Model transformations and validations

Börcsök József

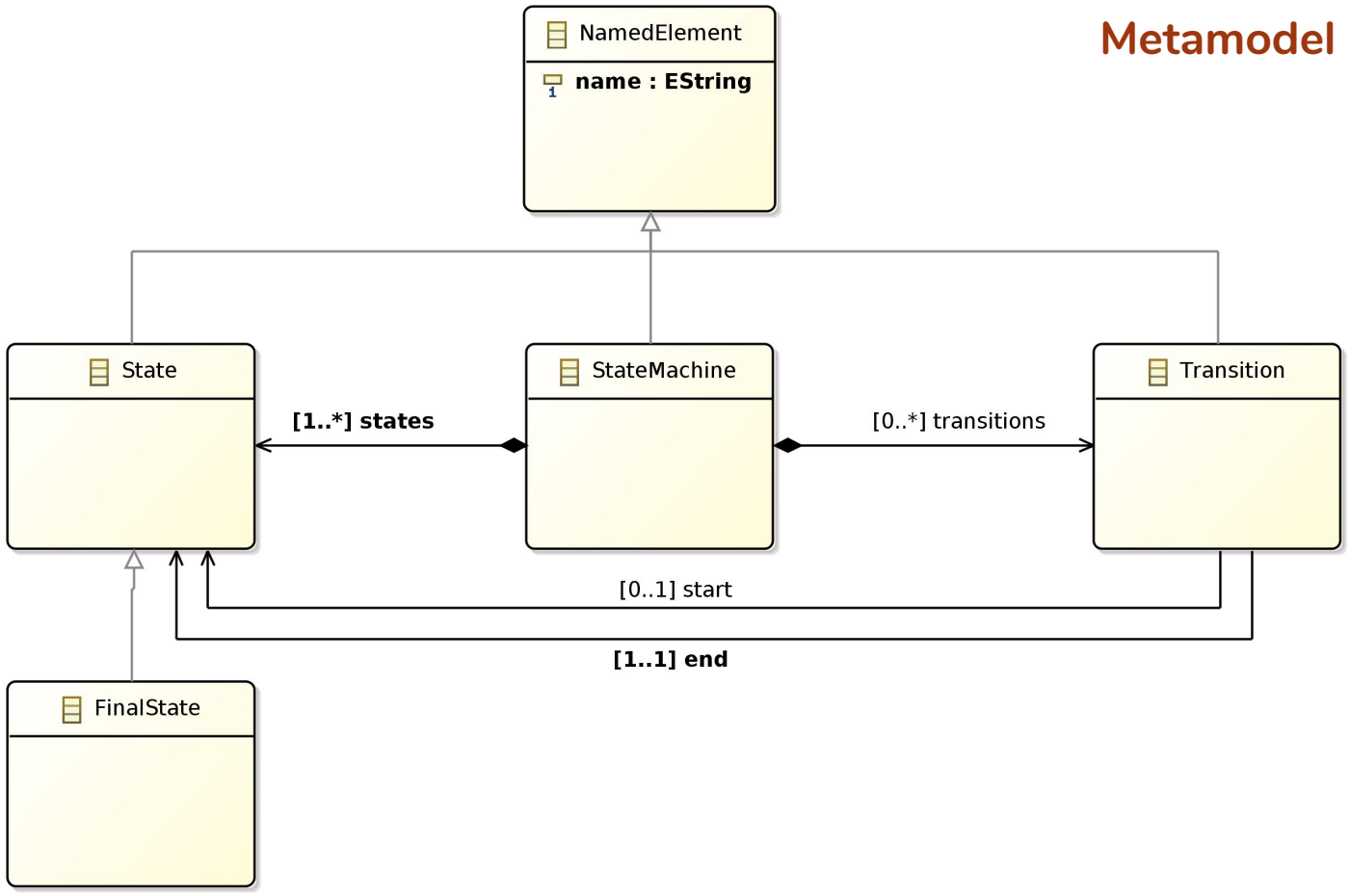
BlackBelt Technology

Models and metamodels

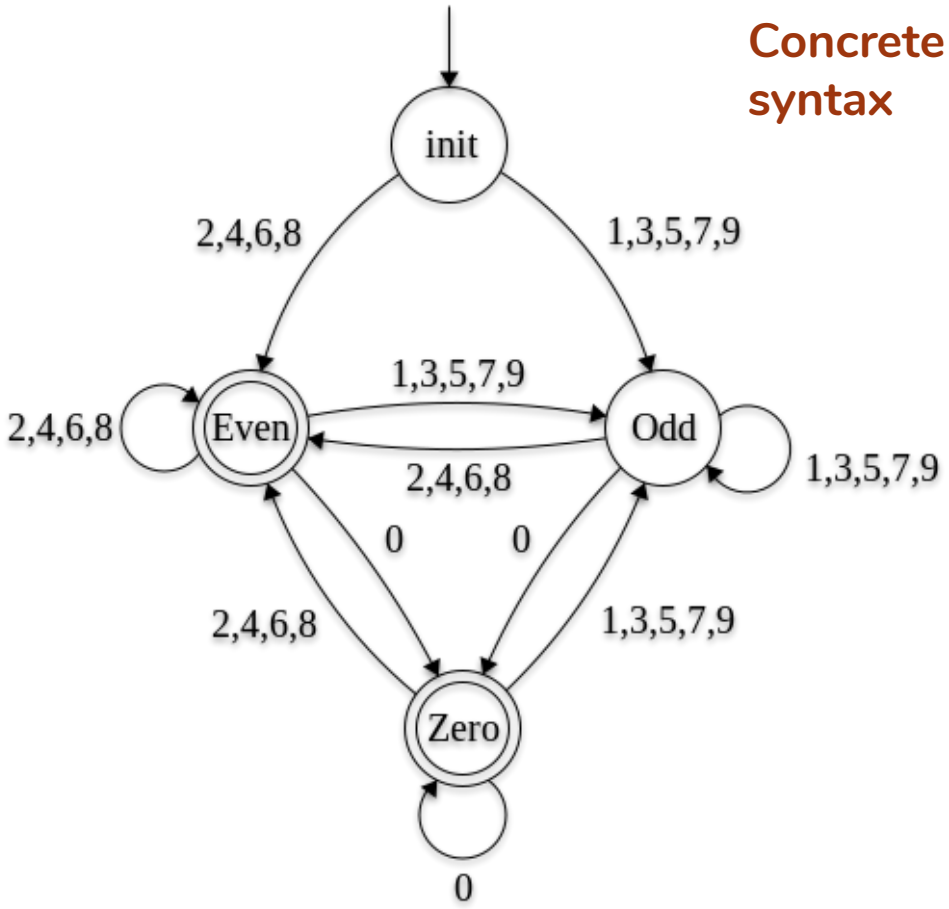
- **Model**: formal description of phenomena of interest
- **Metamodel**: description of *abstract syntax*, *concrete syntax* (graphical or textual) and *semantics* using modeling infrastructure
- **(Formal) grammar**: a set of production rules for strings

Example: State machine metamodel and model (even numbers, concrete syntax)

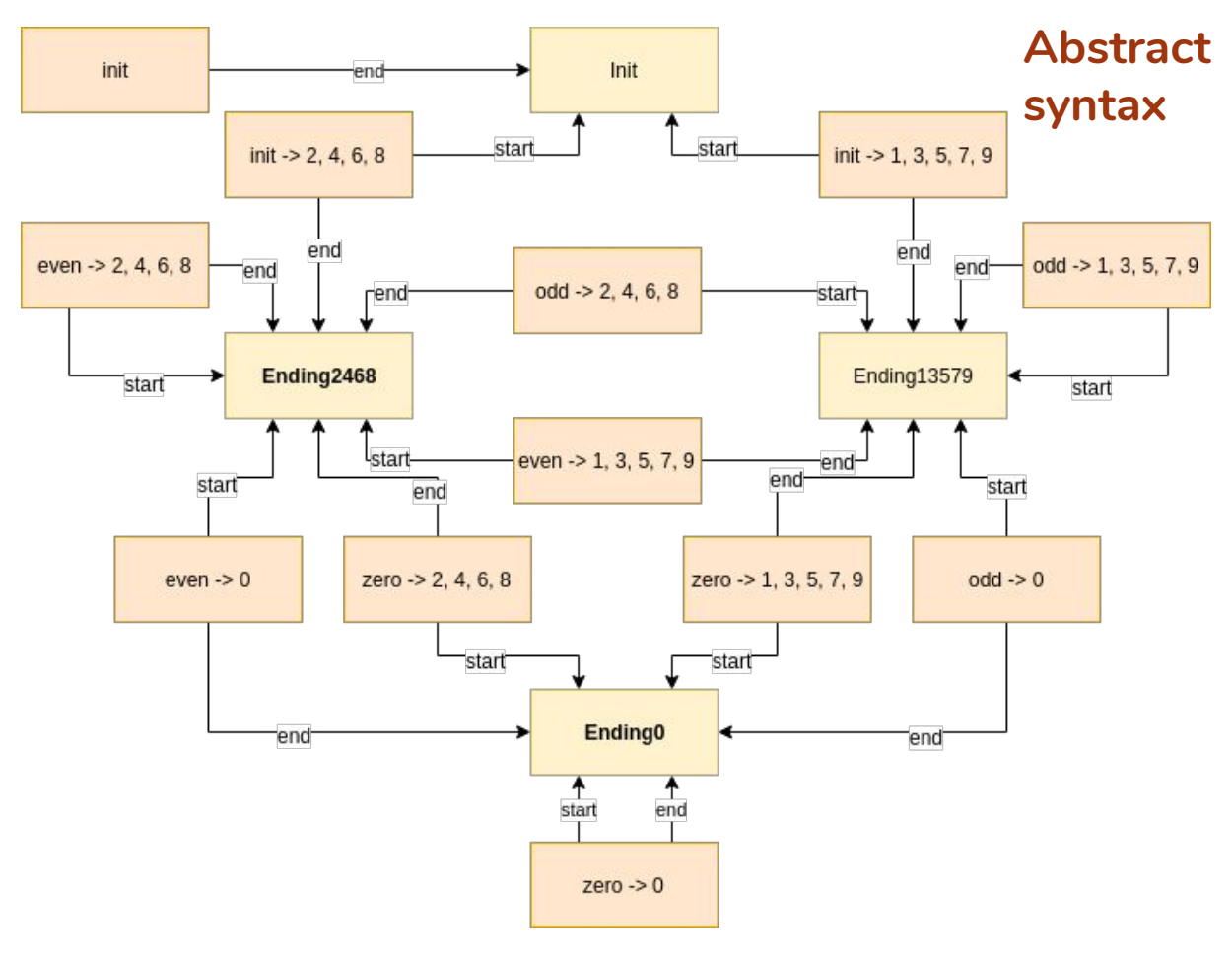
Metamodel



Concrete syntax



Example: Abstract syntax model and grammar of even numbers



ANTLR4 grammar

```
grammar EvenNumbers;
parseEvenNumbers
 : EVEN_NOT_0 ending2468 | ODD ending13579 ;

ending2468
 : EVEN_NOT_0 ending2468 | ODD ending13579 | ZERO ending0
 | terminate ;

ending13579
 : EVEN_NOT_0 ending2468 | ODD ending13579 | ZERO ending0 ;

ending0
 : EVEN_NOT_0 ending2468 | ODD ending13579 | ZERO ending0
 | terminate ;

terminate : EOF ;

ZERO : [0] ;
ODD : [13579] ;
EVEN_NOT_0 : [2468] ;
WS : [ \t\r\n]+ -> skip ;
```

Metamodel syntax and semantics

- **Syntax rules**: defined by metamodel
  - example: all transitions must have an end state
- **Semantic rules**: defining behaviour
  - ie. invariant expressions, algorithms
  - standard: **OCL**
  - example: at least 1 final state, exactly 1 initializer transition
  - wired (not separated) into grammar

Examples: models, metamodels, grammars

Model / sentence	Metamodel	Grammar
XML	XSD	DTD
SQL statement	SQL metamodel	SQL BNF
State machine (SM)	SM metamodel in UML	Regular grammar def.
Swagger (JSON/YAML)	Swagger spec.	

Grammars vs. metamodels

Metamodel	Grammar
trees and graphs	mostly trees
abstract syntax first	concrete syntax first
focus is on syntax, rules and operations/transformations	well-integrated with semantics definitions
EMF/ECore, MOF	EBNF
unified	not unified

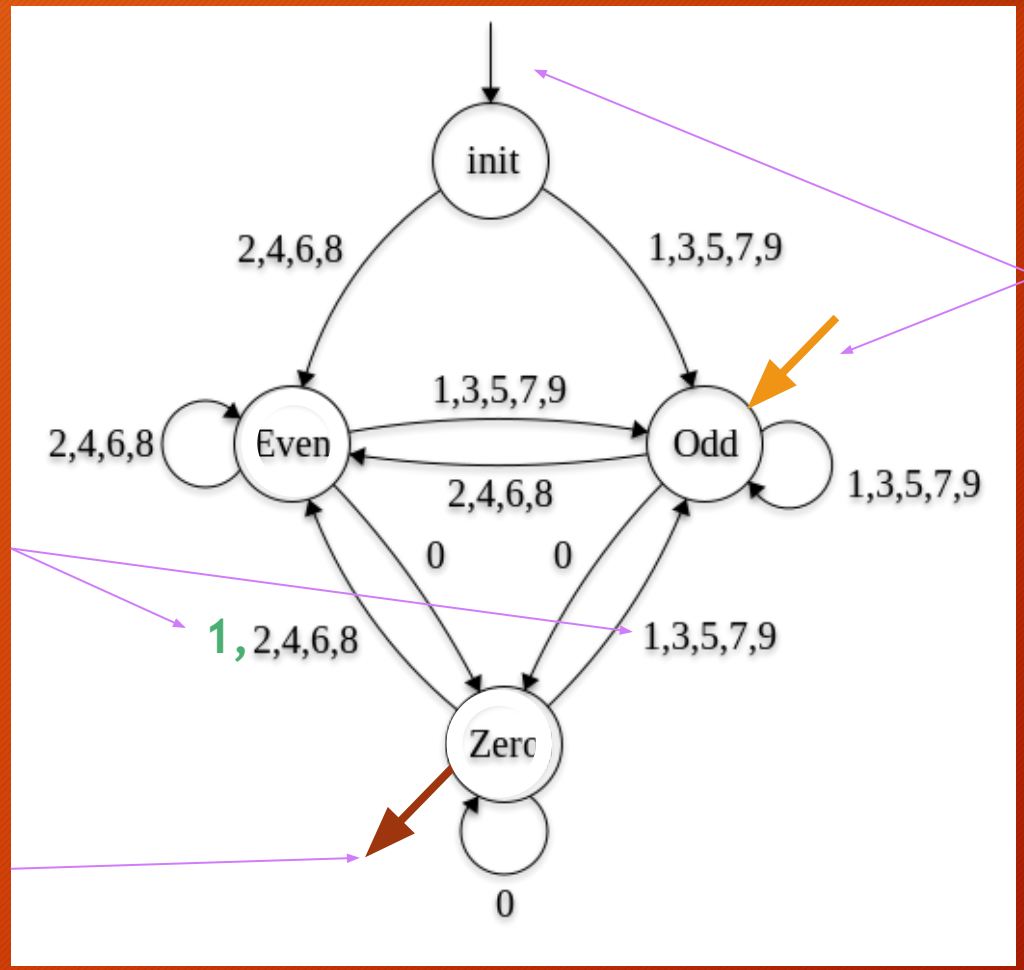
Why metamodel (and grammar)?

- *Present and process large amounts of documentation*
- *Generate valid, well-formed text*
- *Supporting software tools*
  - Epsilon: processing tool for ECore models
- Checking syntax and semantics of a model, model transformations
- Generating text from models
- Model comparison is easier

Validations

- **Syntax**: model is not loadable if invalid
  - tools must keep it valid
- **Semantics**: model editors can fix semantic errors
  - tools should support semantic validation
  - error messages are created for model editors
  - dependencies of rules, severity of messages
- Detect constraint violations as soon as possible

Example: validation rules of even numbers SM



Multiple initial states

State machine is non-deterministic (WARNING)

No final state(s)

Transition has no end reference (syntax error)

Example: validation rule as EVL (Epsilon)

```
context JUDOPSM!EntityType {

    // entity has no multiple attributes with the same name
    constraint AttributeNameIsUnique {

        check: self.attributes
                    .select(a | self.attributes
                                    .excluding(a)
                                    .selectOne(a2 | a2.name == a.name)
                                    .isDefined())
                    .size() == 0

        message: "Multiple attributes are added to entity "+self.name+" with the same name"
    }

}
```

Transformations

- **Declarative rules**
  - based on metamodels
  - operating on input model instance(s)
  - producing output model instance(s)
- Polymorphic behavior and code **reuse**
  - composition, inheritance, references

Example: transform UML classes to RDBMS

- UML class -> SQL table
  - attribute -> field
  - reference -> field with foreign key (ownership + multiplicity!)
  - containment -> table
- name of UML named element -> SQL name
- OCL invariant -> SQL constraint, trigger

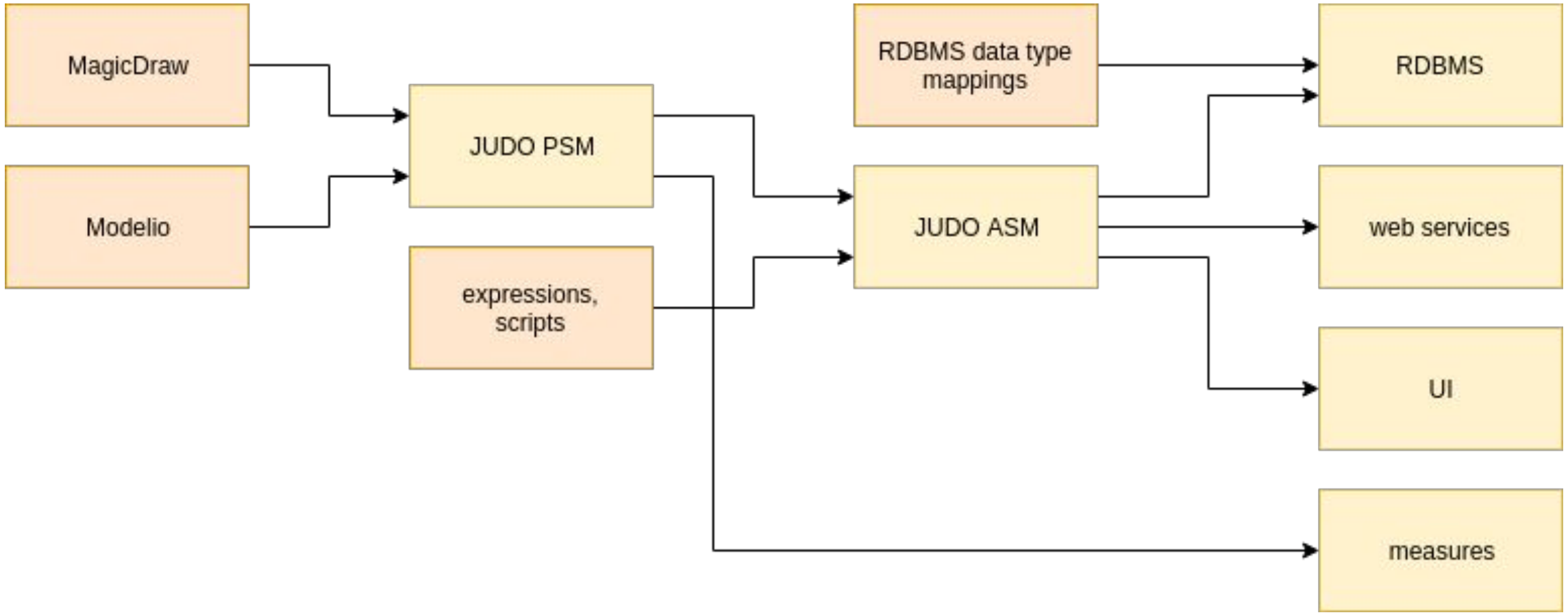
Example: transformation rule as ETL (Epsilon)

```
rule CreateUnit
    transform s : JUDOPSM!Unit
    to t : MEASURES!Unit {

        // copy attributes
        t.name = s.name;
        t.symbol = s.symbol;
        t.rateDividend = new Native("java.math.BigDecimal")(s.rateDividend.toString());
        t.rateDivisor = new Native("java.math.BigDecimal")(s.rateDivisor.toString());

        // add transformed unit into equivalent (transformed) measure
        var m = JUDOPSM!Measure.all.selectOne(m | m.units.contains(s)).equivalent();
        m.units.add(t);
}
```

Chain of model processing

- **Iterative** processing
  - source of single-step processing is too complex (or incomplete)
  - validate input models (not created by transformations)
  - transform model adding more technical details
  - additional input models can be attached too
- **Trace model** for feedbacks
- Separate roles that are working on input of iteration steps

Main model processing steps of JUDO



Extending models

- **Additional source models** (not result of transformations)
  - adding more technical details by modelers and developers
- **Annotations** (not defined by metamodel)
  - key-value pairs or structured metadata
  - model processors can read/use them
- **Dynamic instances**
  - types are defined by metamodels, extendable by models too

Resources

- Metamodels vs. grammar:
  - https://www.sciencedirect.com/science/article/pii/S0167642314002457
- State machine sources:
  - https://drive.google.com/file/d/1YcOnOpTzHpXBw7tLPRzS4mfzKKxPQgsp/view?usp=sharing
- Eclipse Epsilon:
  - https://www.eclipse.org/epsilon/