclScope :: State :: State2SimpleState() : statemach :: SimpleState
inherits hclScope :: State :: AbstractState2State
when {self.states -> isEmpty() }
{
}

mapping hclScope :: State :: State2StateDisjunct() : statemach :: State
disjuncts hclScope :: State :: State2CompositeState, hclScope :: State :: State2SimpleState{}

abstract mapping hclScope :: State :: AbstractState2State() : statemach :: State
{
    result.name := self.name;
}
```

HIGHER-ORDER TRANSFORMATIONS

1. FULLY QUALIFIED NAMES
2. NAMING CONVENTIONS FOR MAPPING RULES
3. AUTOMATIC RULE INVOCATION
4. ABSTRACT MAPPING RULES
5. DISJUNCTION
6. **INHERITANCE**

```
modeltype hclScope uses 'http://www.xtext.org/example/hclscope/HclScope';
modeltype statemach uses 'http://www.eclipse.org/papyrusrt/xtumlrt/statemach';

transformation Textual2Graphical( in hclScopeModel:hclScope, out statemachModel:statemach);
main() {
hclScopeModel.rootObjects()[hclScope::StateMachine] -> map StateMachine2StateMachine();
}
mapping hclScope :: StateMachine :: StateMachine2StateMachine() : statemach :: StateMachine
{
    if (self.states -> size() = 1){
        result.top := self.states -> first().map State2CompositeState();
        result.name := self.name;
    };
    if (self.states -> size() != 1){
        var CompositeStateObject := object statemach :: CompositeState{
        substates := self.states -> map State2StateDisjunct();
    };
    top := CompositeStateObject;
        result.name := self.name;
    };
}
mapping hclScope :: State :: State2CompositeState() : statemach :: CompositeState
inherits hclScope :: State :: AbstractState2State
when {not(self.states -> isEmpty())}
{
    result.substates := self.states -> map State2StateDisjunct();
}
mapping hclScope :: State :: State2SimpleState() : statemach :: SimpleState
inherits hclScope :: State :: AbstractState2State
when {self.states -> isEmpty() }
{
}

mapping hclScope :: State :: State2StateDisjunct() : statemach :: State
disjuncts hclScope :: State :: State2CompositeState, hclScope :: State :: State2SimpleState{}

abstract mapping hclScope :: State :: AbstractState2State() : statemach :: State
{
    result.name := self.name;
}
```

HIGHER-ORDER TRANSFORMATIONS

1. FULLY QUALIFIED NAMES
2. NAMING CONVENTIONS FOR MAPPING RULES
3. AUTOMATIC RULE INVOCATION
4. ABSTRACT MAPPING RULES
5. DISJUNCTION
6. INHERITANCE
7. **GUARDS/OCL FILTERS**

```
modeltype hclScope uses 'http://www.xtext.org/example/hclscope/HclScope';
modeltype statemach uses 'http://www.eclipse.org/papyrusrt/xtumlrt/statemach';

transformation Textual2Graphical( in hclScopeModel:hclScope, out statemachModel:statemach);
main() {
hclScopeModel.rootObjects()[hclScope::StateMachine] -> map StateMachine2StateMachine();
}
mapping hclScope :: StateMachine :: StateMachine2StateMachine() : statemach :: StateMachine
{
    if (self.states -> size() = 1){
        result.top := self.states -> first().map State2CompositeState();
        result.name := self.name;
    };
    if (self.states -> size() != 1){
        var CompositeStateObject := object statemach :: CompositeState{
        substates := self.states -> map State2StateDisjunct();
    };
    top := CompositeStateObject;
        result.name := self.name;
    };
}
mapping hclScope :: State :: State2CompositeState() : statemach :: CompositeState
inherits hclScope :: State :: AbstractState2State
when {not(self.states -> isEmpty())}
{
    result.substates := self.states -> map State2StateDisjunct();
}
mapping hclScope :: State :: State2SimpleState() : statemach :: SimpleState
inherits hclScope :: State :: AbstractState2State
when {self.states -> isEmpty() }
{
}

mapping hclScope :: State :: State2StateDisjunct() : statemach :: State
disjuncts hclScope :: State :: State2CompositeState, hclScope :: State :: State2SimpleState{}

abstract mapping hclScope :: State :: AbstractState2State() : statemach :: State
{
    result.name := self.name;
}
```

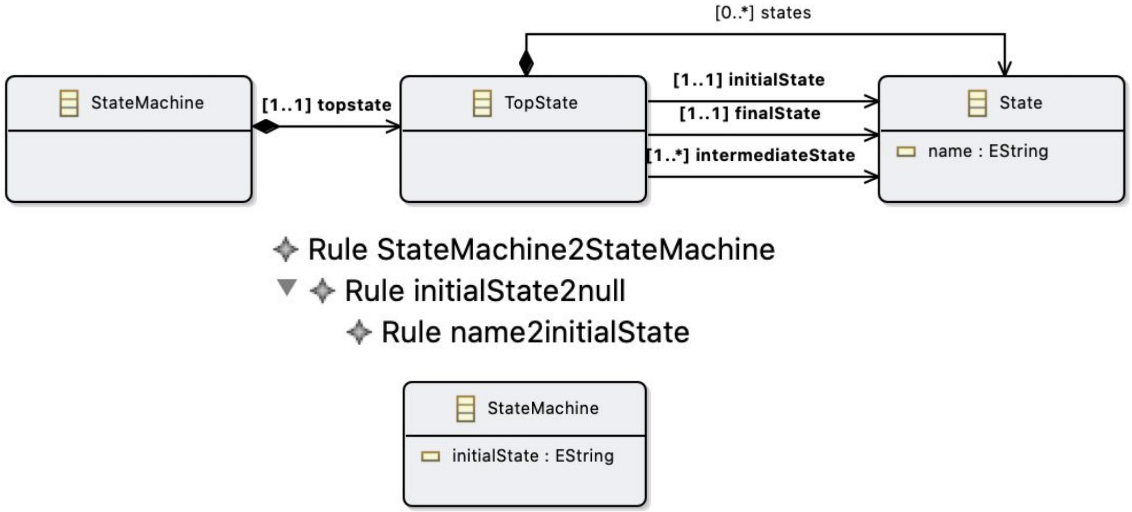
HIGHER-ORDER TRANSFORMATIONS

1. FULLY QUALIFIED NAMES
2. NAMING CONVENTIONS FOR MAPPING RULES
3. AUTOMATIC RULE INVOCATION
4. ABSTRACT MAPPING RULES
5. DISJUNCTION
6. INHERITANCE
7. GUARDS/OCL FILTERS
8. **ASSIGNMENT AND NAVIGATION OPERATORS**

```
modeltype hclScope uses 'http://www.xtext.org/example/hclscope/HclScope';
modeltype statemach uses 'http://www.eclipse.org/papyrusrt/xtumlrt/statemach';

transformation Textual2Graphical( in hclScopeModel:hclScope, out statemachModel:statemach);
main() {
hclScopeModel.rootObjects()[hclScope::StateMachine] -> map StateMachine2StateMachine();
}
mapping hclScope :: StateMachine :: StateMachine2StateMachine() : statemach :: StateMachine
{
    if (self.states -> size() = 1){
        result.top := self.states -> first().map State2CompositeState();
        result.name := self.name;
    };
    if (self.states -> size() != 1){
        var CompositeStateObject := object statemach :: CompositeState{
        substates := self.states -> map State2StateDisjunct();
    };
    top := CompositeStateObject;
        result.name := self.name;
    };
}
mapping hclScope :: State :: State2CompositeState() : statemach :: CompositeState
inherits hclScope :: State :: AbstractState2State
when {not(self.states -> isEmpty())}
{
    result.substates := self.states -> map State2StateDisjunct();
}
mapping hclScope :: State :: State2SimpleState() : statemach :: SimpleState
inherits hclScope :: State :: AbstractState2State
when {self.states -> isEmpty() }
{
}

mapping hclScope :: State :: State2StateDisjunct() : statemach :: State
disjuncts hclScope :: State :: State2CompositeState, hclScope :: State :: State2SimpleState{}

abstract mapping hclScope :: State :: AbstractState2State() : statemach :: State
{
    result.name := self.name;
}
```

HIGHER-ORDER TRANSFORMATIONS

1. FULLY QUALIFIED NAMES
2. NAMING CONVENTIONS FOR MAPPING RULES
3. AUTOMATIC RULE INVOCATION
4. ABSTRACT MAPPING RULES
5. DISJUNCTION
6. INHERITANCE
7. GUARDS/OCL FILTERS
8. ASSIGNMENT AND NAVIGATION OPERATORS
9. **NAVIGATION PATH**




```
mapping DSML1 :: StateMachine :: StateMachine2StateMachine() : DSML2 :: StateMachine {
initialState := self.topstate.initialState.name;
}
```

ADVANTAGES

1. Generic solution for the provision of blended modeling environments from arbitrary Ecore-based DSMLs.

2. Seamless synchronization for both classical blended modeling (between notations of the same language) and extended blended modeling (between two disjoint languages with the same or different notations).

3. Facilitates the co-evolution of the synchronization transformations in response to DSML evolution.

4. Supports the enrichment of modeling tools through the inclusion of additional modeling languages by facilitating the establishment of the synchronization infrastructure between them, pairwise.

5. Enables the inclusion of domain experts with no model transformation language knowledge, and a more accurate and less cumbersome process for developers.

6. Facilitates the transition from a homogeneous modeling approach (appropriate for simple systems/software) to a blended modeling methodology (appropriate for complex, industrial-grade, software-intensive systems).

DEMO DETAILS

UML REAL TIME PROFILE FOR STATE MACHINES

DSMLs

- $DSML_A$: UML-RT profile for state machines used in Papyrus-RT (graphical).
- $DSML_B$: Textual notation used for UML-RT for state machines used in HCL RTist (textual).

STEPS

1. Generation of the graphical and textual notations.
2. Definition of mapping rules between graphical and textual notations.
3. Generation of the synchronisation transformations via higher-order transformations.
4. modeling of software architectures via graphical and textual notations.
5. Synchronisation of model changes across notations.

DEMO DETAILS

- statemach
  - StateMachine -> Behaviour
  - Vertex -> NamedElement
  - Transition -> RedefinableElement
  - State -> Vertex, RedefinableElement
  - Pseudostate -> Vertex
  - SimpleState -> State
  - CompositeState -> State
  - Trigger -> NamedElement
  - Guard -> NamedElement
  - ActionChain -> NamedElement
  - EntryPoint -> Pseudostate
  - ExitPoint -> Pseudostate
  - InitialPoint -> Pseudostate
  - DeepHistory -> Pseudostate
  - ChoicePoint -> Pseudostate
  - JunctionPoint -> Pseudostate
  - TerminatePoint -> Pseudostate

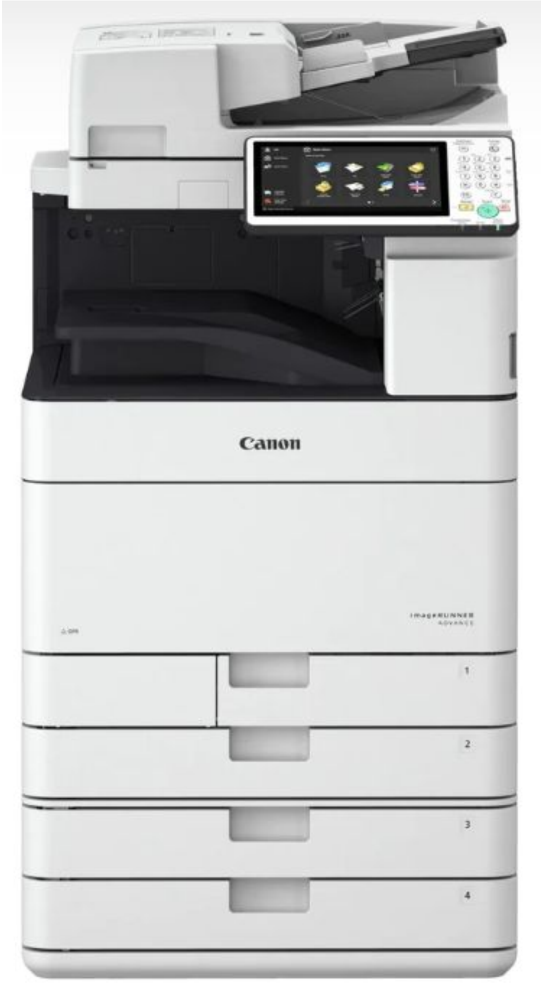
- hclScope
  - StateMachine
  - State -> Vertex
  - EntryAction
  - ExitAction
  - InitialState
  - Junction -> Vertex
  - Choice -> Vertex
  - EntryPoint -> Vertex
  - ExitPoint -> Vertex
  - DeepHistory
  - InitialTransition -> Transitions
  - Transition -> Transitions
  - InternalTransition
  - HistoryTransition -> Transitions
  - TransitionBody
  - TransitionGuard
  - TransitionOperation
  - Trigger
  - Method
  - Parameter
  - MethodParameterTrigger
  - Port
  - Event
  - PortEventTrigger
  - Vertex
  - Transitions

PART 2

CROSS-PLATFORM REAL-TIME COLLABORATION

KOUSAR ASLAM, YUNA CHEN, MUHAMMAD BUTT, IVANO MALAVOLTA

PROBLEM



Model engineering is a complex task

↳ **Collaborative MDSE**

COLLABORATIVE MDSE

- increase communication and collaboration
- among technical and non-technical stakeholders
- for building complex software systems

Collaboration can happen **offline** or in **real-time**

REAL-TIME COLLABORATION

Collaborative editing of data by multiple modelers over a network, in real-time!

A well known example: *Google docs*

WHY CROSS-PLATFORM?

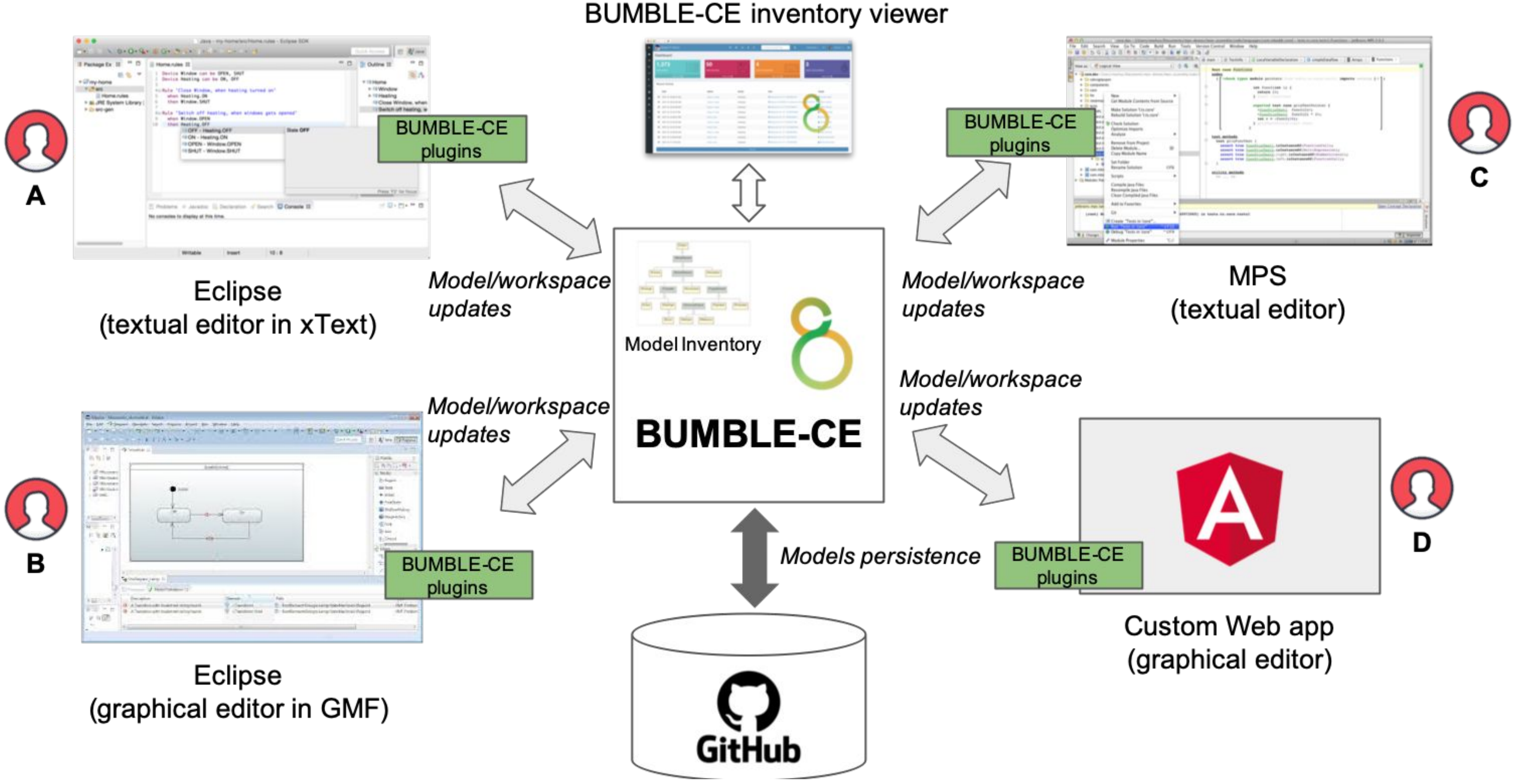
Current limitation: *single-platform support*

What if model engineers use different modeling platforms?

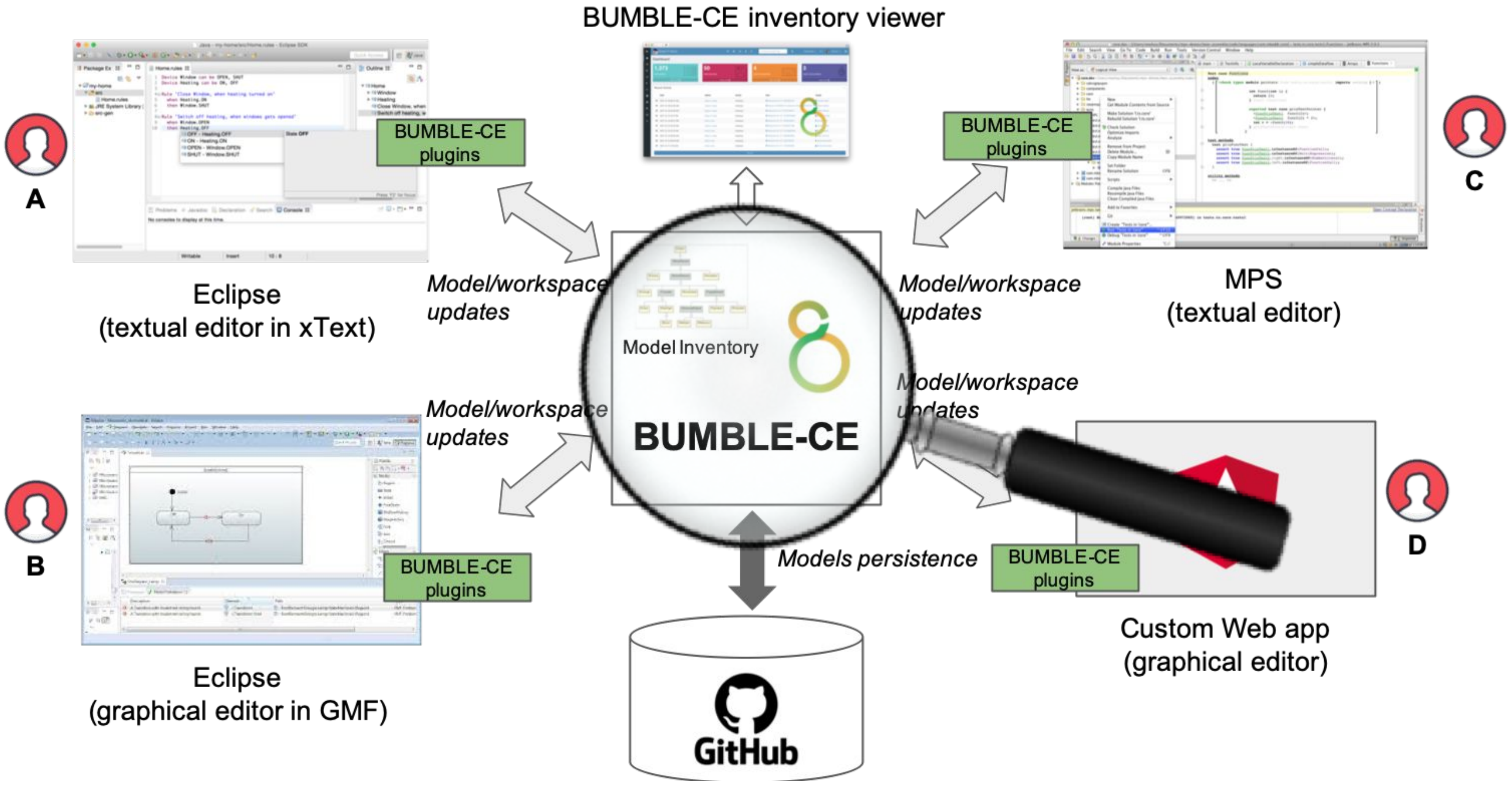
PROPOSED SOLUTION

- Collaboration engine (BUMBLE-CE)
    - real-time collaboration
    - independent of modeling platforms
    - independent of DSMLs

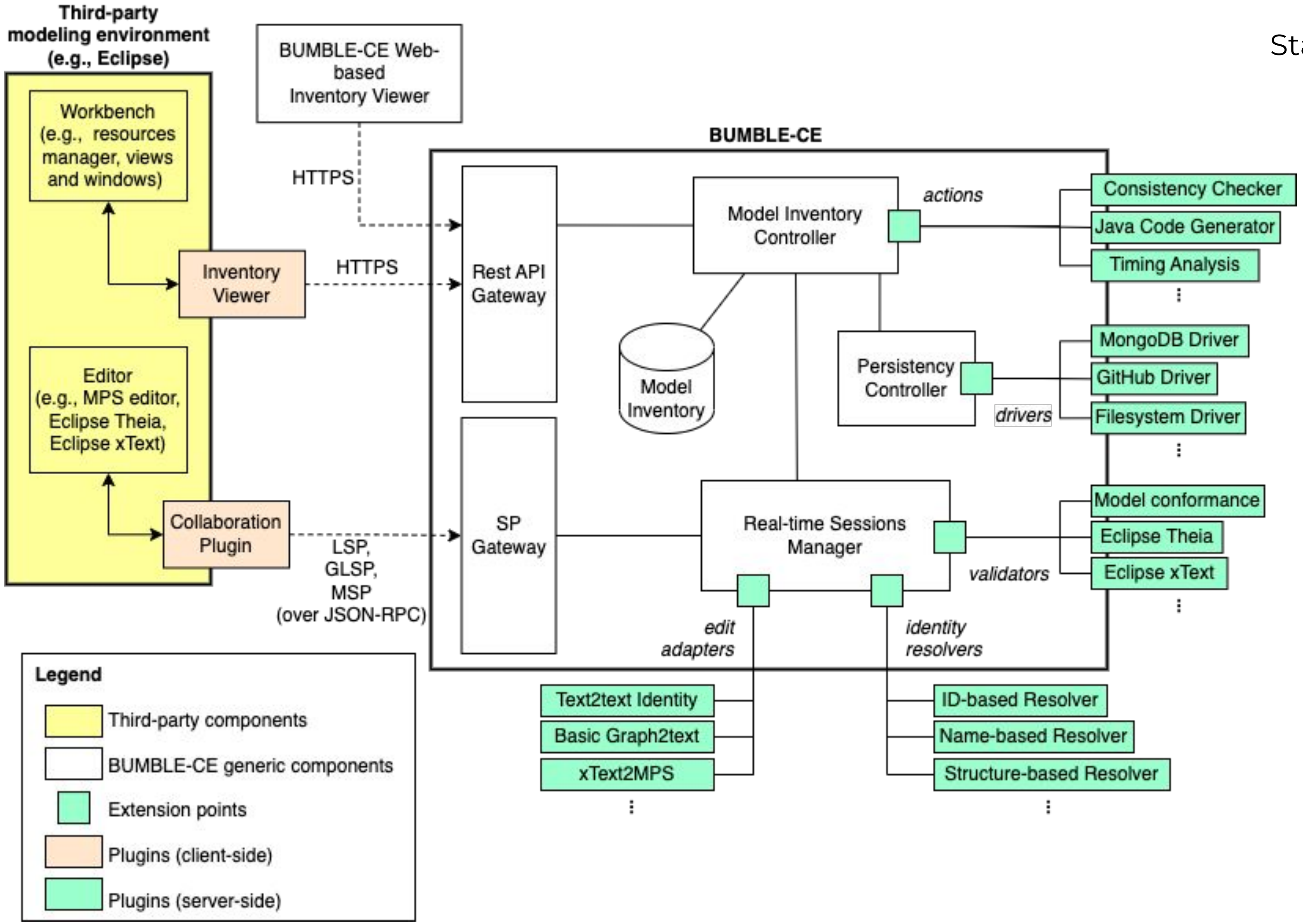
EXAMPLE SCENARIO



EXAMPLE SCENARIO



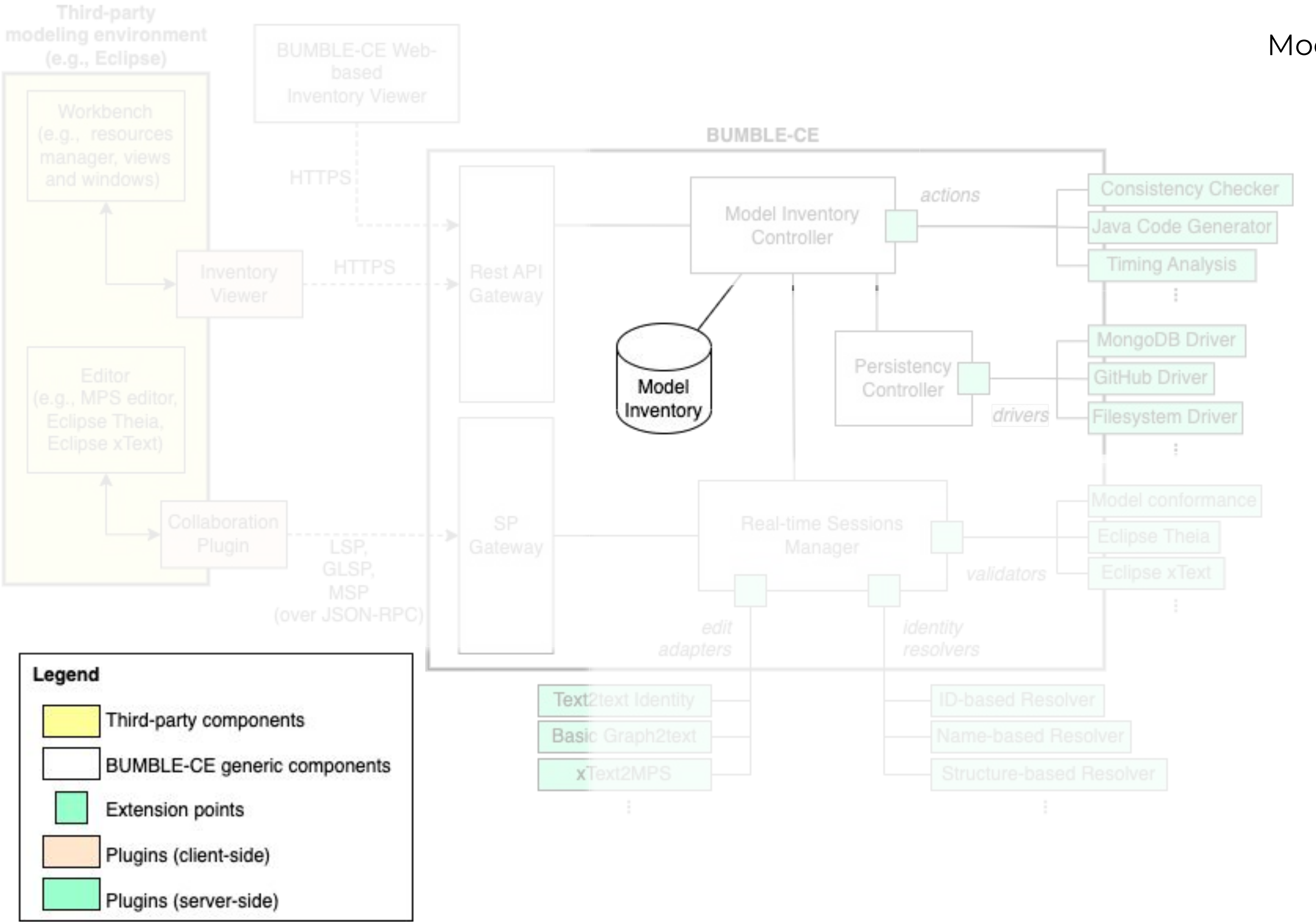
ARCHITECTURE



Stakeholders

- BUMBLE-CE admin
- MDSE engineers
- modelers

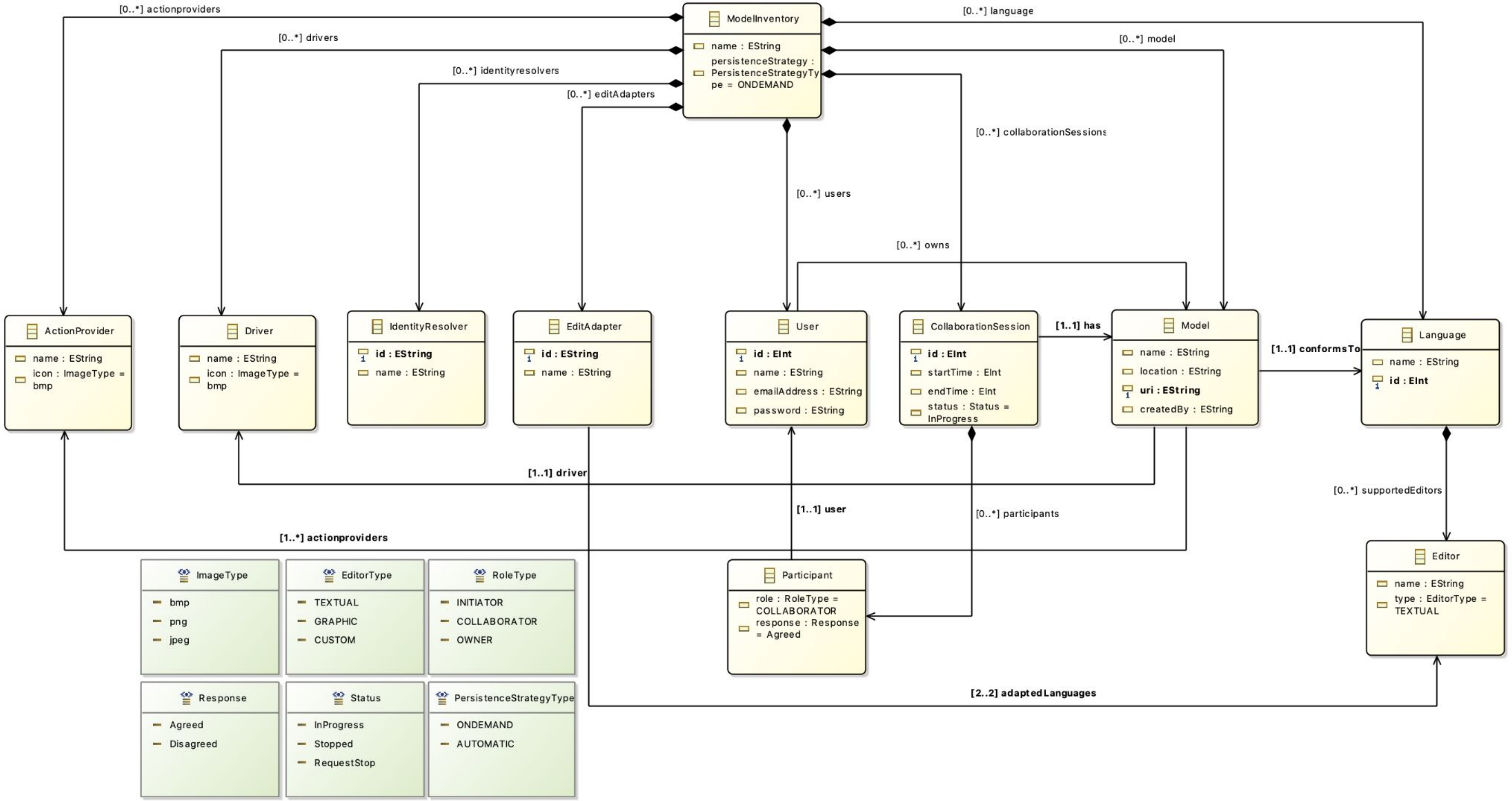
ARCHITECTURE



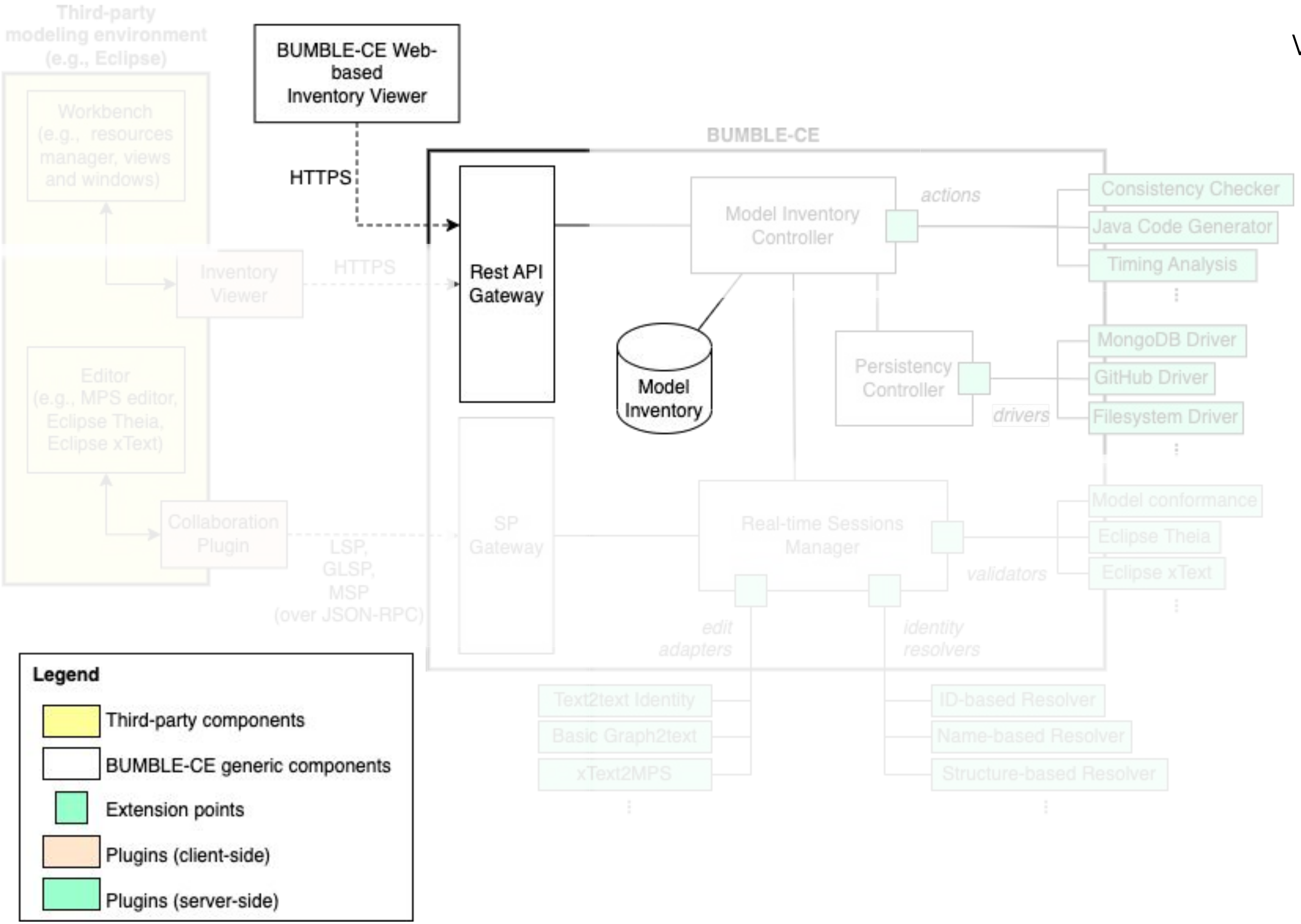
Model Inventory

- Stores models, DSML definitions of models and their relationships
- Implemented as megamodel

METAMODEL: Model Inventory



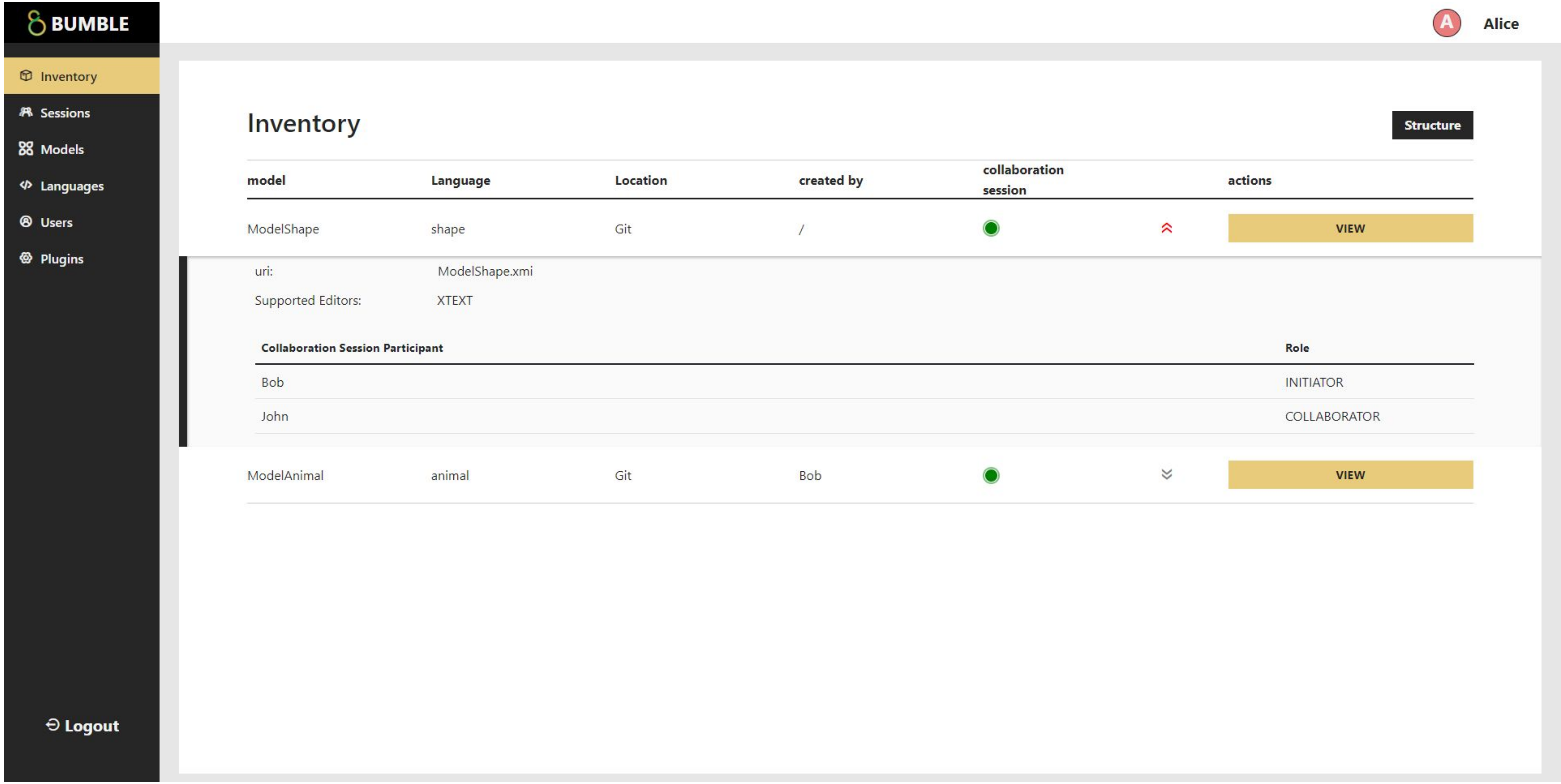
ARCHITECTURE



Web-based Model Inventory viewer

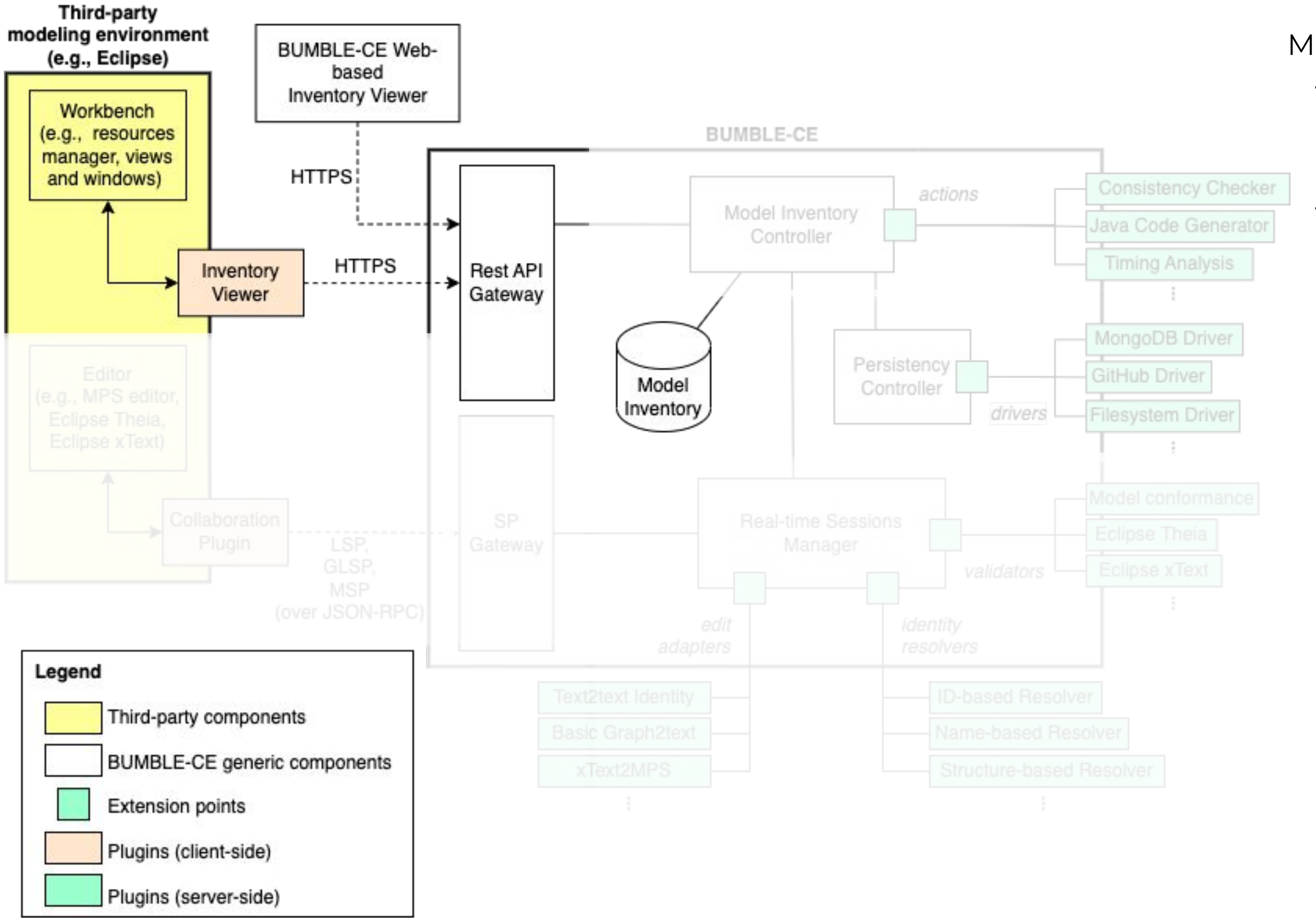
- Manipulate megamodel from model inventory
- Allows users to trigger *actions* and *drivers*

Web-based Model Inventory Viewer



Implementation of the Web-based model inventory viewer can be found in our repository.

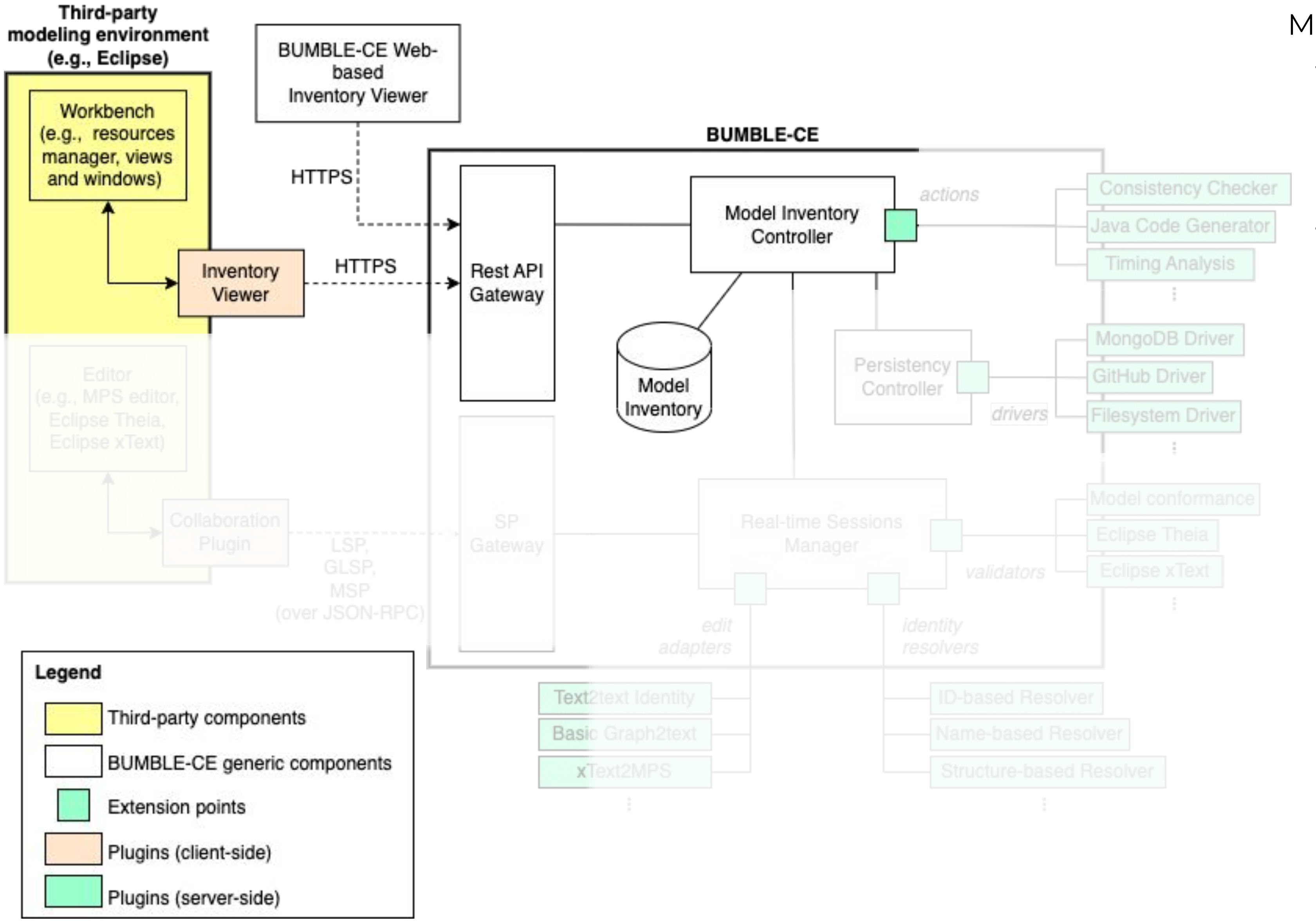
ARCHITECTURE



Model Inventory viewer

- Platform specific plugin interacting with model inventory
- Integrates API provided by Rest API gateway into the client

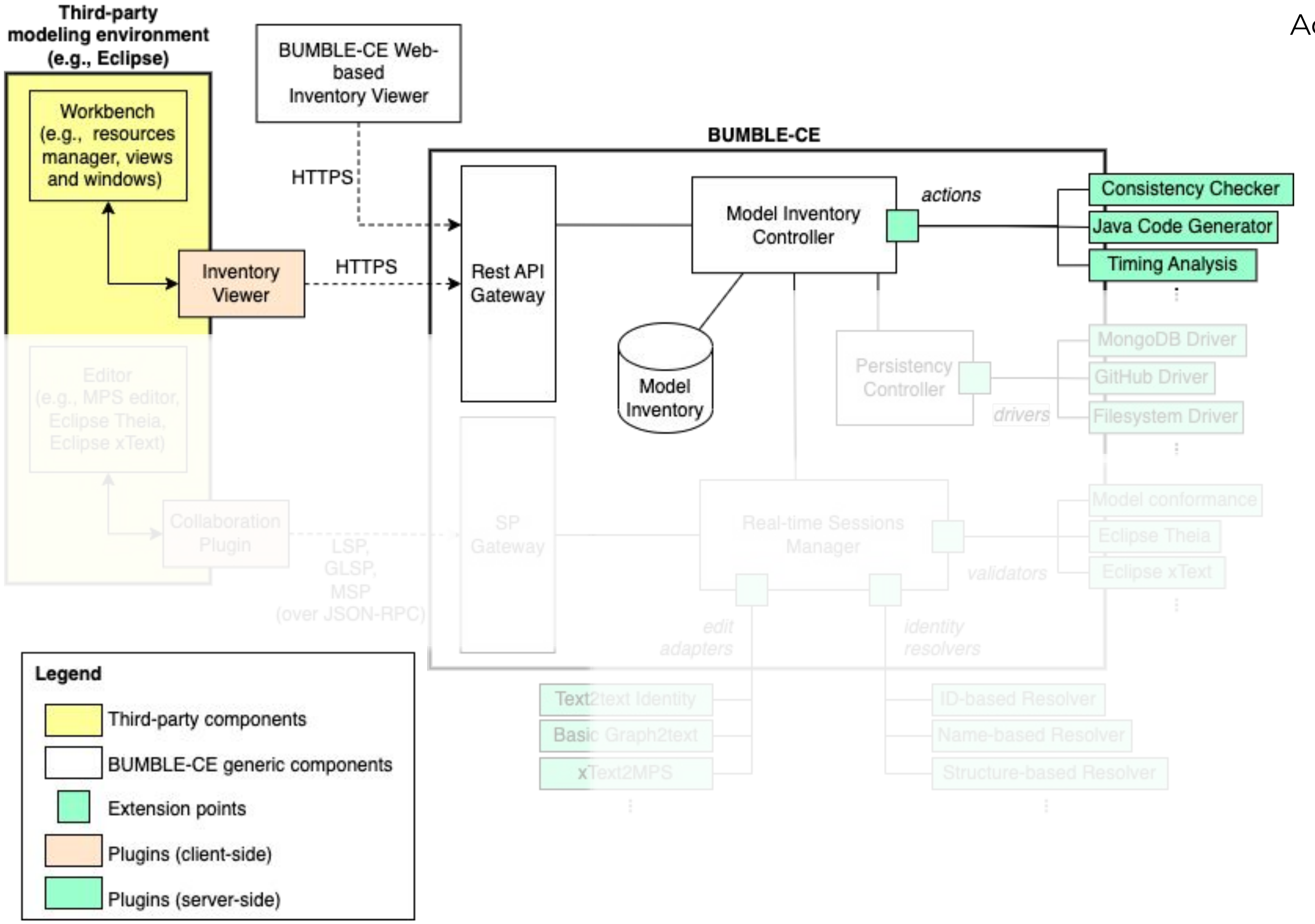
ARCHITECTURE



Model Inventory Controller

- standard functionalities for accessing and manipulating model inventory
- an extension point for third-party plugins to allow *actions*

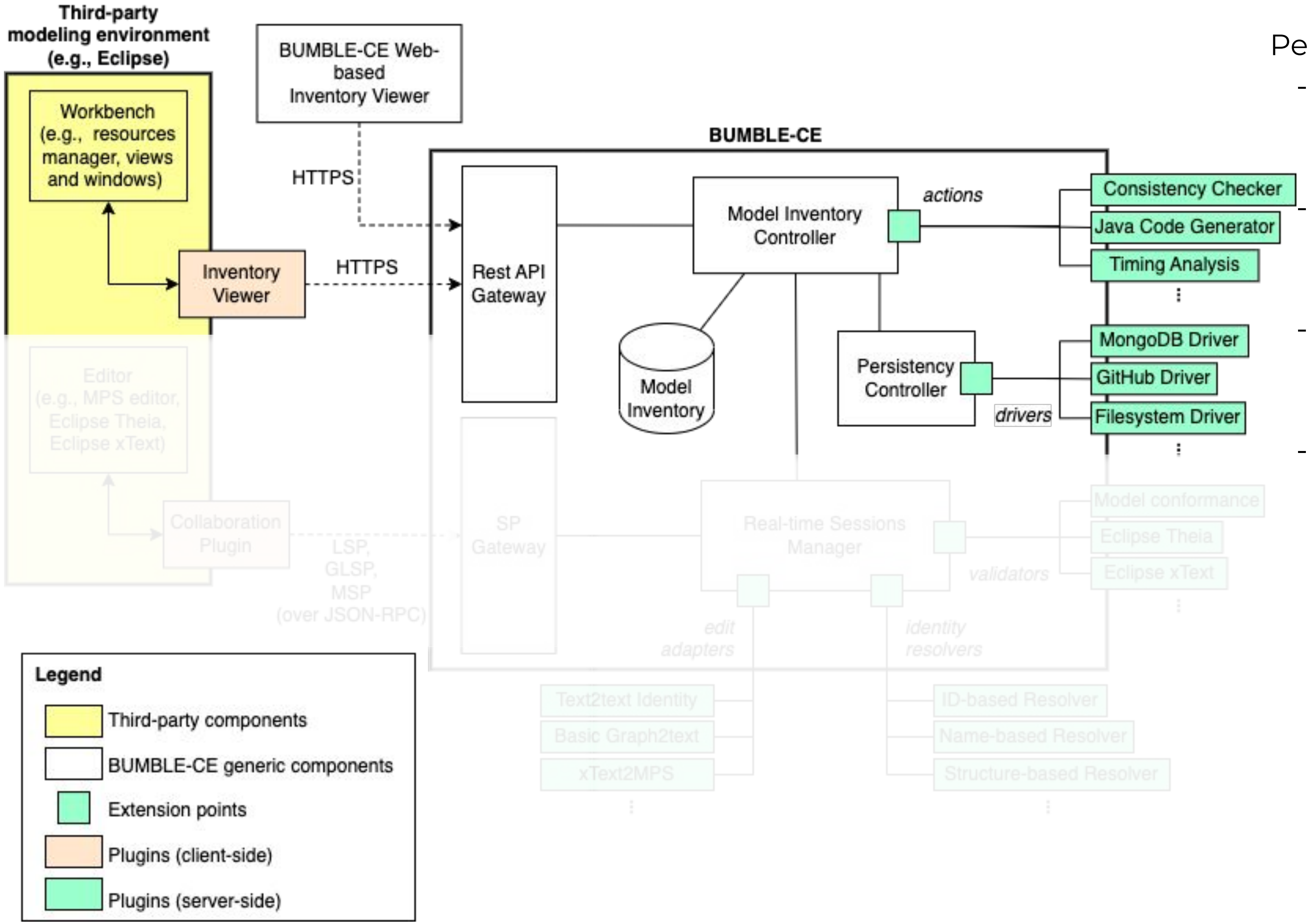
ARCHITECTURE



Actions Provider

- allows third-party actions to be called at specific moments

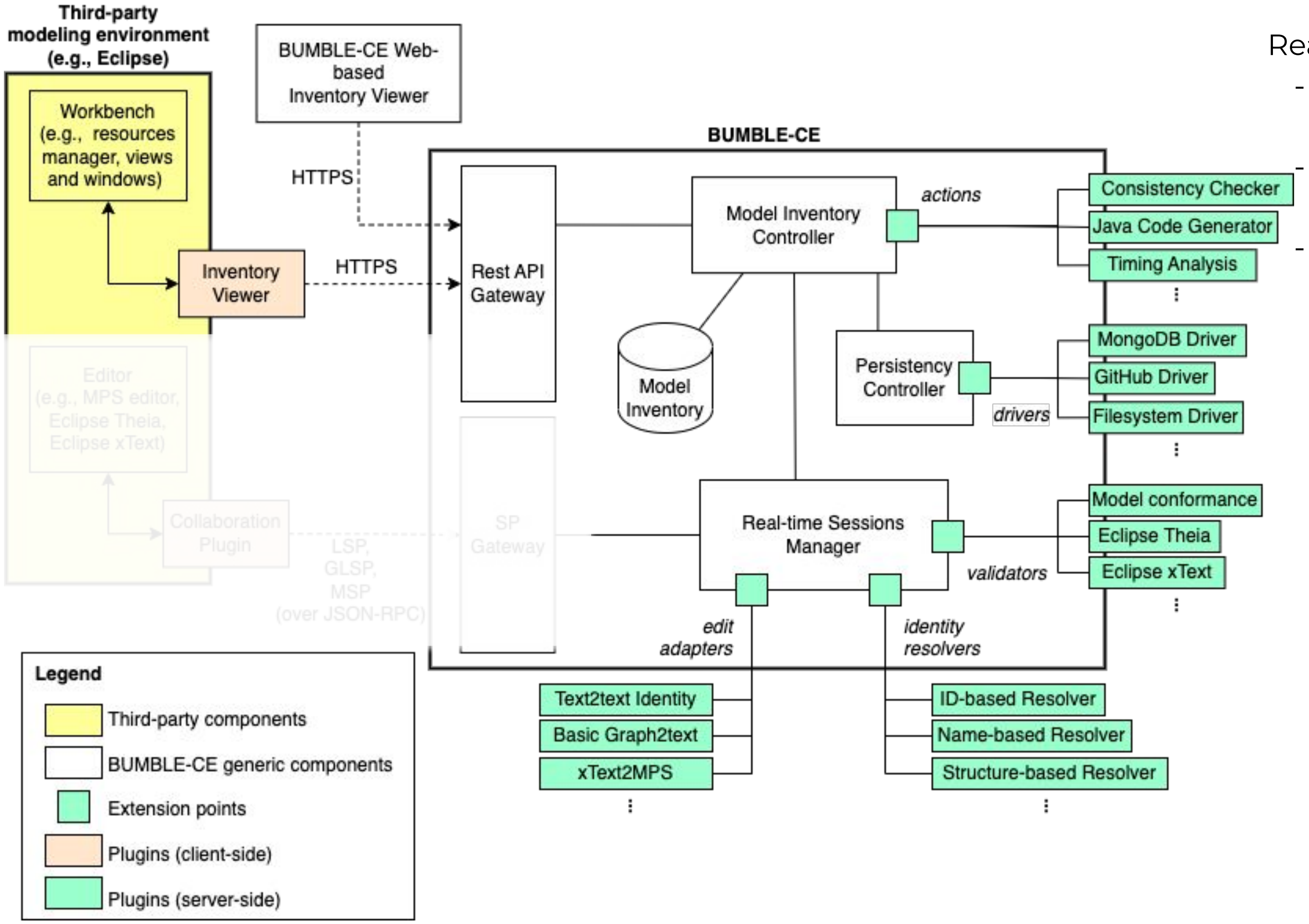
ARCHITECTURE



Persistency Controller

- Abstract layer for persisting models
- Independent of any persistence technology
- Provides generic API to Model Inventory Controller
- An extension point for third-party *drivers* e.g., for MongoDB, Git or simply Filesystem

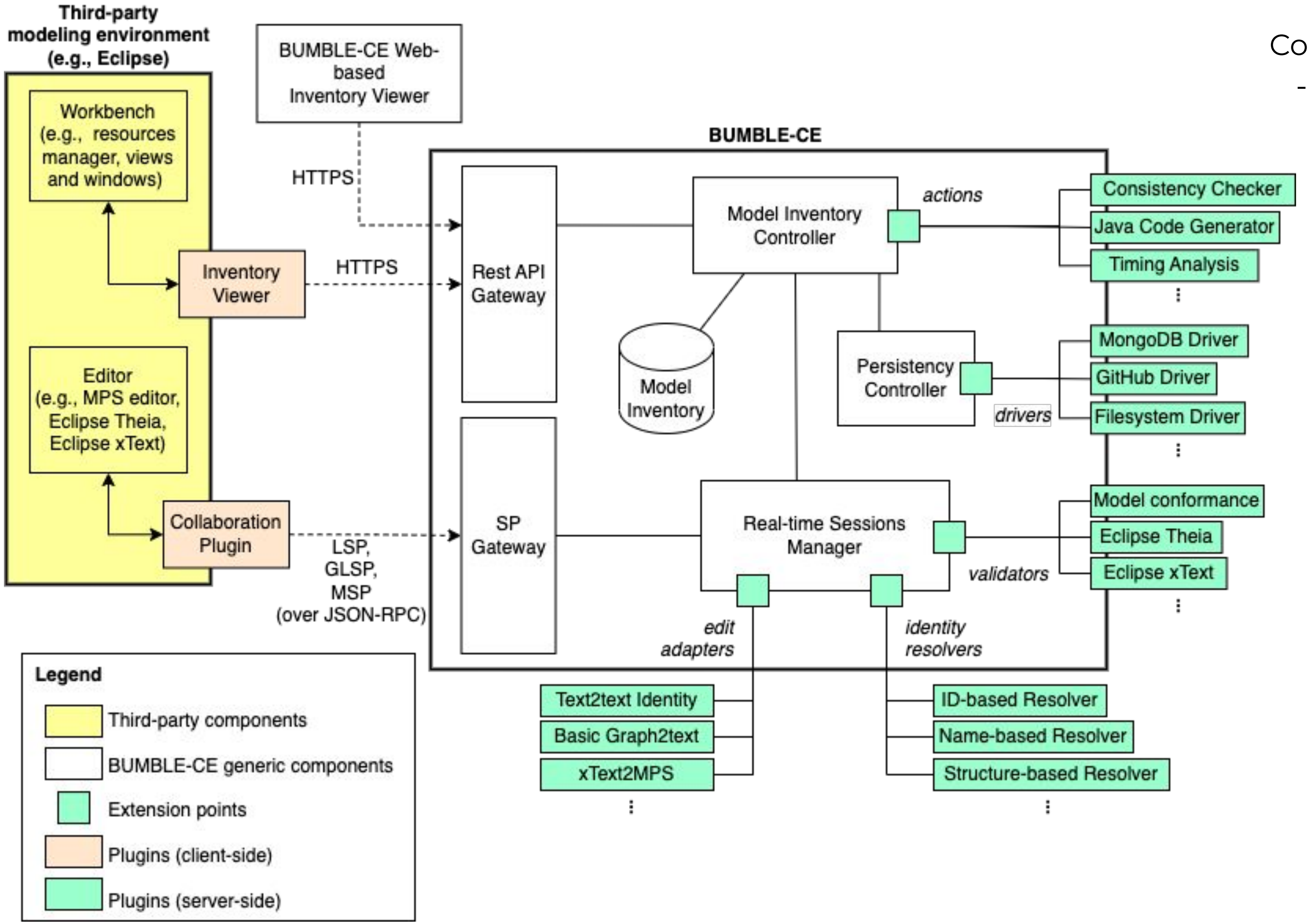
ARCHITECTURE



Real-time Sessions Manager

- brings up collaboration sessions
- propagate edit operations
- provide extension points to incorporate specific business logic

ARCHITECTURE



Collaboration plugin

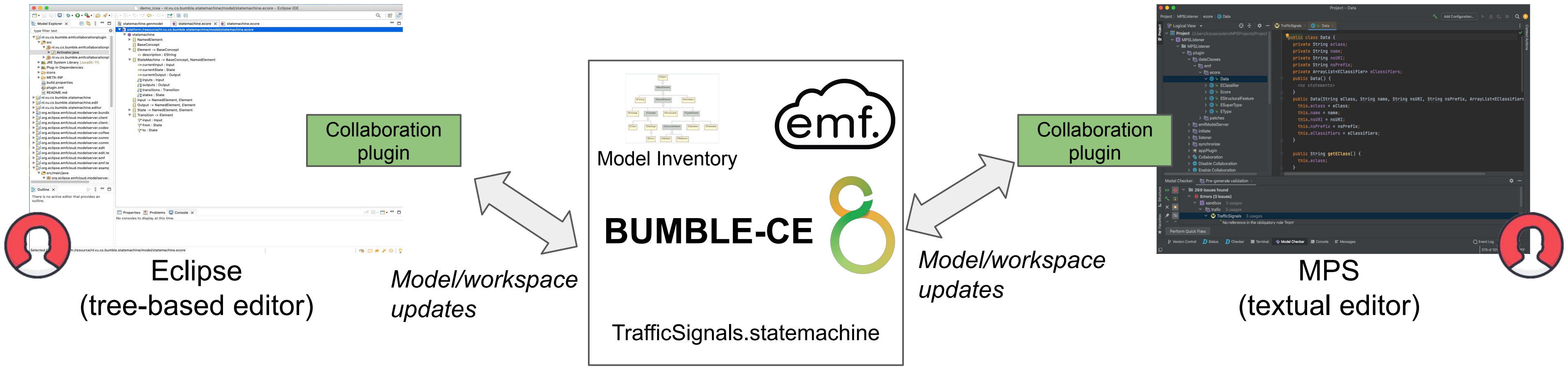
- establish and maintain bidirectional communication channel during collaboration session

Prototype Implementation

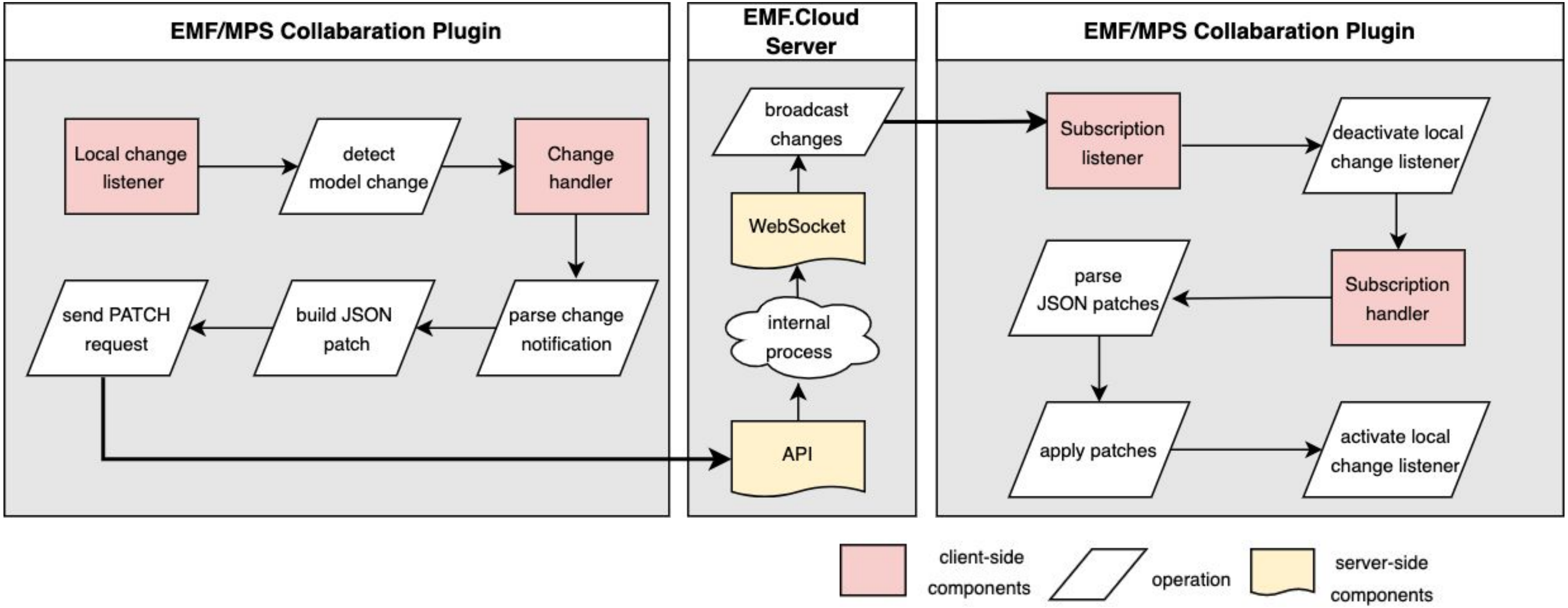
EMF Model server

- based on EMF, still supports non-EMF models
- provides set of APIs to retrieve and interact with models
- can keep several editors on the same model instance in sync
- provides command-based editing and undo/redo support

COLLABORATION SCENARIO

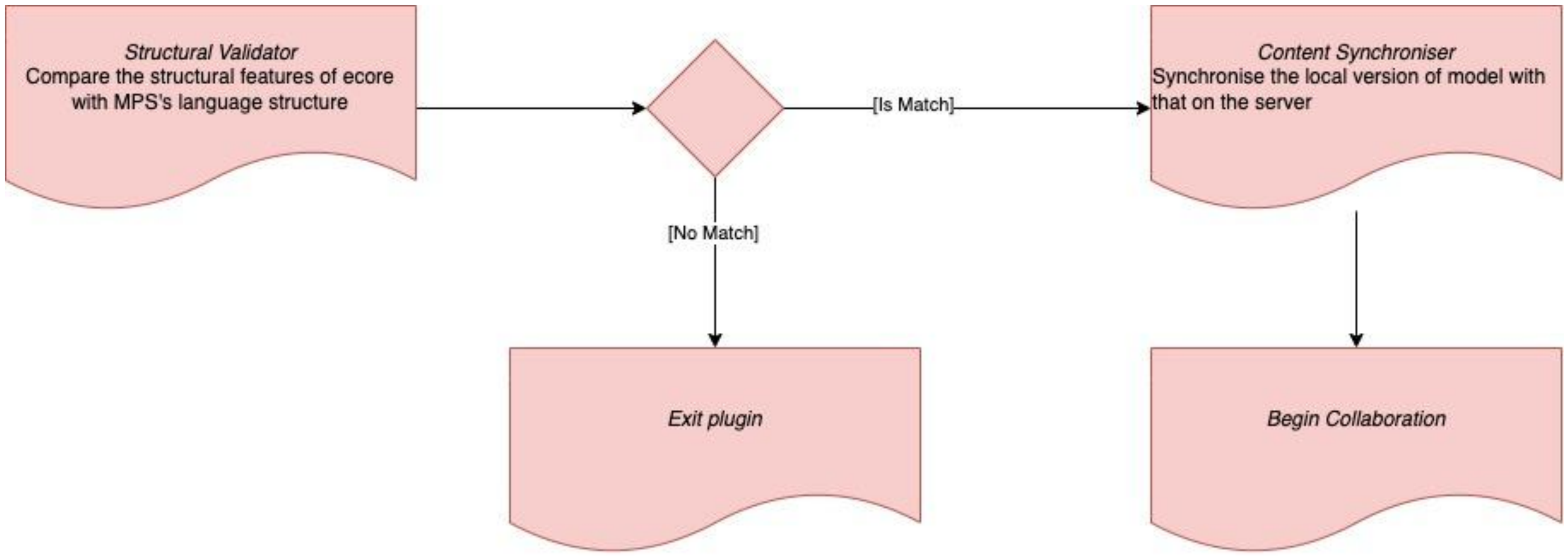


COLLABORATION PLUGIN: EMF/MPS



Implementations for EMF collaboration plugin can be found [here](#) and for MPS collaboration plugin can be found [here](#).

Validation and Synchronisation



Structural Validator

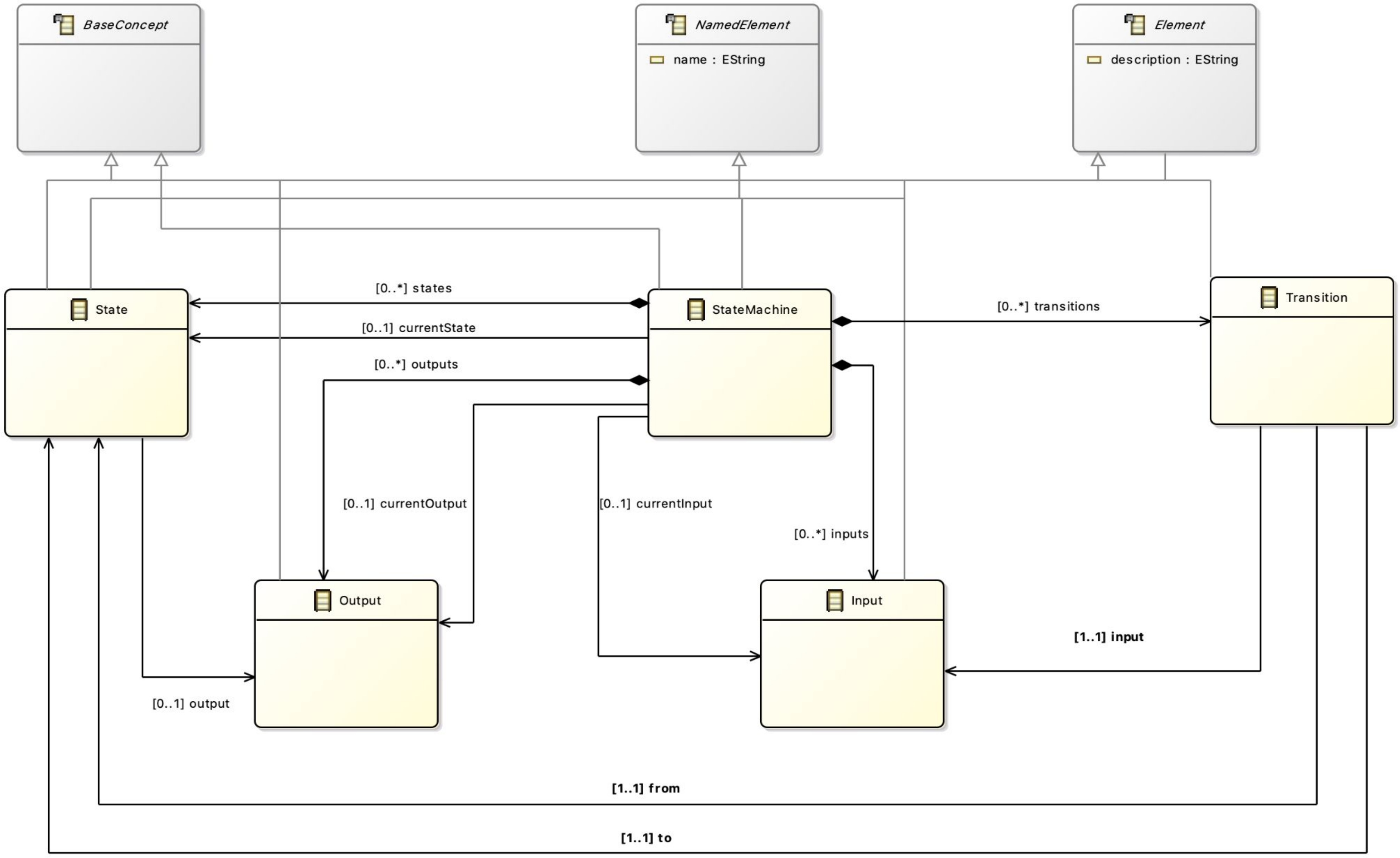
Note: Fails validation when any validation step fails.

Validation is performed such that the structural features of a given Ecore is compared with MPS's language structural features.

The validation is performed in a chronological order:

1. Ensure all EClassifiers are present by name in MPS.
2. For each EClassifier, ensure its supertype(s) are present by name in MPS.
3. For each EClassifier, ensure EStructural features are identical. To do so, the following checks are performed:
    a. References and containment links.
    b. Multiplicities.
    c. Attributes.

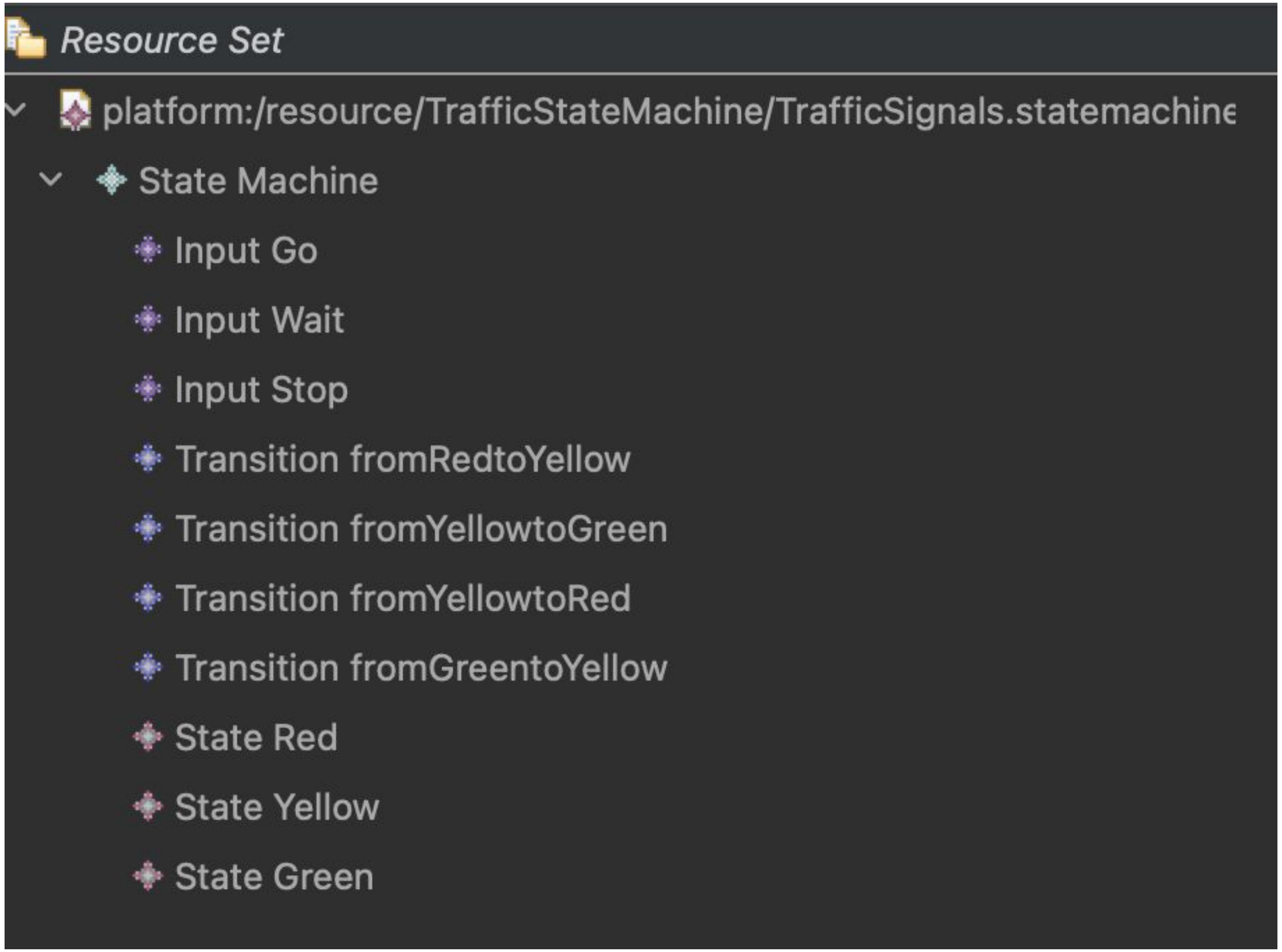
METAMODEL: StateMachine.ecore



- Developed by MVG originally in MPS
- Manually translated to EMF metamodel

MPS language structure for StateMachine can be found here ,and EMF metamodel for StateMachine can be found here.

Model: TrafficSignals.statemachine



Demo

PART 3

FACILITATED DISCUSSION AND RETROSPECTIVE SESSION

AGENDA

1. Brief round of introductions (5 minutes)
2. Guided discussion (40 minutes)

*In this focus group, answer the questions by considering a recent significant project you have been (or still are) involved in, and in which **the software architecture of the system was explicitly modelled**.*

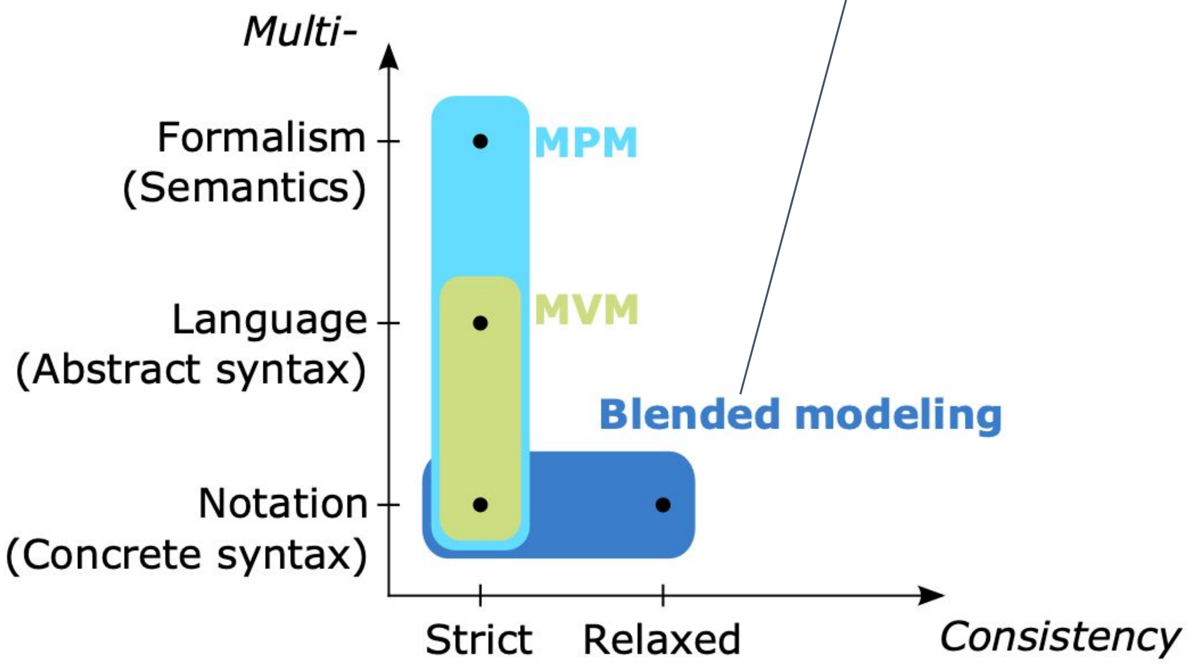
If you have been involved in more than one project like that, please consider ONLY ONE

INTRODUCTIONS

BLENDED MODELING

- Did you use some form of blended modeling?
  - If **yes**: which ones exactly?
  - If **not**: would blended modeling be beneficial in your project?

The activity of interacting seamlessly with a single model (i.e., abstract syntax) through multiple notations (i.e., concrete syntaxes), allowing a certain degree of temporary inconsistencies.



NOTATIONS

- What would be the **ideal combinations of notations** supported by blended modeling solutions?
    - Textual+graphical, textual+tabular, textual+graphical+tabular, etc.
- Would **embedded notations** be useful?
    - AKA: having notations embedded into each other
- What would be the ideal **overlap** between notations?
    - None, partial, complete

VISUALIZATION AND NAVIGATION

- Would it be useful to **visualize multiple notations at the same time**?
  - Example: a side-to-side view of both the graphical and textual representation of the architecture
- Would it be useful to **synchronously navigate among notations**?
  - Example: if the user selects a component in the graphical editor, also the cursor in the textual editor moves to that component
- Should the navigation among notations be **immediate** or **more “regulated”**?
  - Example of immediate navigation: a click or a keyboard shortcut
  - Example of regulated navigation: through multiple steps/menus

FLEXIBILITY

- **MODEL to MODELS** - Should (temporary) inconsistencies among different representations of the same model be supported?
- **MODEL to LANGUAGE** - Should (temporary) inconsistencies between the modelled architecture and the used language be supported?
- Should inconsistent models be **persisted**?

How blending can be **BENEFICIAL** for architectural descriptions?

What are the **RISKS** of using blending for architectural descriptions?

Answer the following two questions ***in general****,*
without thinking about a specific project

WRAP-UP

Final comments?

NEXT STEPS

- We will create a transcript of the recording.
  (And destroy the video immediately afterwards)
- We will integrate and synthesize the main points of the discussion
- The (draft) results will be shared with you
- You can add to this draft if there is something missing, or to be clarified
- We may contact you for further clarifications later on, before publishing the results

RELATED PUBLICATIONS

Ciccozzi, F., Tichy, M., Vangheluwe, H., & Weyns, D. (2019, September). Blended modelling-what, why and how. In 2019 ACM/IEEE 22nd International Conference on Model Driven Engineering Languages and Systems Companion (MODELS-C) (pp. 425-430). IEEE.

David, I., Latifaj, M., Pietron, J., Zhang, W., Ciccozzi, F., Malavolta, I., Raschke, A., Steghöfer, JP., Hebig, R. (2022, June) Blended modeling in commercial and open-source model-driven software engineering tools: A systematic study. Software and Systems Modeling. 1-33.

Latifaj, M., Ciccozzi, F., Mohlin, M., & Posse, E. (2021, September). Towards Automated Support for Blended Modelling of UML-RT Embedded Software Architectures. In ECSA (Companion).

Latifaj, M., Ciccozzi, F., Anwar, M. W., & Mohlin, M. (2022, August). Blended Graphical and Textual Modelling of UML-RT State-Machines: An Industrial Experience. In Software Architecture: 15th European Conference, ECSA 2021 Tracks and Workshops; Växjö, Sweden, September 13–17, 2021, Revised Selected Papers (pp. 22-44). Cham: Springer International Publishing.

Latifaj, M., Ciccozzi, F., & Mohlin, M. (2023). Higher-order transformations for the generation of synchronization infrastructures in blended modeling. *Frontiers in Computer Science*, *4*.

Latifaj, M. (2022, October). The path towards the automatic provision of blended modeling environments. In Proceedings of the 25th International Conference on Model Driven Engineering Languages and Systems: Companion Proceedings (pp. 213-216).

Voogd, S.N, Aslam, K., van Gool, L., Theelen, B., Malavolta, I. (2021) Real-Time Collaborative Modeling across Language Workbenches - a Case on Jetbrains MPS and Eclipse Spoofax HoWCoM workshop colocated with MODELS.

David, I., Aslam, K., Faridmoayer, S., Malavolta, I., Syriani, E., Lago, P. (2021) Collaborative Model-Driven Software Engineering: A Systematic Update (MODELS, 2021

Other related publications and training material can be found in the official BUMBLE repository.