TECHNISCHE
UNIVERSITÄT
DRESDEN

Reference Attribute Grammars for Metamodel Semantics

Christoff Bürger[1], Sven Karol[2], Christian Wende[3], Uwe Aßmann

SLE 2010, Eindhoven, 12.10.2010

[1] Research granted by the ESF and the free-state of Saxony
[2] Research granted by the DFG (German Research Foundation) within the Project HyperAdapt
[3] Research granted by the European Commission within the FP7 project MOST #216691.

Contents

- **Motivation**
- **RAGs and Metamodel Semantics**
- **The JastEMF Approach**
- **Remarks and Observations**
- **Conclusion and Outlook**

01 Motivation

Benefits of Metamodelling

Metamodelling is a standardisation process with the following benefits:

- MM 1 Metamodelling Abstraction
- MM 2 Metamodelling Consistency
- MM 3 Metamodel Implementation Generators
- MM 4 Metamodel/Model Compatibility
- MM 5 Tooling Compatibility

However, metamodelling leaks convenient mechanisms for semantics specification.

01 Motivation

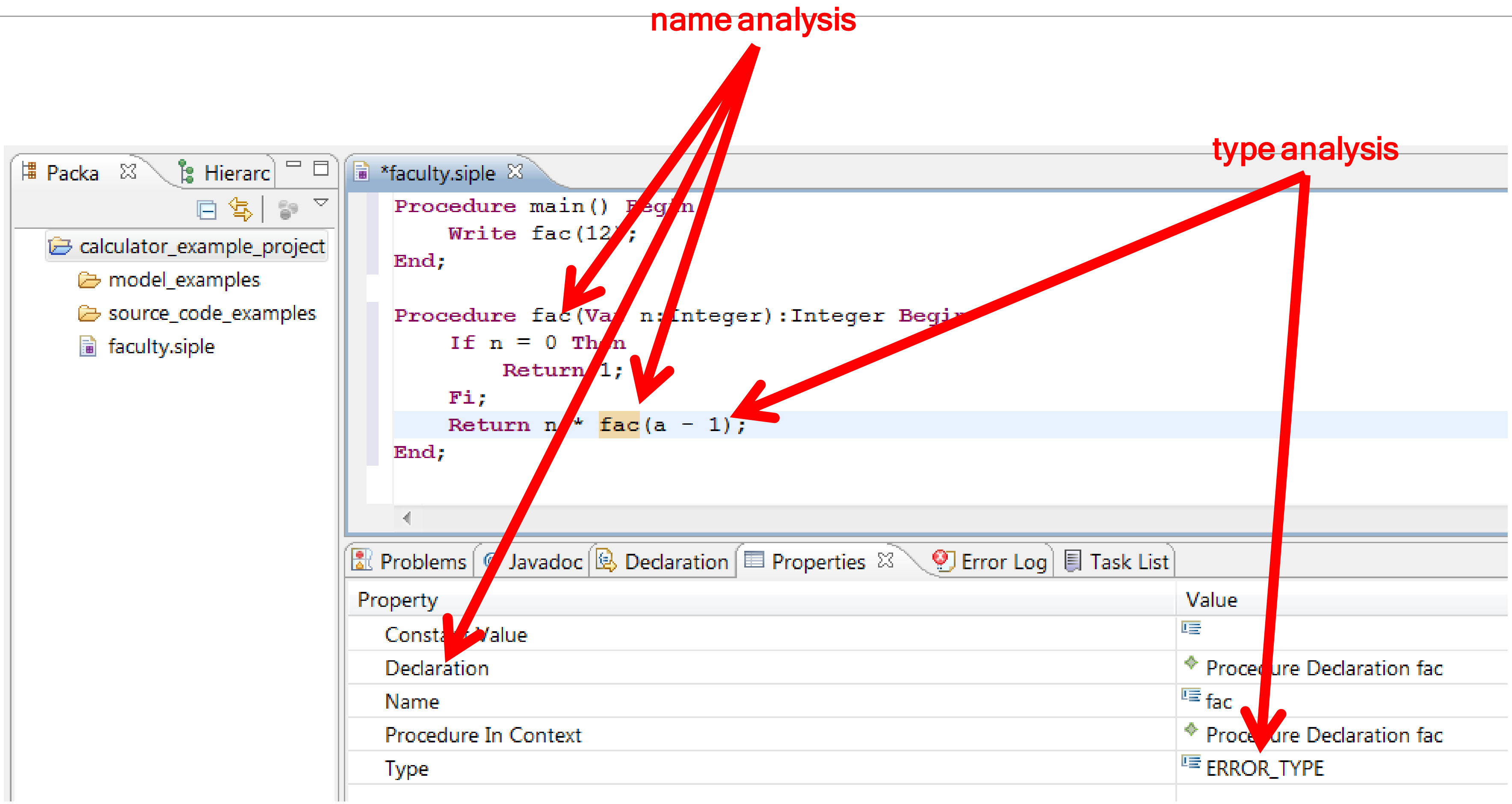
Benefits of Attribute Grammars in Compiler Construction

AGs are very convenient to specify semantics for tree structure with the following benefits:

- AG 1: Declarative Semantics Abstraction
- AG 2: Semantics Consistency
- AG 3: Semantics Generators
- AG 4: Semantics Modularity

Claim: A combination of MM and AGs enables *semantics integrated metamodelling* and leads to more successful and reliable tool implementations.

01 Motivation: Example I



name analysis

type analysis

```
Procedure main() Begin
    Write fac(12);
End;

Procedure fac(Var n:Integer):Integer Begin
    If n = 0 Then
        Return 1;
    Fi;
    Return n * fac(a - 1);
End;
```

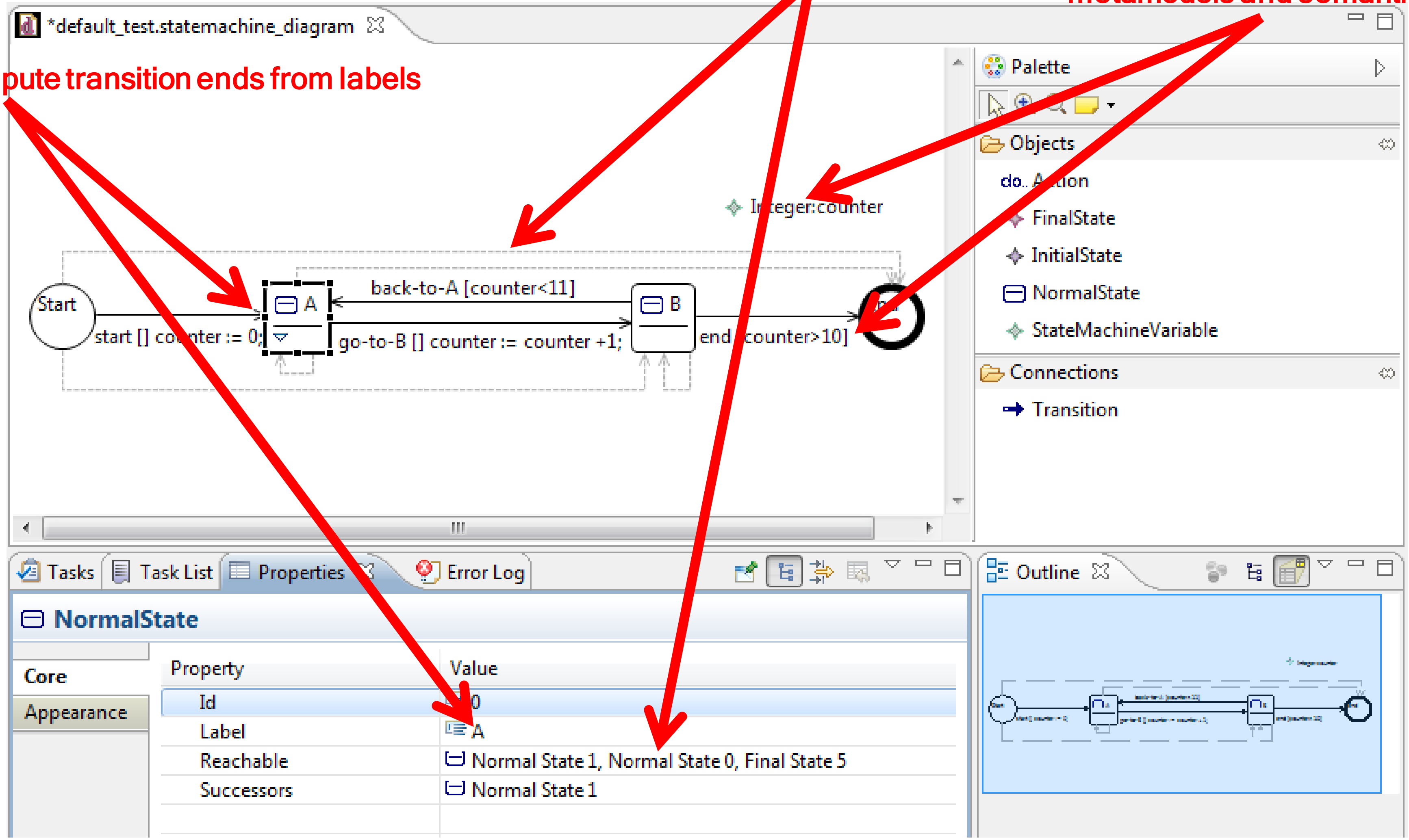
Property	Value
Constant Value	
Declaration	Procedure Declaration fac
Name	fac
Procedure In Context	Procedure Declaration fac
Type	ERROR_TYPE

01 Motivation: Example II

compute closure

reuse of metamodels and semantics

compute transition ends from labels



NormalState

Property	Value
Id	0
Label	A
Reachable	Normal State 1, Normal State 0, Final State 5
Successors	Normal State 1

02 RAGs and Metamodel Semantics

A few general Words about Semantics

Semantics is

- Always specified w.r.t. well defined structures
- Reasoning about structures to derive information or to extend/manipulate it

The complicated part of semantics is

- Distributing local information across the structure
- Combining such information and
- Further redistribute the results

AGs are very convenient to specify semantics for <u>tree structures</u>, if the structure is not changed or only extended.

02 RAGs and Metamodel Semantics

Syntax and Semantics for Ecore

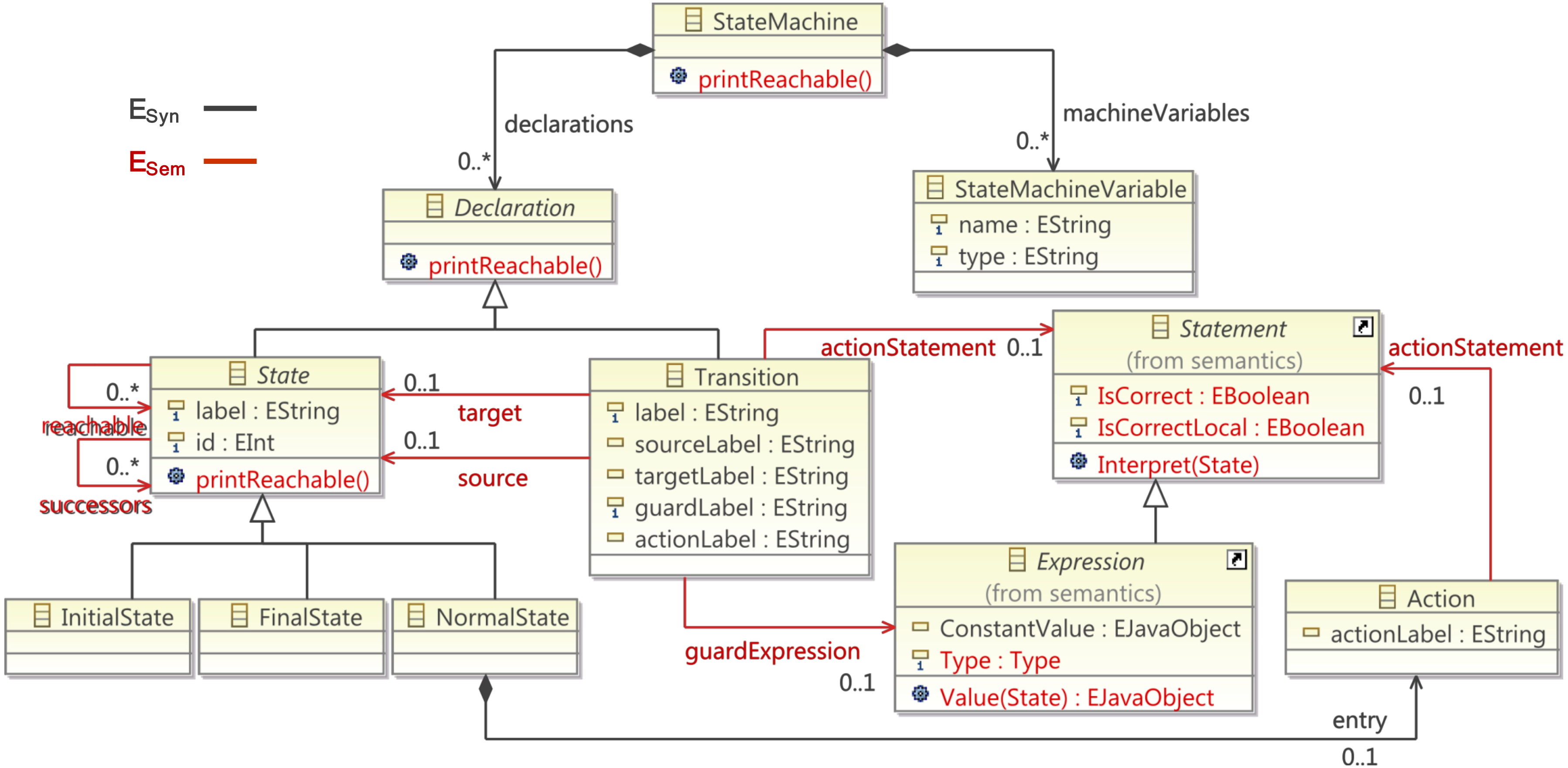
Syntax in Ecore	Syntax in RAGs
EClass	AST Node Type
EReference[containment]	Non Terminal
EAttribute[non-derived]	Terminal

E_{Syn}

Semantics Interface in Ecore	Semantics in RAGs
EAttribute[derived]	[synthesized|inherited] attribute
EAttribute[derived,multiple]	collection attribute
EReference[non-containment]	collection attribute, reference attribute
EOperation[side-effect free]	[synthesized|inherited] attribute

E_{Sem}

02 RAGs and Metamodel Semantics



(Ecore-based, extended version of Statemachine example in Hedin, G.: Generating Language Tools with JastAdd. In: GTTSE '09. LNCS ,Springer (2010))

02 RAGs and Metamodel Semantics

Intermediate Conclusion

Ecore (EMOF in general) separates model instances into

- A tree structure (AST) and
- A graph structure based on references between tree nodes (ASG)

In language theory and compiler construction

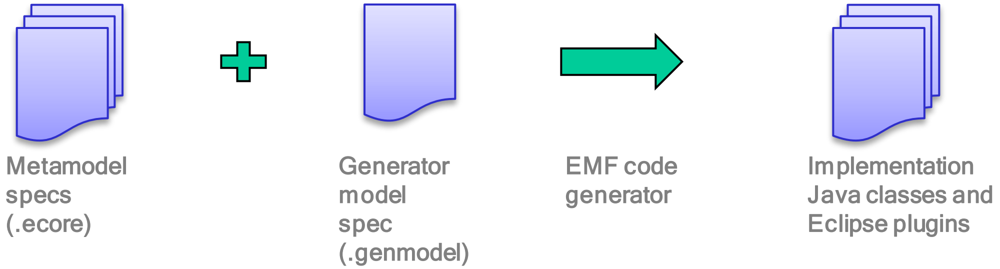
- *context-free grammars* specify context-free structures (ASTs)
- Reference attribute grammars (RAGs) are a well-known concept to specify ASGs based on ASTs and to reason about ASGs

RAGs are well suited to specify semantics of Ecore-based Metamodels.

03 The JastEMF Approach

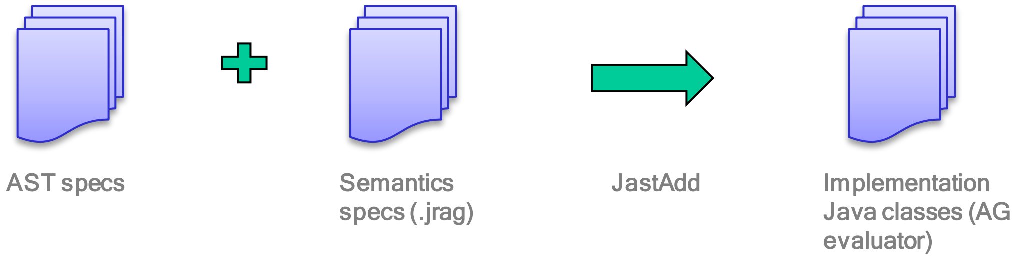
Eclipse Modelling Framework (EMF) & JastAdd

EMF basic generation process


Metamodel specs (.ecore)
Generator model spec (.genmodel)
EMF code generator
Implementation Java classes and Eclipse plugins

Metamodel implementation without semantics (semantic stubs/interfaces)

JastAdd basic generation process

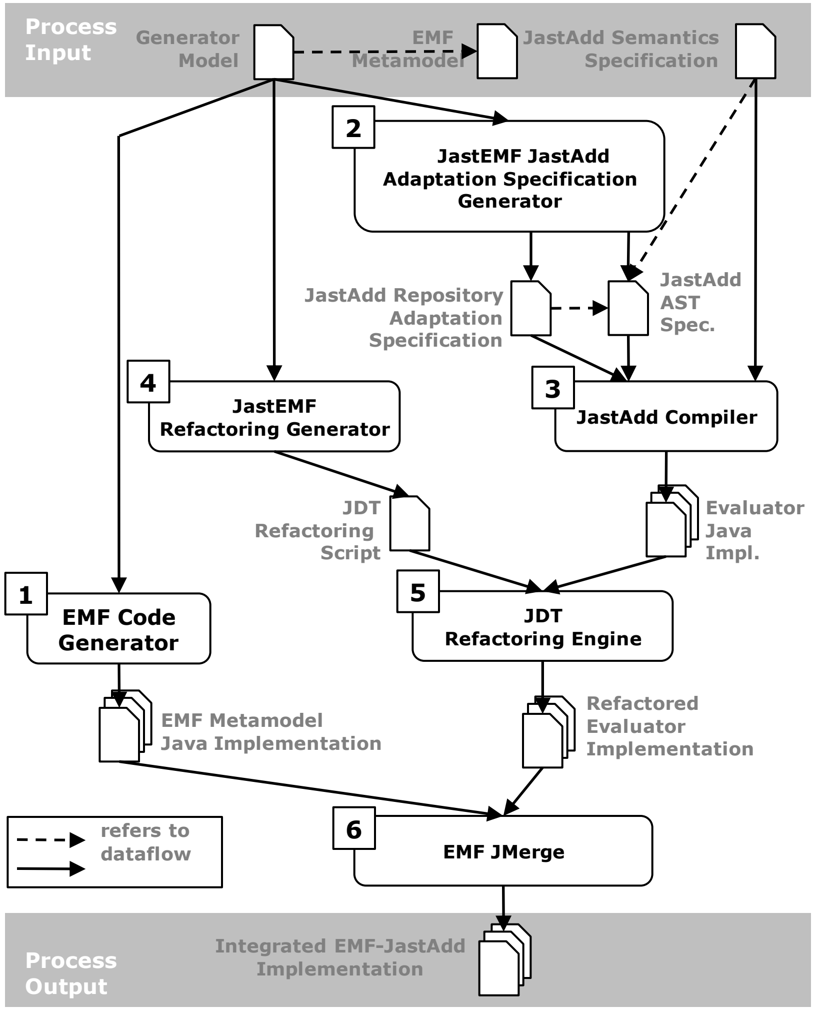


Semantics implementation, but no metamodel implementation

JastEMF's Integration Process

⇒ JastEMF steers EMF & JastAdd

⇒ EMF and JastAdd development can be handled as used to

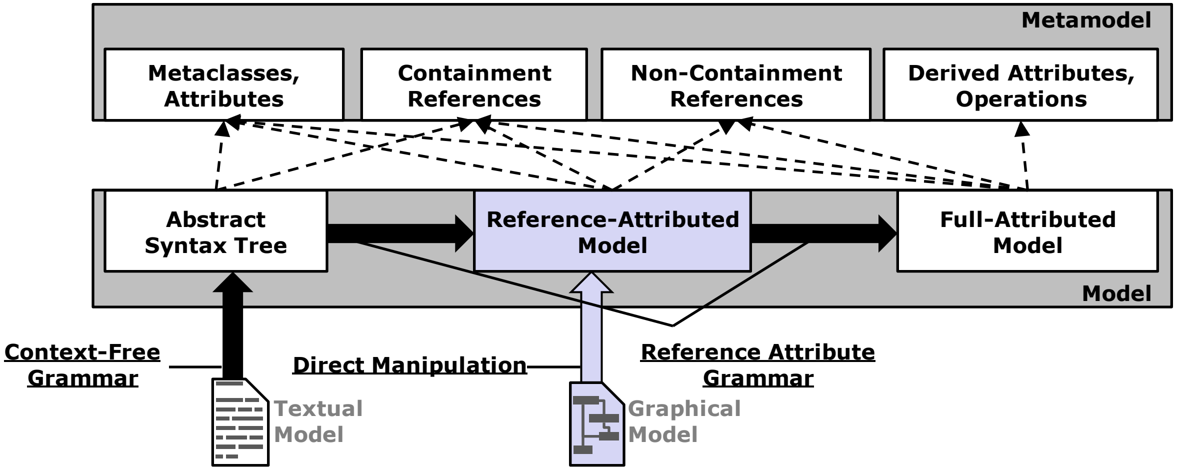

Process Input
Generator Model
EMF Metamodel
JastAdd Semantics Specification
2
JastEMF JastAdd Adaptation Specification Generator
JastAdd Repository Adaptation Specification
JastAdd AST Spec.
4
JastEMF Refactoring Generator
3
JastAdd Compiler
JDT Refactoring Script
Evaluator Java Impl.
1
EMF Code Generator
5
JDT Refactoring Engine
EMF Metamodel Java Implementation
Refactored Evaluator Implementation
refers to
dataflow
6
EMF JMerge
Process Output
Integrated EMF-JastAdd Implementation

04 Remarks and Observations

A few Words about Graphs

Semantic evaluation can start from (partly) reference-attributed models

- Non-containment references can have predefined values (e.g., specified by users using a diagram editor)



- If a value is given: Use it instead of attribute equation

04 Remarks and Observations

“Degenerated” Graphs

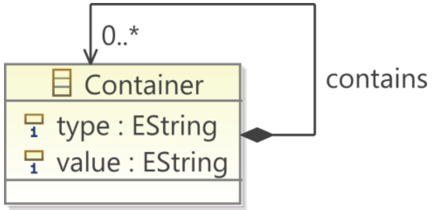
RAGs are only <u>well-suited</u>, if the metamodel does not specify a degenerated tree structure.

Degenerated means:

- Nearly no structure modeled at all
- Models have few structural distinguishable entities and/or flat trees

⇒ Not common in practice (Often a bad modelling indication)

⇒ Similar to model everything just with collections of collections



04 Remarks and Observations

EMF related problems

The EMF does not yet sufficiently consider semantics

- No visualisation support in editors (E.g. error marking)
- No appropriate handling of semantics in the case of syntax errors
  - Possible solution: Reuse attribute dependency graph
- Editors/tools implement and expect default semantics that may differ from a metamodel‘s semantics

05 Conclusion and Outlook

Conclusion

Common metamodelling languages�prime metamodels like Ecore or EMOF specify tree structures enriched with semantic interfaces.

RAGs can be used to specify static semantics for such metamodels.

JastEMF *(www.jastemf.org)***: Tool to generate semantic metamodel implementations based on Ecore metamodels and JastAdd AGs.**

05 Conclusion and Outlook

Outlook

Many JastEMF improvements possible, e.g. :

- Incorporation of incremental AG concepts
- Persistency support for manually changed attribute values
- Incorporation of JastAdd's rewrite capabilities
- Integration based on JDT refactorings is slow
  - A JastAdd EMF backend would lead to an enormeous speed-up

Thank you!

Our sponsors:

SEVENTH FRAMEWORK PROGRAMME

Europa fördert Sachsen.
ESF
Europäischer Sozialfonds

DFG
Deutsche
Forschungsgemeinschaft

SACHSEN

MOSTPROJECT

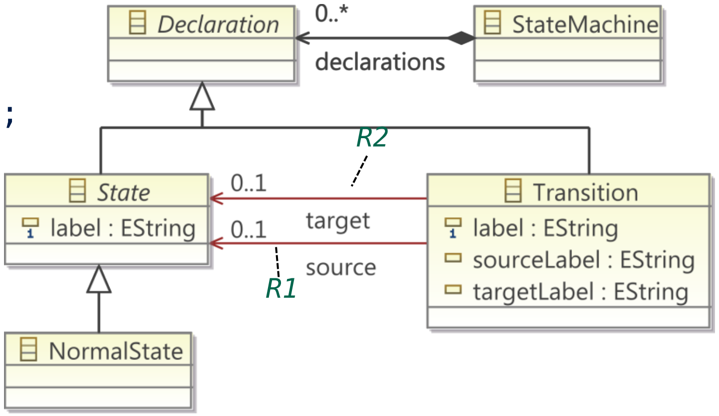
Hyper
adapt

03 The JastEMF Approach

Nameanalysis in Statemachine Example

AST specification (partial):

```
abstract State:Declaration ::= <label:String>;
NormalState:State;
Transition:Declaration ::=<label:String>
    <sourceLabel:String><targetLabel:String>;
```



Attribution example:

```
syn lazy State Transition.source() = lookup(getSourceLabel()); // R1
syn lazy State Transition.target() = lookup(getTargetLabel()); // R2
inh State Declaration.lookup(String label); // R3
eq StateMachine.getDeclarations(int i).lookup(String label) { … } // R4
syn State Declaration.localLookup(String label) =
    (label==getLabel()) ? this : null; // R5
```

(Ecore-based, extended version of Statemachine example in Hedin, G.: Generating Language Tools with JastAdd. In: GTTSE ’09. LNCS ,Springer (2010))