Model-Driven Engineering (MDE)
Lecture 1: Metamodels and Xtext

Regina Hebig, Thorsten Berger

Reuses some material from: Andrzej Wasowski, *Model-Driven Development*, ITU Copenhagen

Where I am from

Universität Potsdam

UNIVERSITAS GOTHOBURGENSIS 1891

UNIVERSITY OF GOTHENBURG

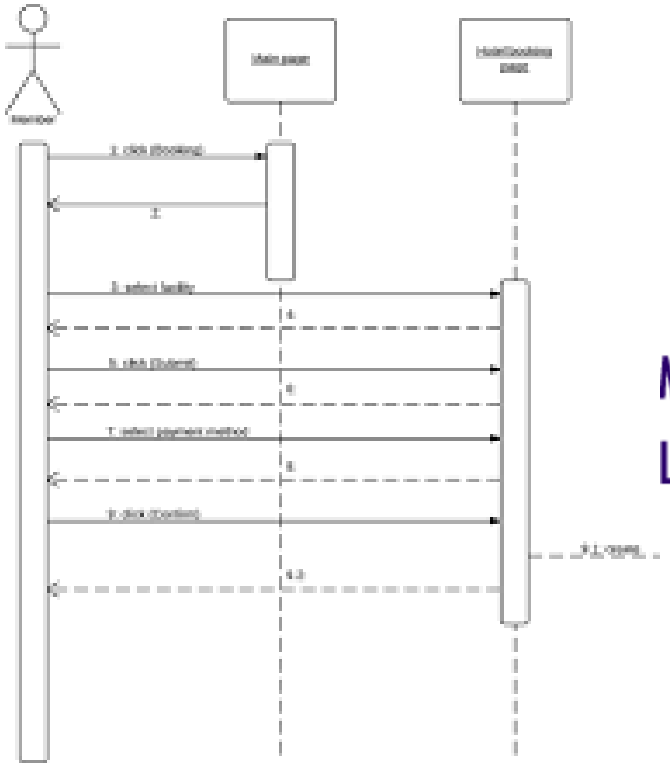
CHALMERS

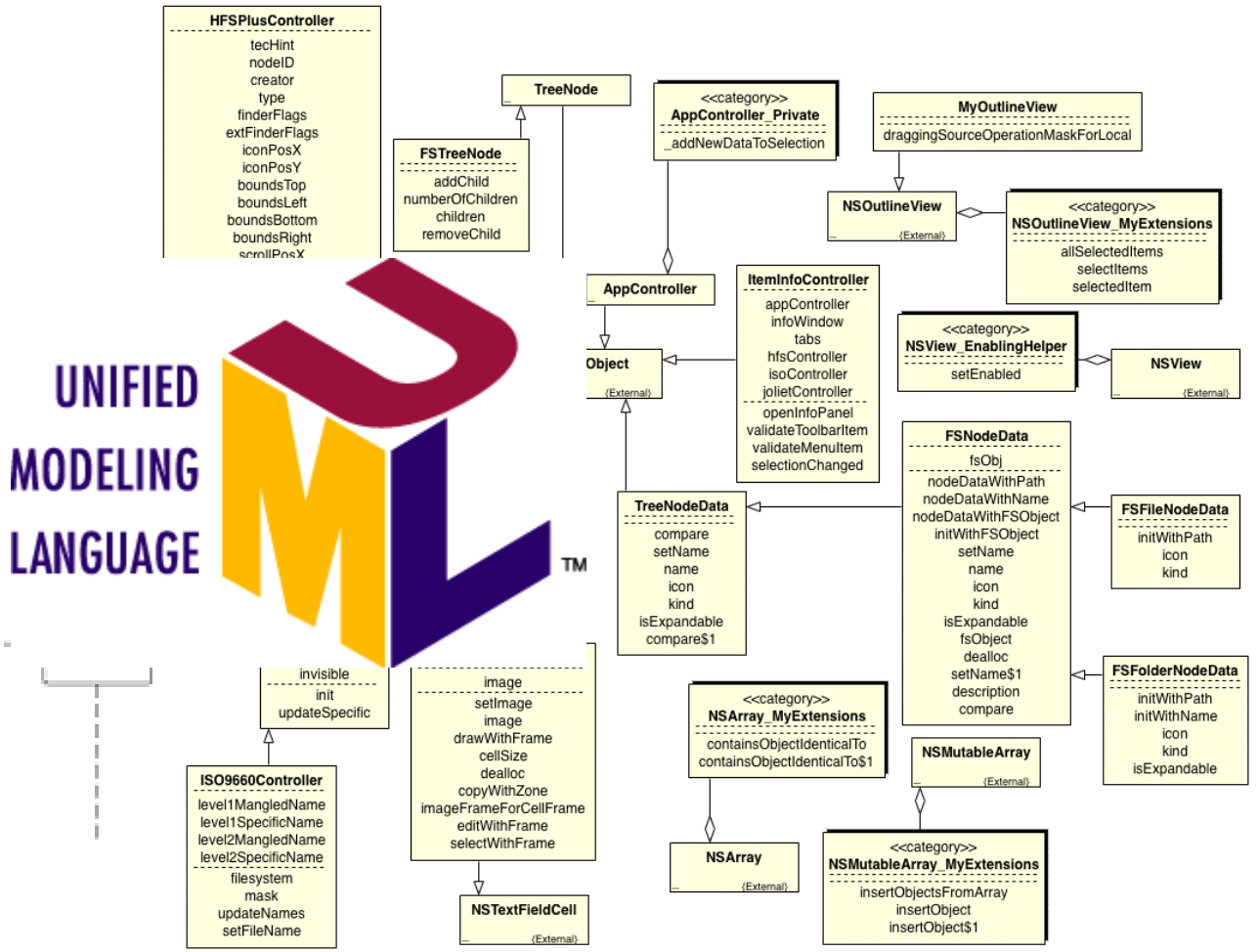
UPMC

SORBONNE UNIVERSITÉS

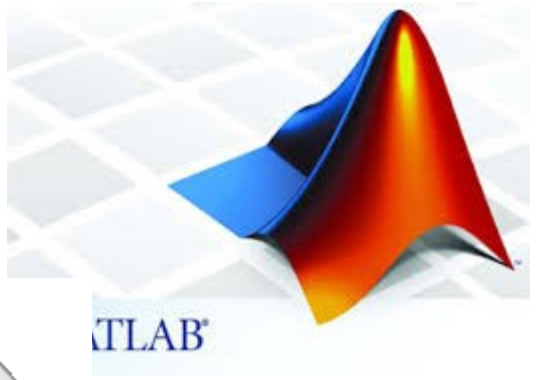
Introduction MDE

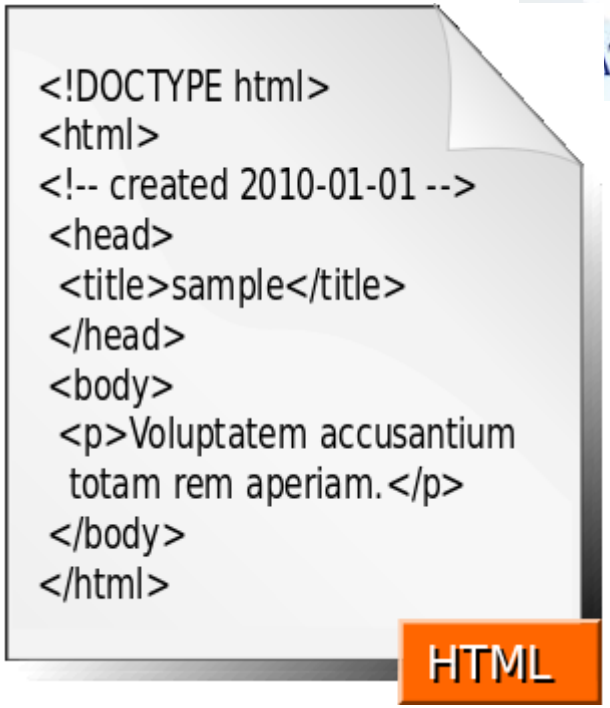
- Models and DSLs





- Models and DSLs





- Models and DSLs

Robotics DSL Zoo | Home | Subdomains | A&P Disciplines | Development Phases | Index | Contribute | DSLRob

An Overview of Domain-Specific Languages in Robotics

hosted by DSLRob, initiated and maintained by A. Nordmann, N. Hochgeschwender , D. Wigand, and S. Wrede, updated on August 9th 2016

This is an index of all publications in the Robotics DSL Zoo. For a more structured overview have a look at the different Subdomains.

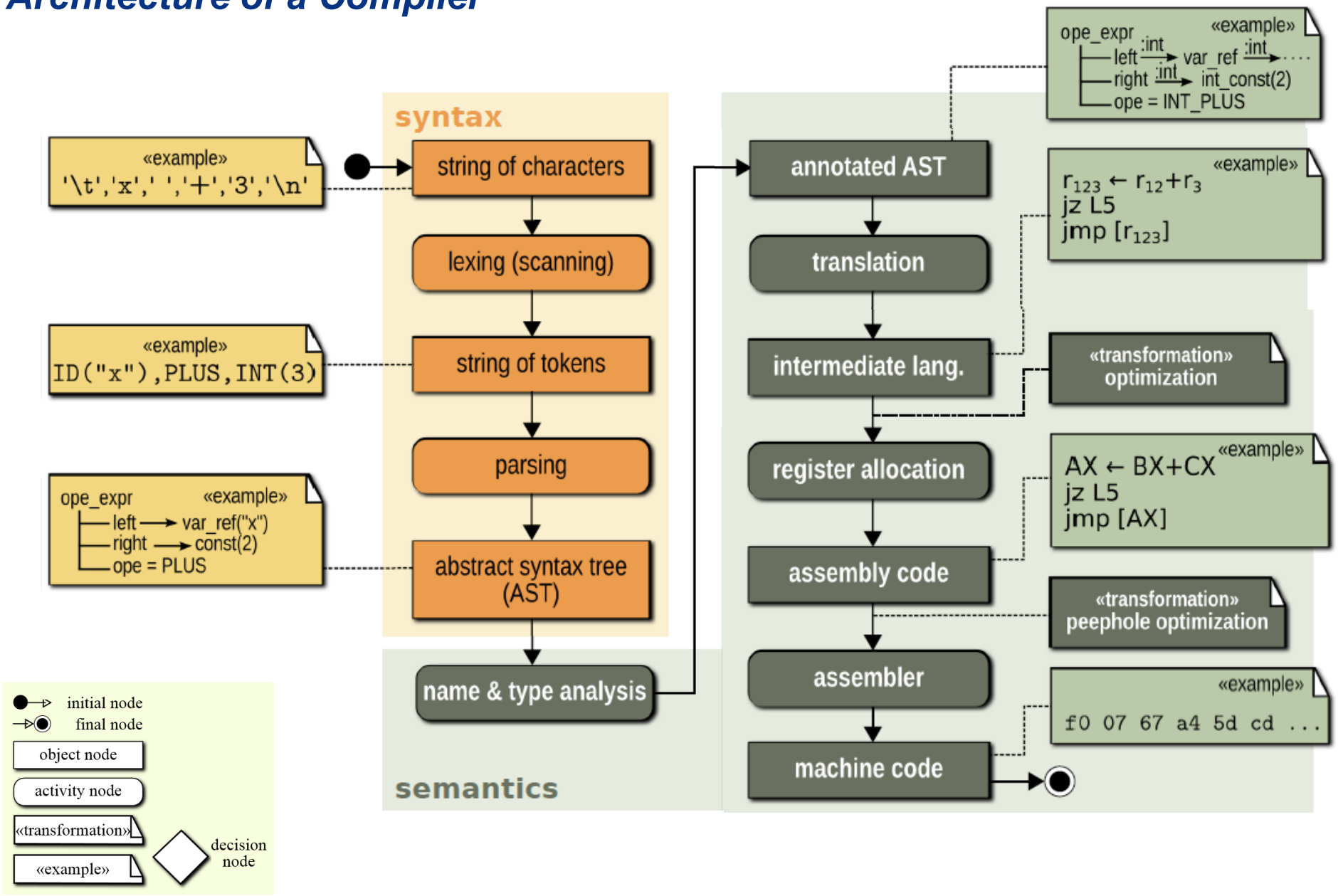
Index

Key	Title	Authors	Year	Subdomains	Phases
abdellatif2012rigorous	**Rigorous design of robot software: A formal component-based approach**	Abdellatif, Tesnim and Bensalem, Saddek and Combaz, Jacques and De Silva, Lavindra and Ingrand, Felix	2012	Architectures and Programming,	Capability Building, System Benchmarking,
adam2014towards	**Towards rule-based dynamic safety monitoring for mobile robots**	Adam, Sorin and Larsen, Morten and Jensen, Kjeld and Schultz, Ulrik Pagh	2014	Architectures and Programming,	Functional Design, Capability Building, Product Maintenance,
aertbelien2014etasl	**eTaSL/eTC: A constraint-based task specification language and robot controller using expression graphs**	Aertbeli\ en, Erwin and De Schutter, Joris	2014	Architectures and Programming, Kinematics, Dynamics,	Capability Building,
aggarwal1994specification	**Specification and automated implementation of coordination protocols in distributed controls for flexible manufacturing cells**	Aggarwal, Sudhir and Mitra, Sandeep and Jagdale, Sanjay S	1994	Architectures and Programming,	Capability Building,
Akim2010	**Events! (Reactivity in Urbiscript)**	Akim, Jean-christophe Baillie and Quentin, Demaille and Matthieu, Hocquet and Gostai, S A S and Berlier, Jean-baptiste and Paris, F-	2010	components, Architectures and Programming,	
alonso2010v3cmm	**V3cmm: A 3-view component meta-model for model-driven robotic software development**	Alonso, Diego and Vicente-Chicote, Cristina and Ortiz, Francisco and Pastor, Juan and Alvarez, Barbara	2010	Architectures and Programming,	Capability Building, System Deployment,
Anderson2011	**Work in Progress: Enabling Robot Device Discovery through Robot Device Descriptions**	Anderson, Monica and Kilgo, Paul and Crawford, Chris and Stanforth, Megan	2011	Architectures and	

http://corlab.github.io/dslzoo/all.html

How to build a language?

Architecture of a Compiler



Language = Syntax and …
→ Concrete and Abstract Syntax

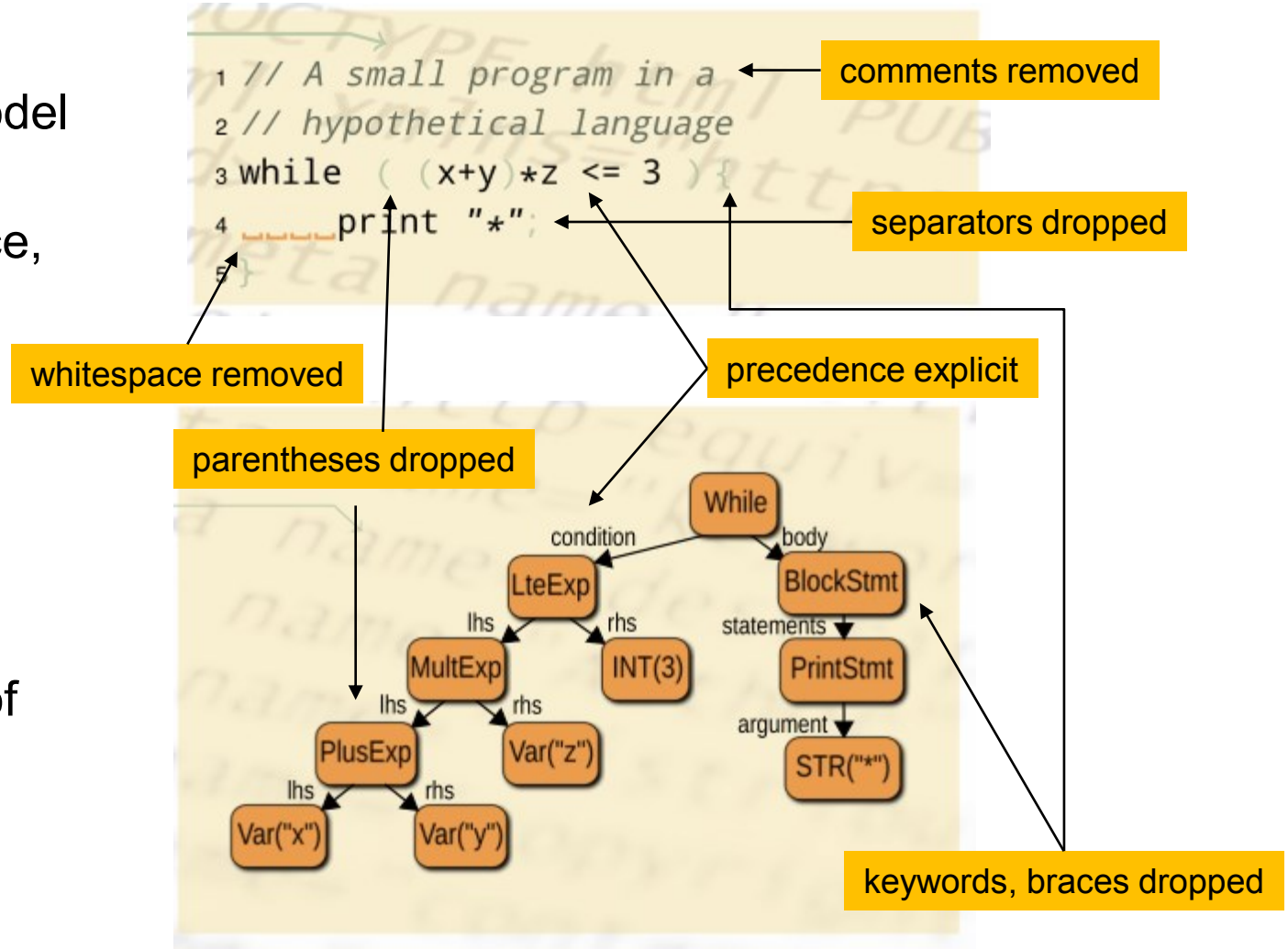
Concrete Syntax

Representation of a model instance/program as

- Text, with whitespace, parentheses, curly braces, …
- Graphs in an editor

Abstract Syntax

Elements and their relations independent of the representation



Language = Syntax and Semantics

Concrete Syntax

Representation of a model instance/program as

- Text, with whitespace, parentheses, curly braces, …
- Graphs in an editor

Abstract Syntax

Elements and their relations independent of the representation

Static Semantics

semantics evaluatable without executing/ interpreting the model (using constraints), e.g.

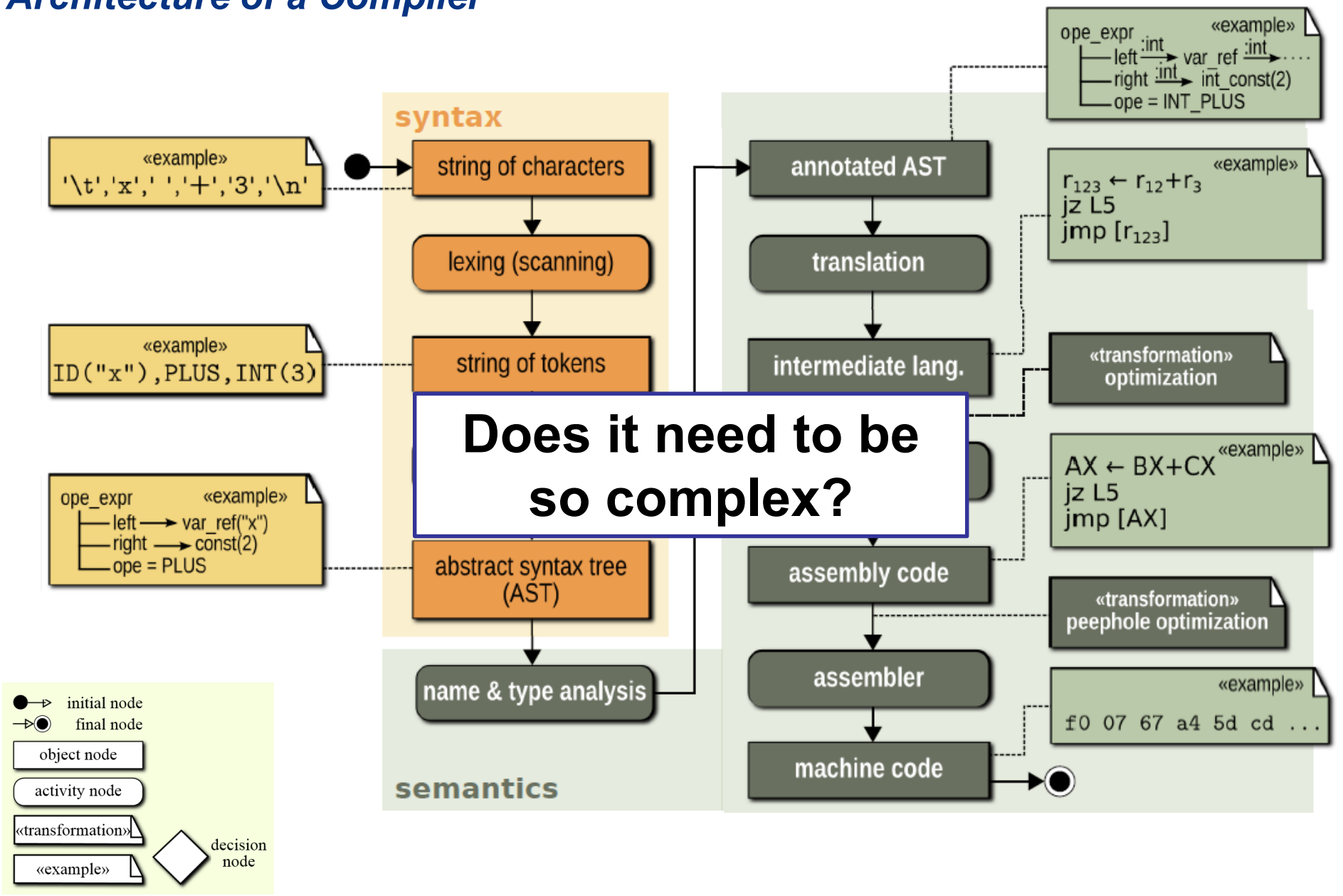
- compile-time correctness of a program: are variables declared? is it well-typed?

Dynamic Semantics

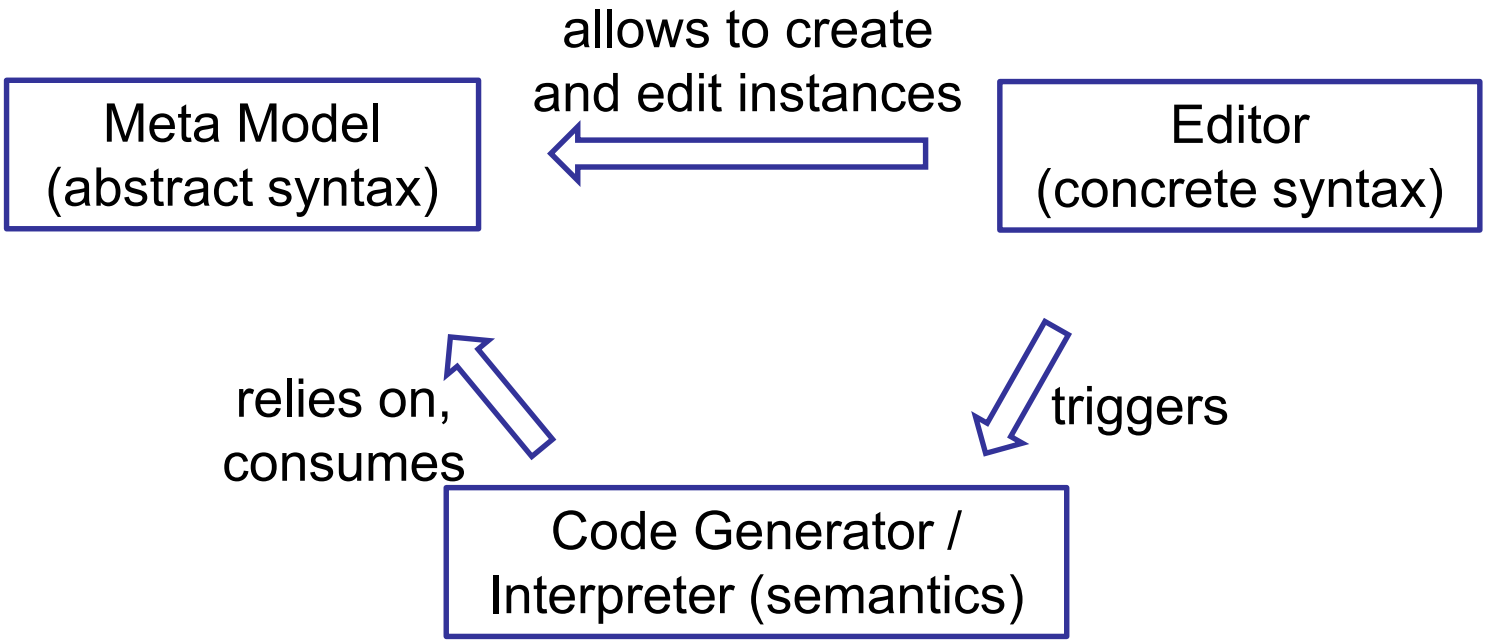
the meaning or effect of a model/program at run-time; what happens when it is executed?; captured by generated code or interpreter

Language

Architecture of a Compiler



Building a Language with Model-Driven Techniques

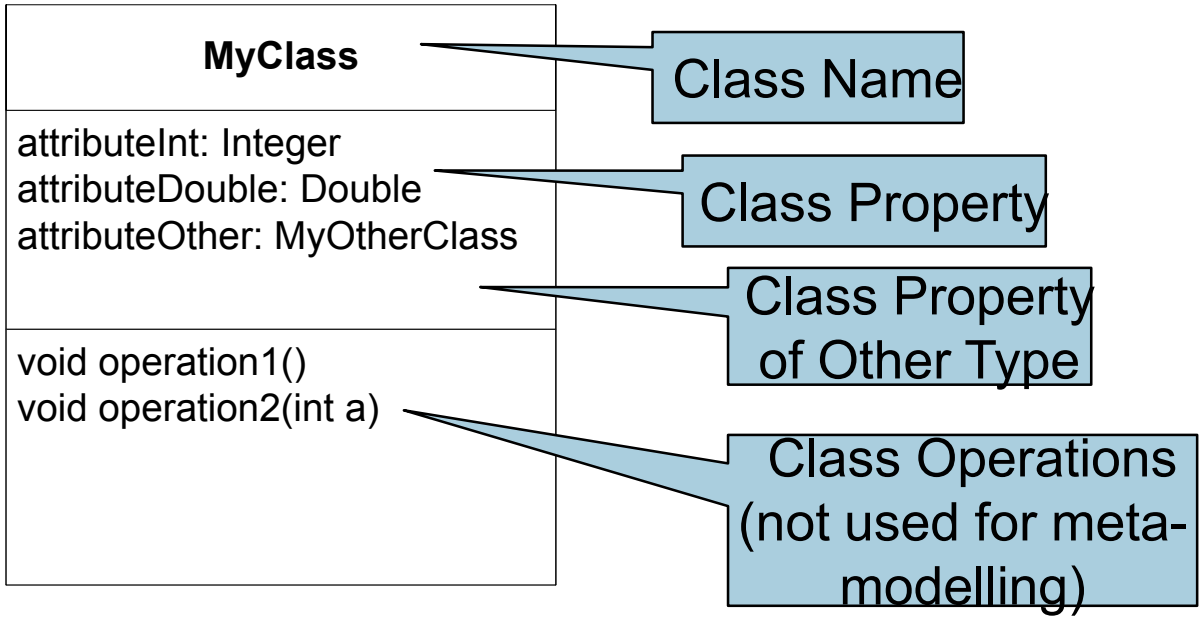


Abstract Syntax: Metamodeling

Before we Start: Reminder Class Modeling

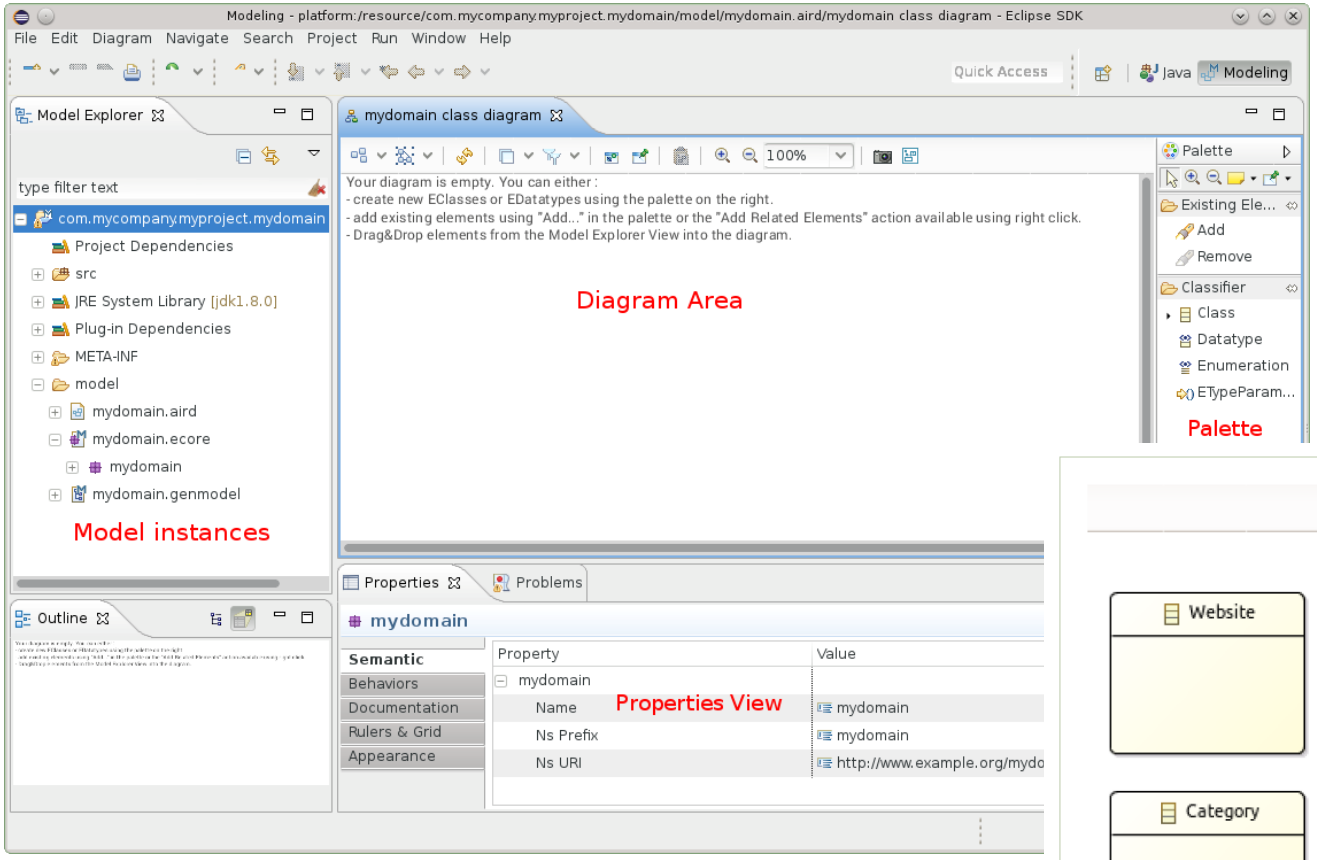
- UML Class Diagram are used to meta-models (abstract syntax of a DSL)
- Allow structural and conceptual modeling

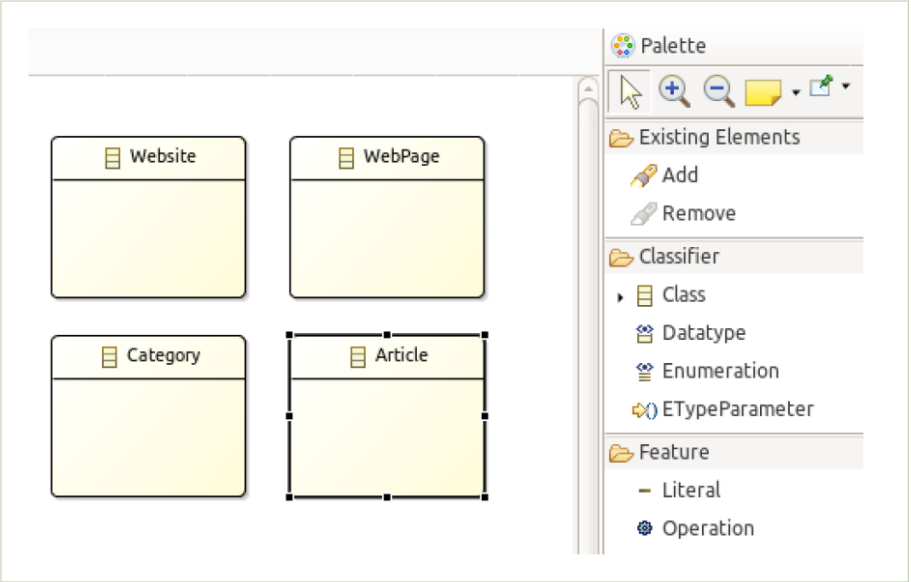
Definition: A class is an abstraction that specifies attributes of a set of concept instances (objects) and their relations to other concepts (classes).



Before we Start: Reminder Class Modeling Creating Class (Ecore) Diagrams with EMF

- Good tutorial: https://www.eclipse.org/ecoretools/



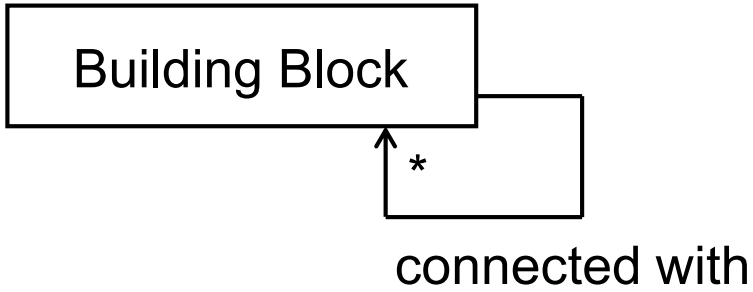


Meta-Modeling

- Meta-modeling is *modeling modeling languages*
  - The Greek prefix "meta" means "about" - meta-model states something "about" other models
  - EMF is the most popular tool in this space
  - Tools can generate editors, serializers, and data structures from meta-models
- Why use meta-models in language definitions?
  - Concise and precise definition of the language concepts
  - Standardized exchange format
  - Checking correctness of models
  - Management of models in repositories

Meta-Model: Lego

Meta-Model
Building Rules



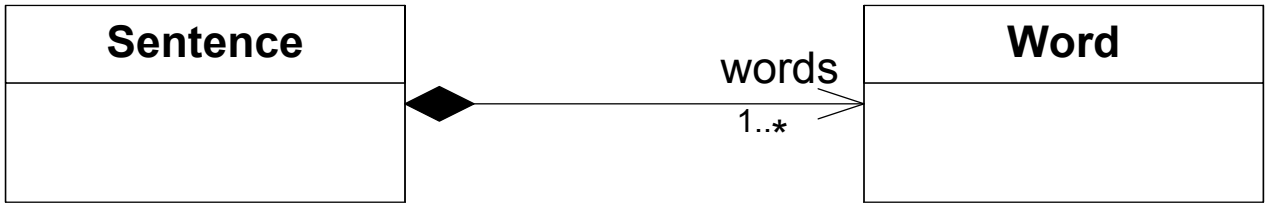
Model
Lego

Real world
House

Images: pixelquelle.de

Meta-Model: Languages

Meta-Model
Building Rules



Model
A natural description

"A nice brown and white coloured house in the middle of the black forest"

Real world
House

Meta-Model: Definition

A meta-model is a model that precisely defines the parts and rules needed to create valid models.

Parts: domain concepts (model elements)

Rules: Well-formedness rules, determine validity of a model.

→ **Metamodel defines the abstract syntax of a language**

Elements and their relations independent of the representation

Abstract vs. Concrete Syntax (1)

Abstract Syntax

Music notation

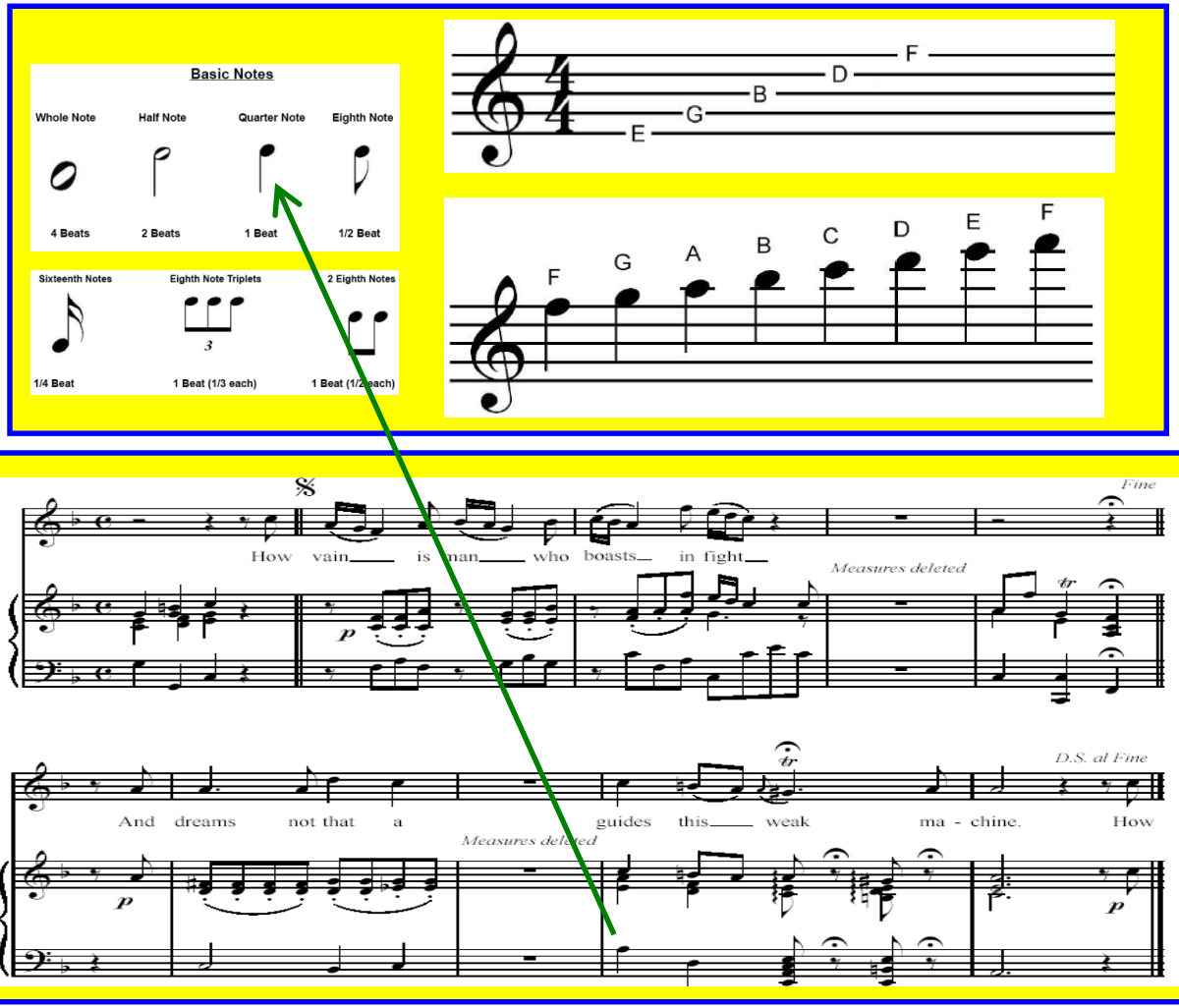
Meta-Model

conforms to

Model

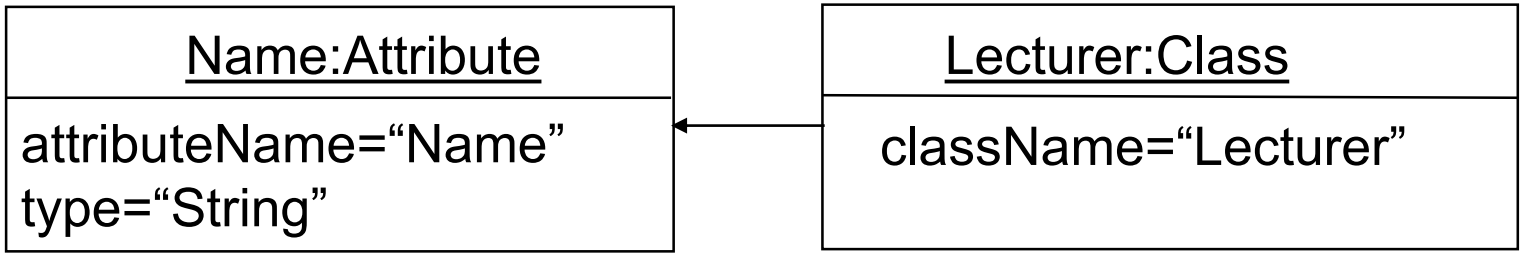
Music sheet

Concrete (Graphical) Syntax

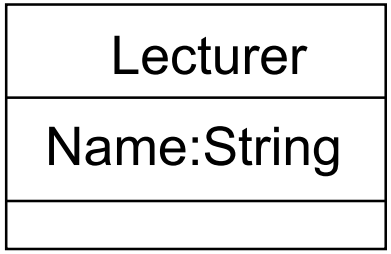


Abstract vs. Concrete Syntax (2)

Abstract Syntax



Concrete Syntax




```
Class Lecturer {
    String Name;
}
```

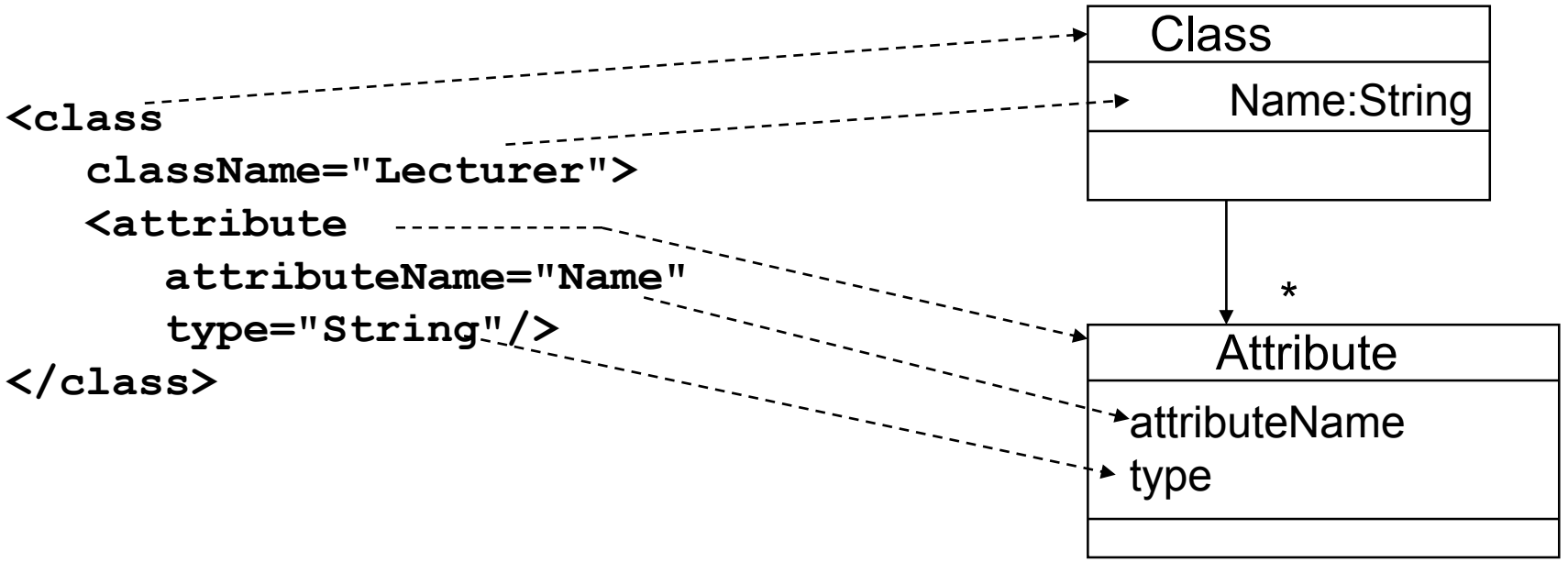
```
modelclass Lecturer
(
    attributes
    (
        Name : Type String
    )
)
```

```
<class
    className="Lecturer">
    <attribute
        attributeName="Name"
        type="String"/>
</class>
```

Compare to the AST concept in compiler construction!

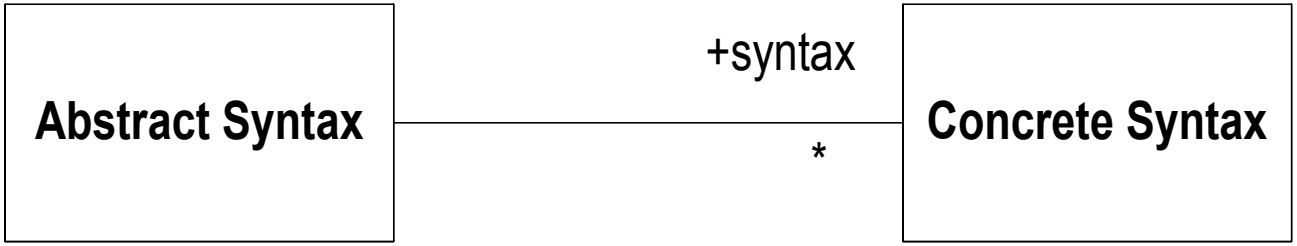
Abstract vs. Concrete Syntax (4)

- A mapping maps elements of the concrete syntax to the meta-model elements



There is a standard that states the mapping from (MOF) models to XML: XMI (XML Metadata Interchange)

Abstract vs. Concrete Syntax (3)



- There can be many concrete syntaxes for an abstract syntax
- Different types
  - Textual
  - Graphical
  - …

Metamodel (15 minutes)

Have a look at the 2 metamodels in the handout. Both metamodels stem from students who participated in the Spring 2017 MDE course at Chalmers. The metamodels function as basis for two DSLs that allow the definition of custom made Lego bricks.

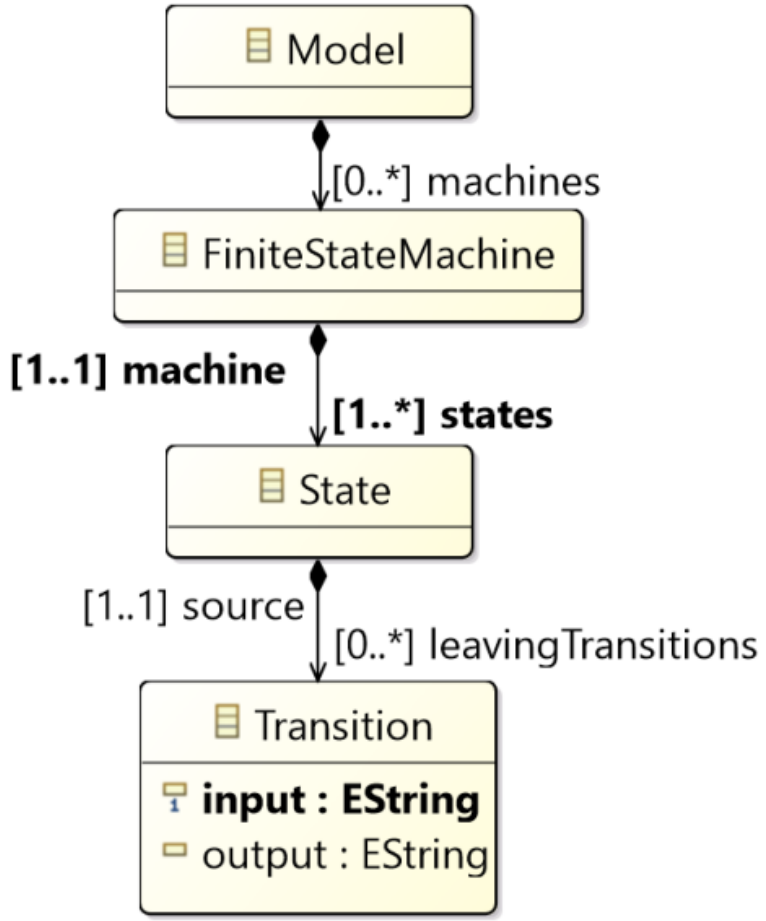
Discuss the models with your neighbors (2-3 persons) and try to answer the following questions:

1. Have a look at differences and commonalities between the 2 metamodels. Consider that the metamodels are the basis of two DSLs. Try to name
   - 2-3 features that you expect work in both DSLs.
   - At least 1 feature that you expect to work in only one of the DSLs
2. When having a look at the metamodels, can you identify a common design principle? (Hint: have a look at something that holds for most classes, direct or by inheritance)

Design Guidelines

Design Guidelines

- In EMF all objects (and thus all classes) must be directly or indirectly owned by a single root class
- Meta-models should have a single partonomy



Design Guidelines

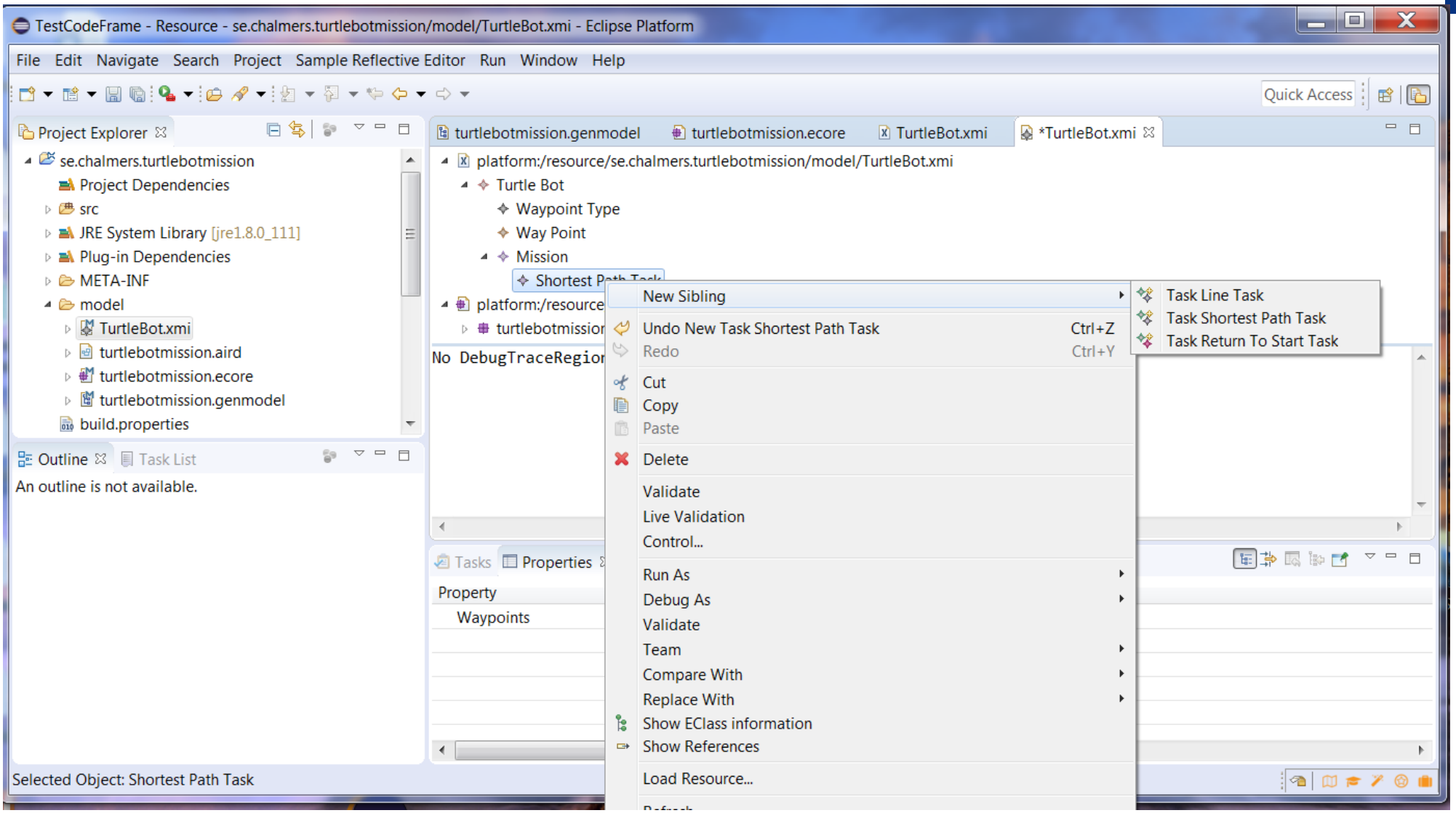
- It is a bad smell if you see interfaces or methods in your meta-model
- It is useful to verify the taxonomy (generalization hierarchy) of the meta-model with domain experts
- If relations between concepts have properties, you can reify them as classes
- It is a bad smell if multiple classes have the same property
  - Recall the abstract class NamedElement

Wrap-up Metamodels

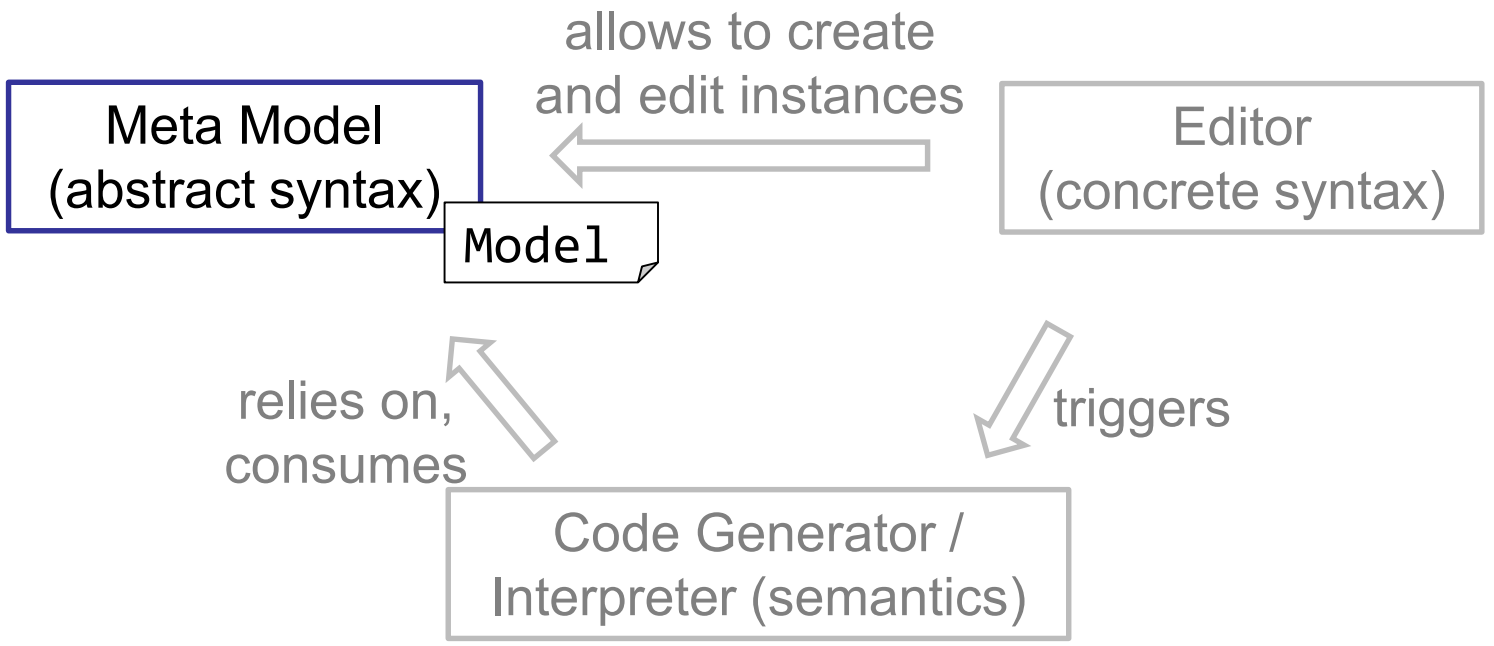
Wrap-up Metamodels

- Class modeling
  - Classes represent domain concepts
  - Class modeling is a means for meta-modeling
- Meta-modeling
  - Meta-models model models, i.e., meta-models are languages in which models are expressed
  - Describe the abstract syntax; concrete syntaxes associated
- Meta-levels
  - Meta-Modeling languages, such as Ecore, can be used to model meta-models (i.e., languages)
  - Both use the concept of class modeling

Editors Generated from Metamodel

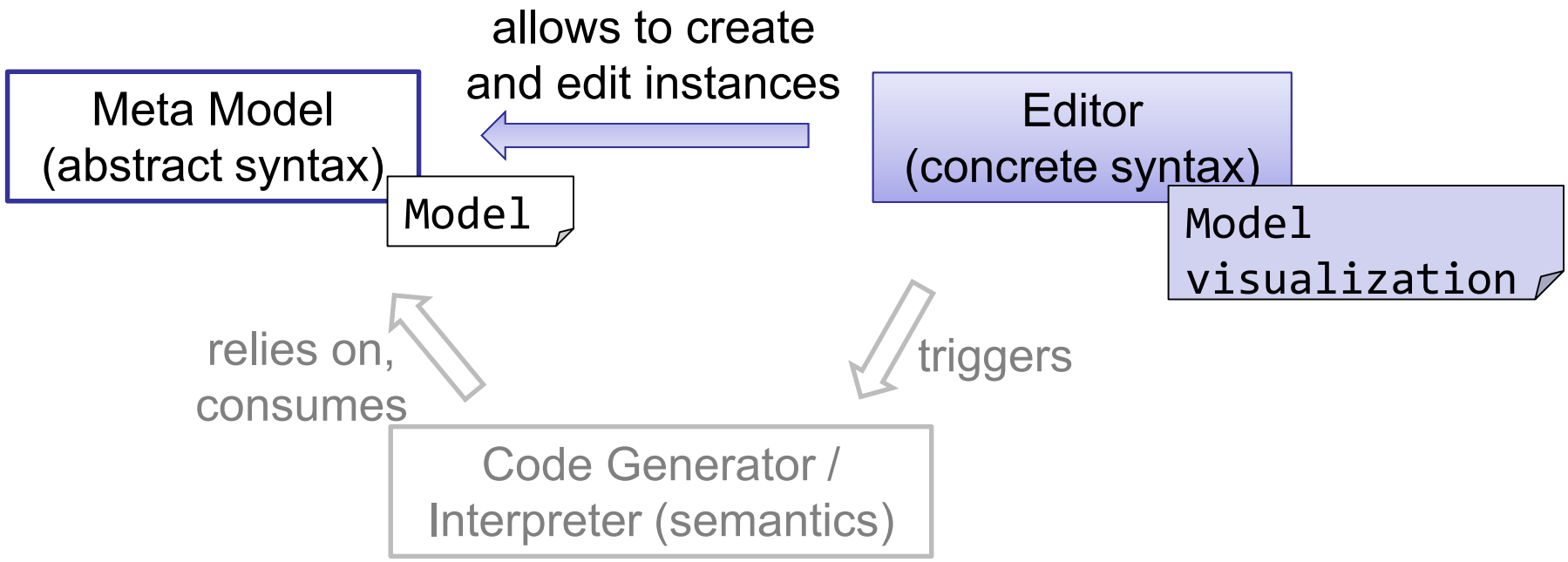


Wrap-up Metamodels



Concrete Syntax

Concrete Syntax



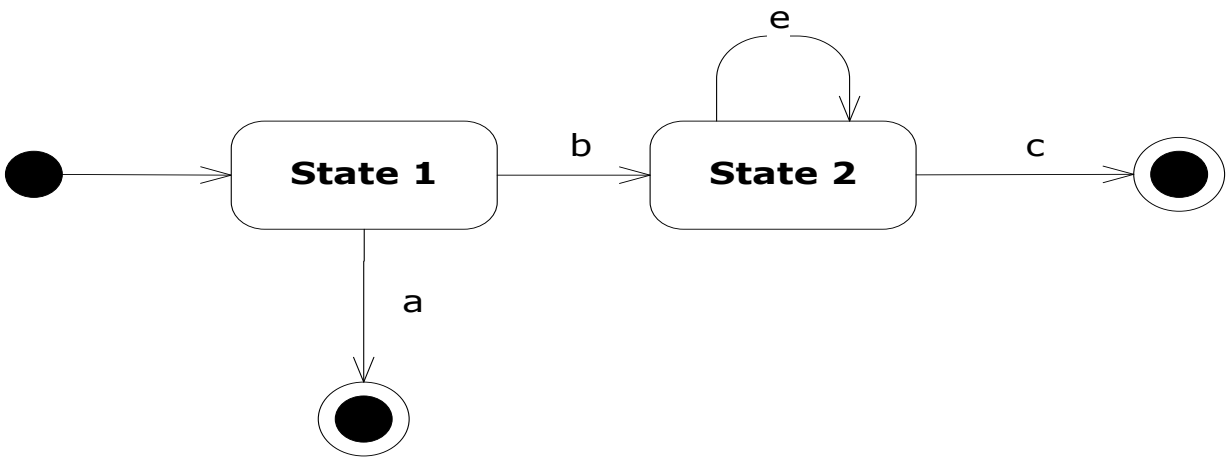
Textual vs. Graphical Syntax

FSM Example:

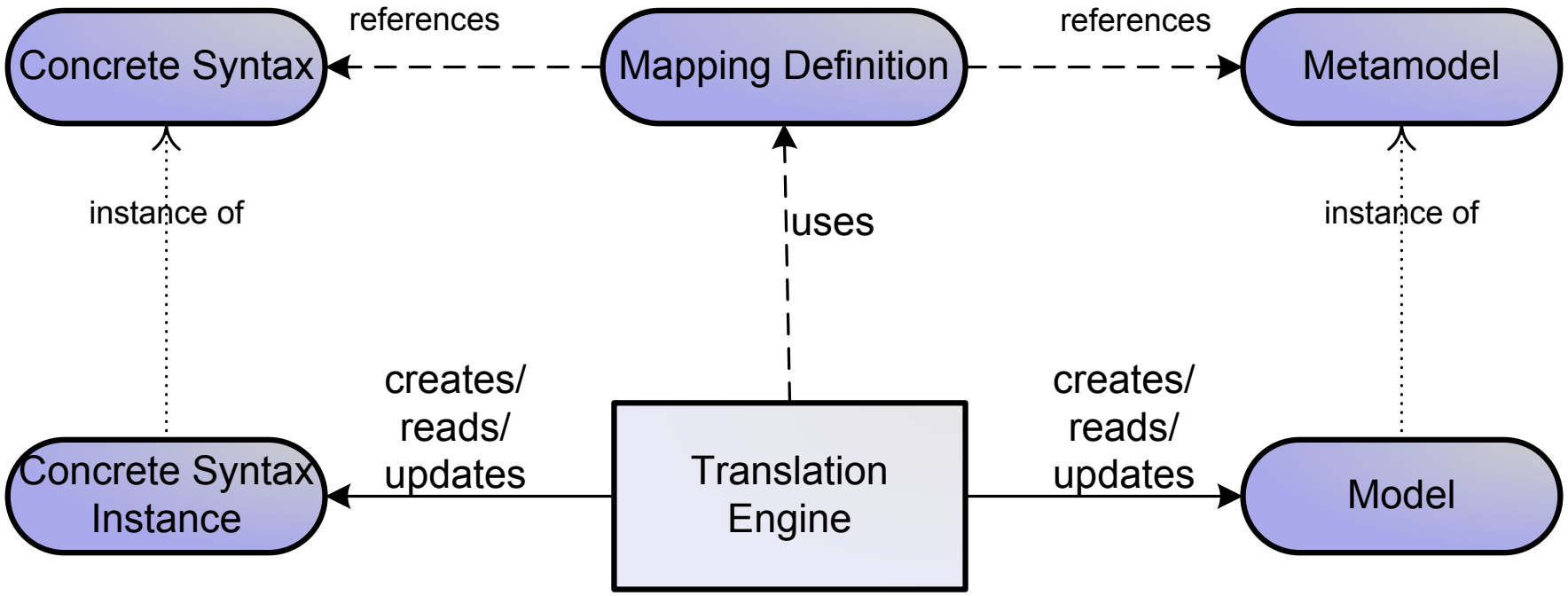
Textual

```
STATES
   State 1, State 2, Start(start), Stop 1(stop), Stop 2(stop)
TRANSITIONS
   Start->State 1, State 1 -b-> State 2, State 2 -e-> State 2,
   State 2 -c-> Stop 1, State 1 -a-> Stop 2
```

Graphical



Creating a Concrete Syntax

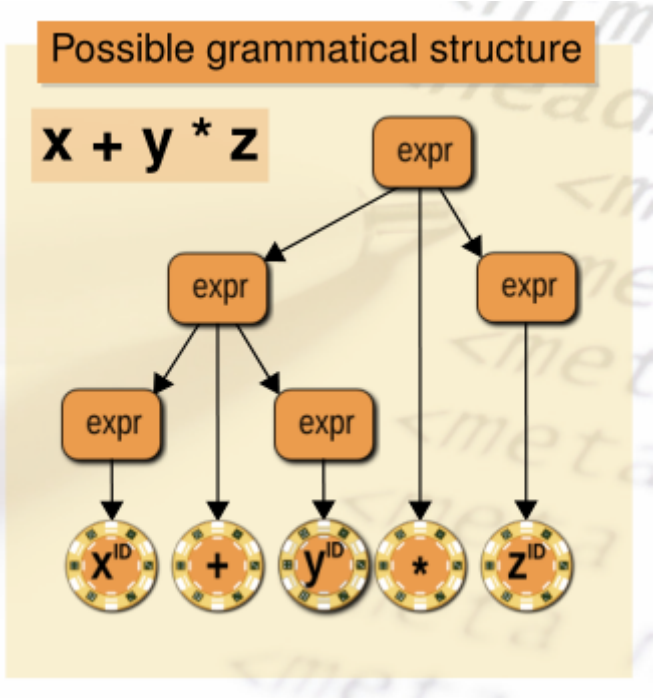


- Generative approaches available for …
  - graphical (e.g., Eclipse GMF)
  - and textual external DSLs (e.g., Xtext)

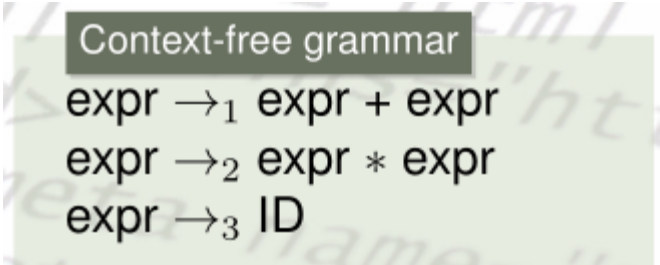
Basics: Grammar

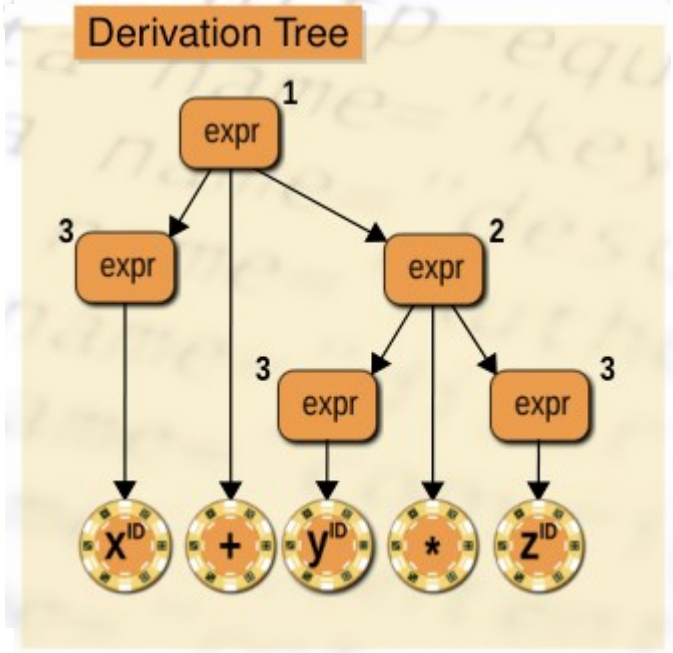
Syntax Trees

- How are tokens organized into phrases?

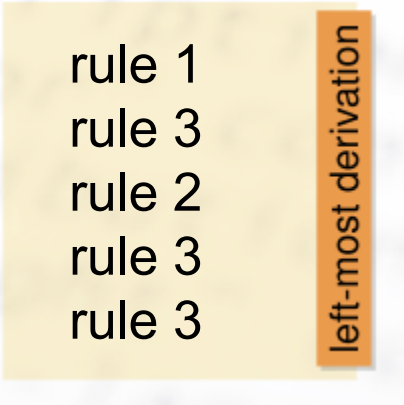


How do we specify **all legal syntax trees** like this one?





The grammar is ambiguous!



Context-Free Grammars

Context-Free Grammar

- Set of productions over disjoint sets of terminal symbols (tokens) and nonterminal symbols
- Production rewrites a nonterminal (left) to a list of terminals and nonterminals (right)
- Any nonterminal used on the right of a production appears on the left of some production
- One nonterminal is a start symbol

Context-Free Language

- Let α, β and γ be sequences of symbols
- A derivation relation: $\alpha N \beta \Rightarrow \alpha \gamma \beta$ iff $N \rightarrow \gamma$
- A CFG G over terminals T defines a context-free language over T $\{w \in T* \mid S \Rightarrow* w\}$.
- Each regular language is context-free

Extended Backus-Naur Form (EBNF): Syntactic Sugar

alternative: $S \longrightarrow \alpha|\beta \longrightarrow \begin{cases} S \longrightarrow \alpha \\ S \longrightarrow \beta \end{cases}$

optional: $S \longrightarrow \alpha\ T?\ \beta \longrightarrow \begin{cases} S \longrightarrow \alpha\ T'\beta \\ T' \longrightarrow T|\varepsilon \end{cases}$

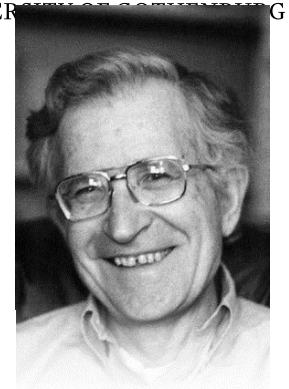
Kleene$*$: $S \longrightarrow \alpha\ T\ *\ \beta \longrightarrow \begin{cases} S \longrightarrow \alpha\ T'\beta \\ T' \longrightarrow (TT')? \end{cases}$

grouping: $S \longrightarrow \alpha(\beta)\gamma \longrightarrow \begin{cases} S \longrightarrow \alpha\ T'\gamma \\ T' \longrightarrow \beta \end{cases}$

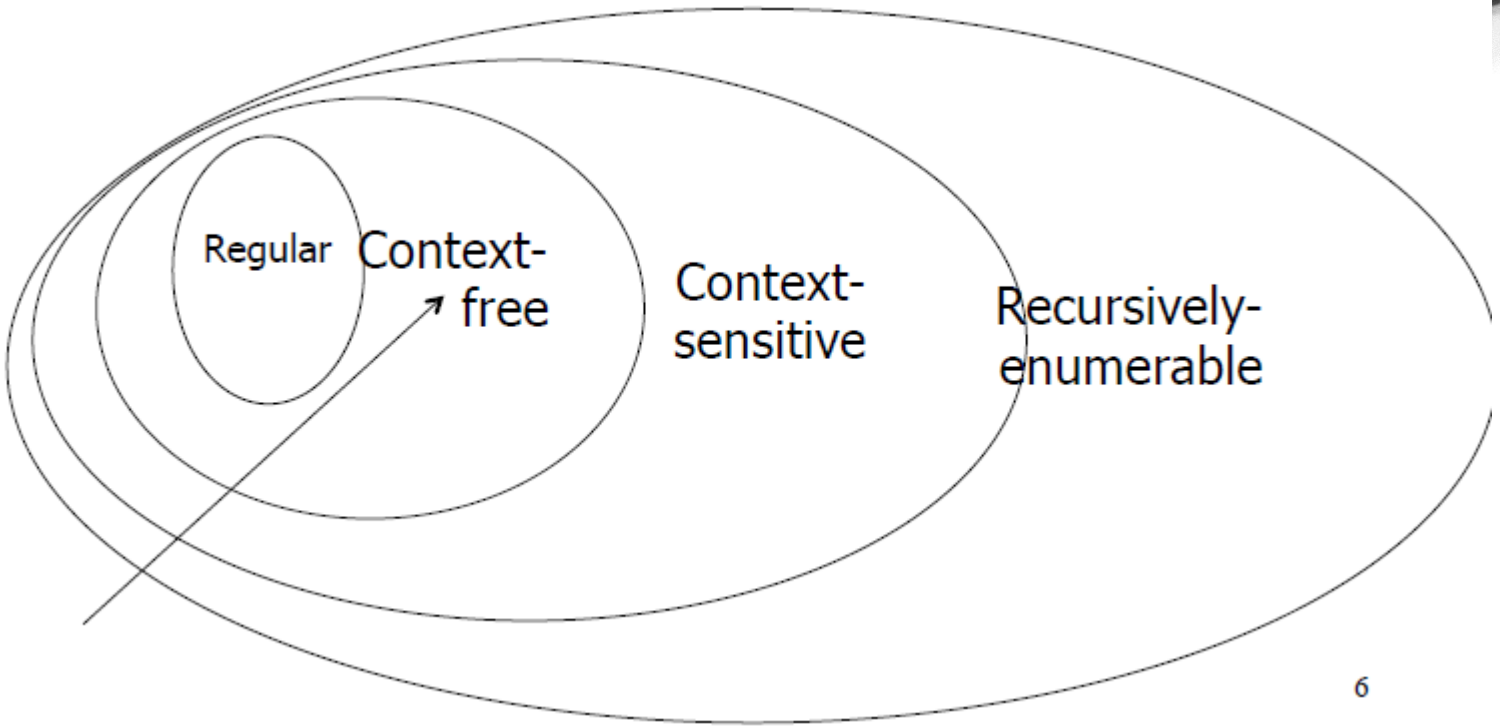
Example

op → "+"|" * "
exp → exp op exp | ID | "("exp")"

The Chomsky Hierarchy



Noam Chomsky

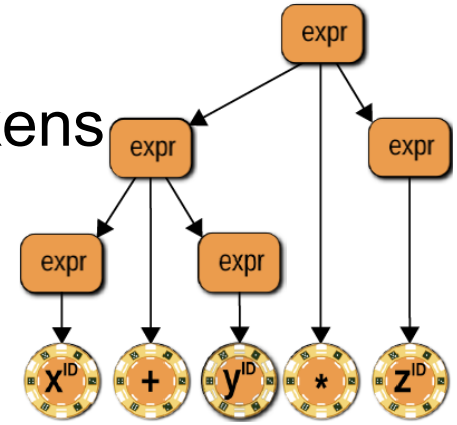


N. Chomsky. Three models for the description of language. IEEE Trans. on Information Theory 2(3):113–124. 1956

LL-Parsing

- Parser: component translating a sequence of tokens to a data structure representing a parse tree

parsing



Recall our example grammar

expr $\rightarrow_1$ expr "+" expr
expr $\rightarrow_2$ expr "$*$" expr
expr $\rightarrow_3$ ID

Ambiguity (conflict)

Derive from a start symbol expr:

x + y $*$ z $\longrightarrow$ use rule 1?

x + y $*$ z $\longrightarrow$ or rule 2?

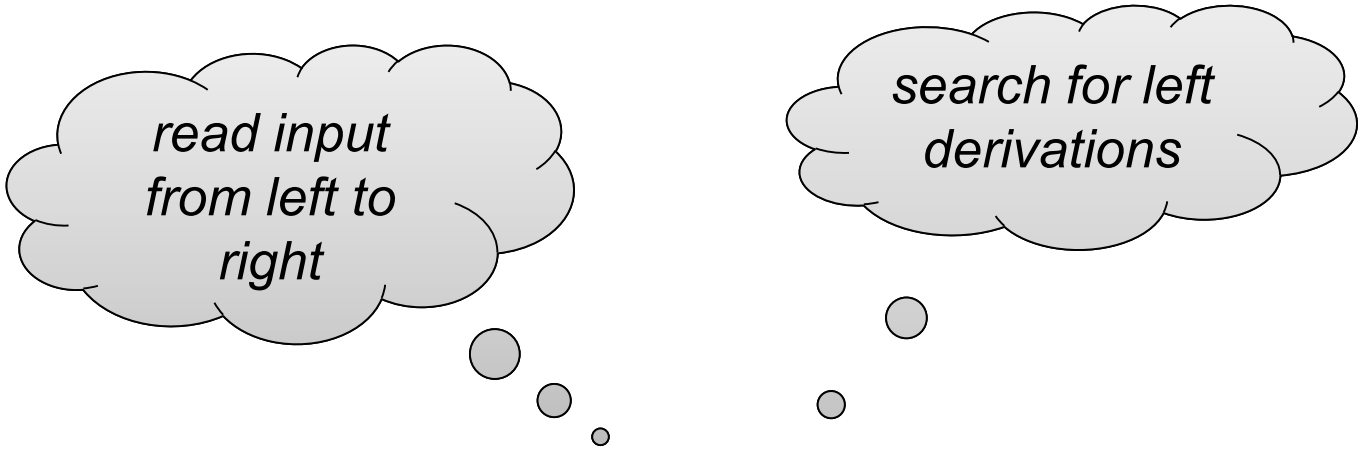
Not an ANTLR error!

LL(k) grammar

A for which always the next production in a left derivation can be selected deterministically (unambiguously) based on reading next k tokens.

LL($*$) parser

Selects productions by recognizing if following tokens belong to regular language (no limit on prefix length).



LL-Parsing

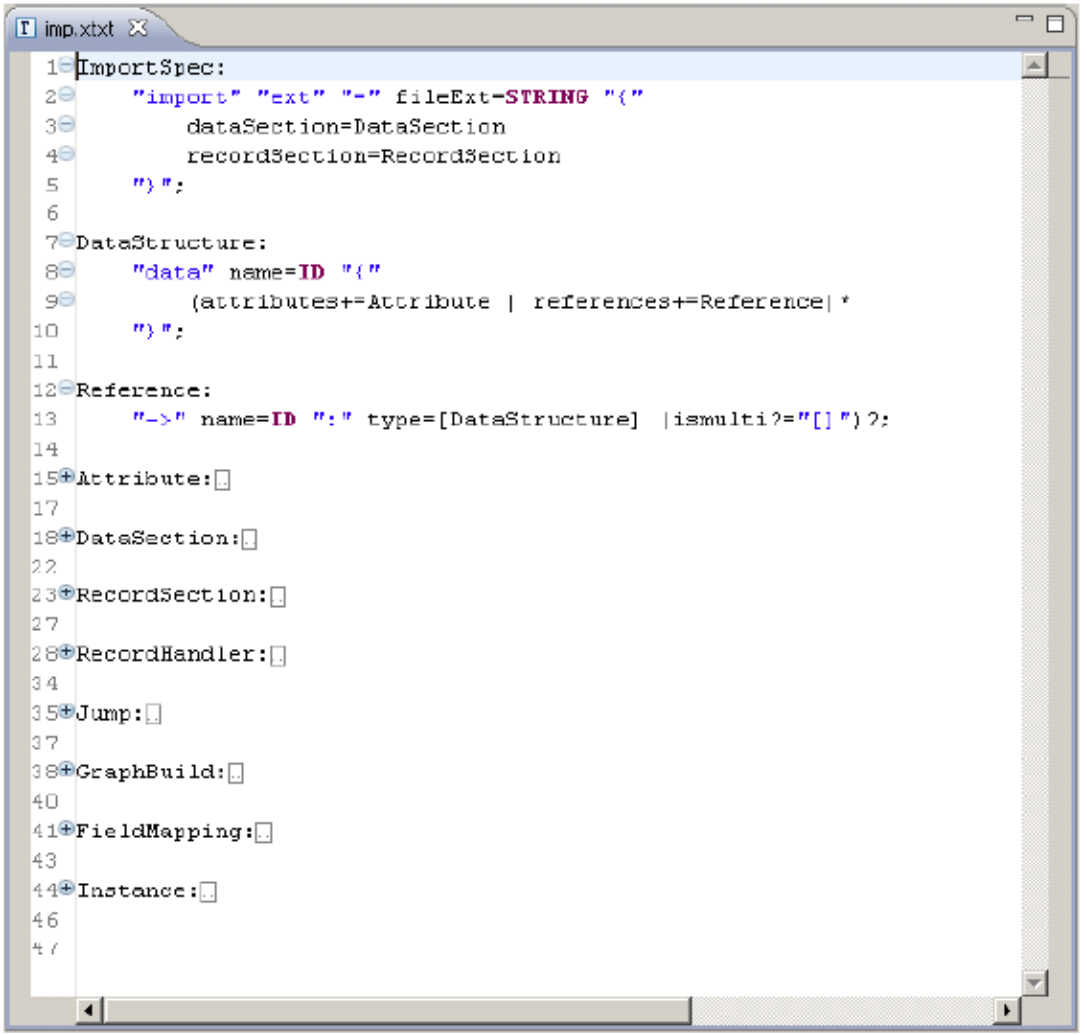
Torben Ægidius Mogensen. Introduction to Compiler Design.
Springer-Verlag London Limited. 2011 <- has a guide on writing and disambiguating grammars

XText

For External DSLs

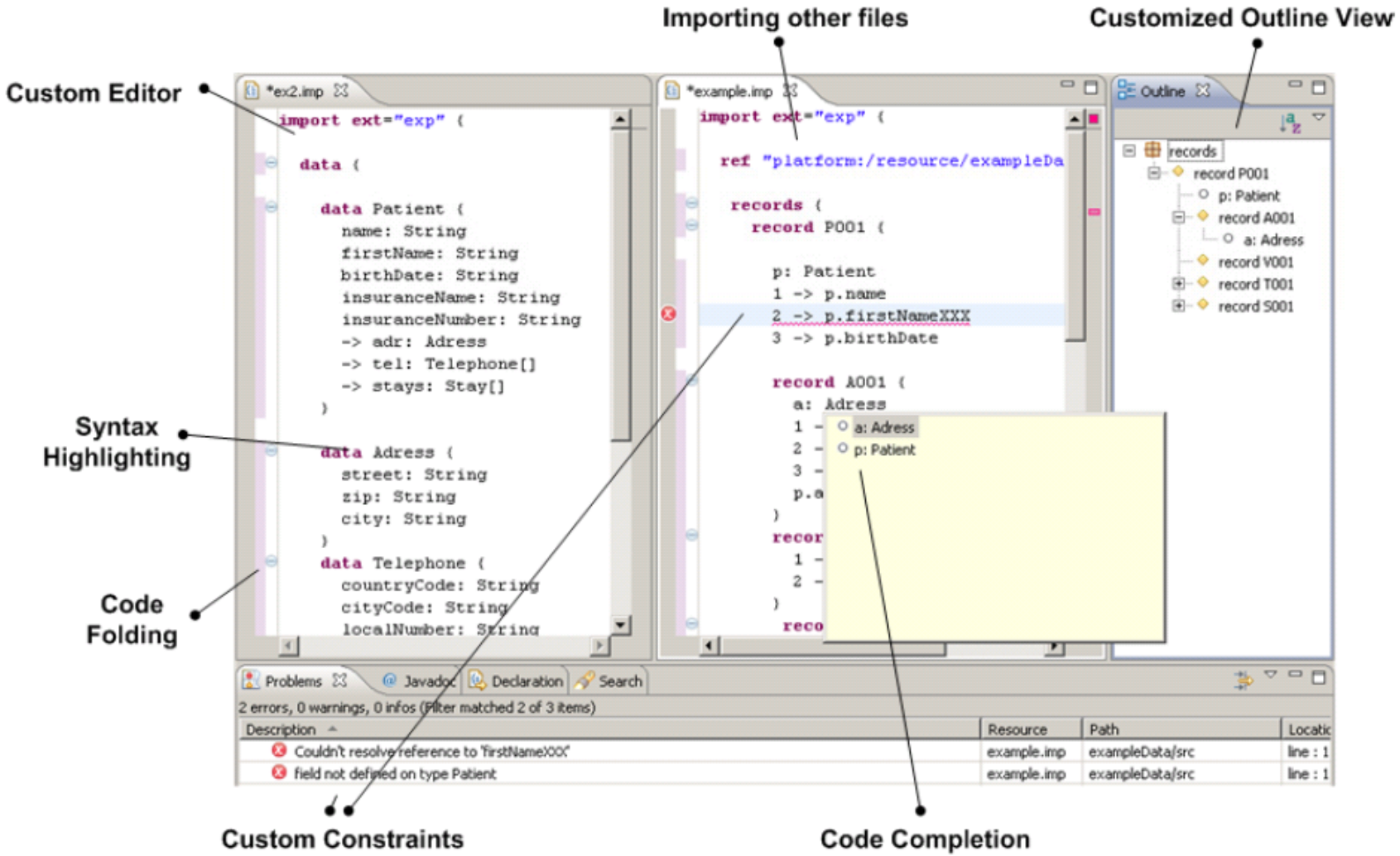
xText

- Grammar is defined in an **EBNF-like format** in the **xText editor**
- The editor provides **code completion and constraint checking** for the **grammars** themselves
- Grammar is a collection of **Rules.** Each Rule is responsible for one **metamodel element**



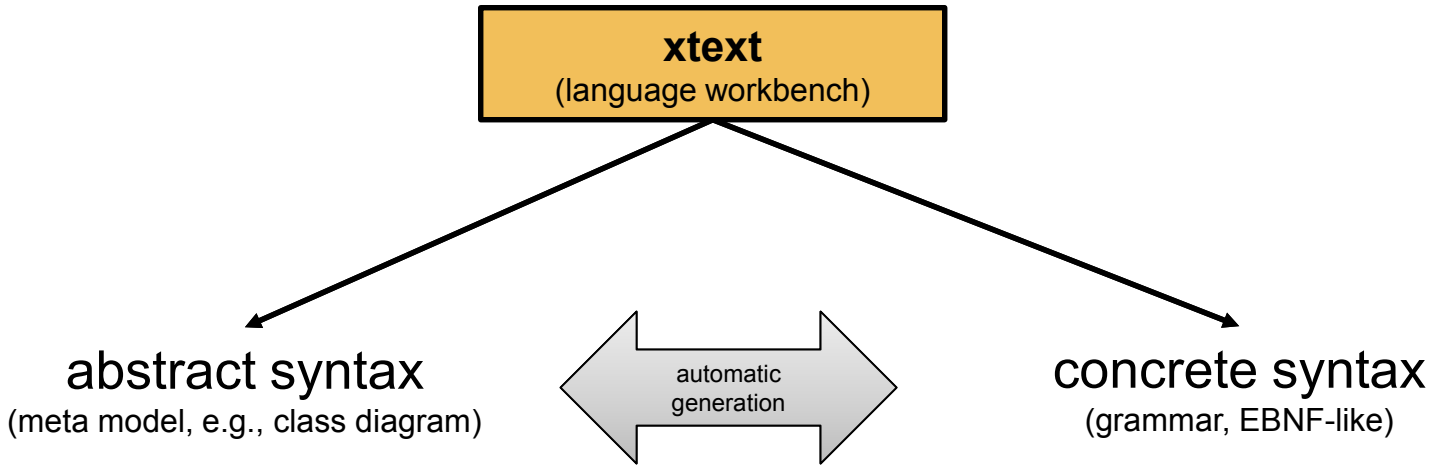
http://de.slideshare.net/deimos/markus-voelter-textual-ds-ls

Example of a Generated Xtext-based Editor



http://de.slideshare.net/deimos/markus-voelter-textual-ds-ls

How to start?



- No need to **use EMF** – Xtext allows starting with the grammar, and then generates a meta-model
- If starting with EMF, Xtext **generates a** default textual syntax; you should **adjust it later** to your taste

Xtext Walkthrough

more details:
https://eclipse.org/Xtext/documentation/301_grammarlanguage.html
http://www.eclipse.org/Xtext/documentation/1_0_1/xtext.pdf
xtext tutorial in 06 – concrete syntax.pdf

- Make sure the Xtext Complete SDK is installed
- Create the .ecore **file with the meta-model**
- Create the default **generator model** for the meta-model
- **Generate model code** using the generator model
- Add the Xtext nature to the project with the meta-model (project context menu -> Configure -> Convert to Xtext project)
- **Create an Xtext project** from existing Ecore models, using the genmodel
- **Generate Xtext code** (in the grammar, context menu: Run As -> Generate Xtext Artifacts)
- **Run** your first DSL editing environment (Eclipse application)
- **Iteratively revise** the grammar to increase usability

Xtext Grammar Specifications (1)

- **Declare** grammar name and **import** grammars that you **reuse**:

```
grammar mdsebook.Fsm with org.eclipse.xtext.common.Terminals
```

- **Import** meta-models (make types and methods available in the grammar):

```
2 import "http://www.mdsebook.org/mdsebook.fsm"
3 import "http://www.eclipse.org/emf/2002/Ecore" as ecore
```

- The first symbol (left-hand side of first production rule) is the **start symbol**:

```
4 FiniteStateMachine returns FiniteStateMachine:
5     'FiniteStateMachine' name=EString
6     '{'
7         'initial' initial=[State|EString]
8         'states' '{' states+=State ( ',' states+=State)* '}'
9     '}';
```

Xtext Grammar Specifications (2)

- **line 1: returns** introduces the type instantiated for a nonterminal
- Terminals are 'quoted' as strings

```
1 FiniteStateMachine returns FiniteStateMachine:
2    'FiniteStateMachine' name=EString
3    '{'
4       'initial' initial=[State|EString]
5       'states' '{' states+=State ( ',' states+=State)* '}'
6    '}';
```

- **line 2:** `EString` is a symbol matches string literal or ID from the imported grammar org.eclipse.xtext.common.Terminals
- **line 2: attribute** `name` refers to an attribute of FiniteStateMachine, the type of the currently parsed nonterminal
- **line 2:** `name=EString` : result of parsing the symbol is be stored in the respective attribute of the constructed object

Xtext Grammar Specifications (3)

```
terminal ID:
        ('^')?('a'..'z'|'A'..'Z'|'_') ('a'..'z'|'A'..'Z'|'_'|'0'..'9')*;
```

- starts with an optional character ('^‘)
- followed by a letter ('a'..'z'|'A'..'Z') or underscore '_'
- followed by any number of letters, underscores or/and numbers ('0'..'9')

- **line 5:** EBNF notation (parentheses, Kleene*, +, ?, etc are available)

```
1 FiniteStateMachine returns FiniteStateMachine:
2    'FiniteStateMachine' name=EString
3    '{'
4       'initial' initial=[State|EString]
5       'states' '{' states+=State ( ',' states+=State)* '}'
6    '}';
```

Xtext Grammar Specifications (4)

- **line 5:** assignment "+=" extends a collection
- **References:**
  - **line 4:** a more complex name of parsing an attribute. Square brackets mean that the parsed `EString` is a reference to a State object
  - In general: storingAttribute=[ReferredType|ReferenceSyntax]
  - Object needs to have an attribute `name` for this to work automatically

```
1 FiniteStateMachine returns FiniteStateMachine:
2    'FiniteStateMachine' name=EString
3    '{'
4       'initial' initial=[State|EString]
5       'states' '{' states+=State ( ',' states+=State)* '}'
6    '}';
```

  - `ReferenceSyntax` defaults to `ID` if omitted

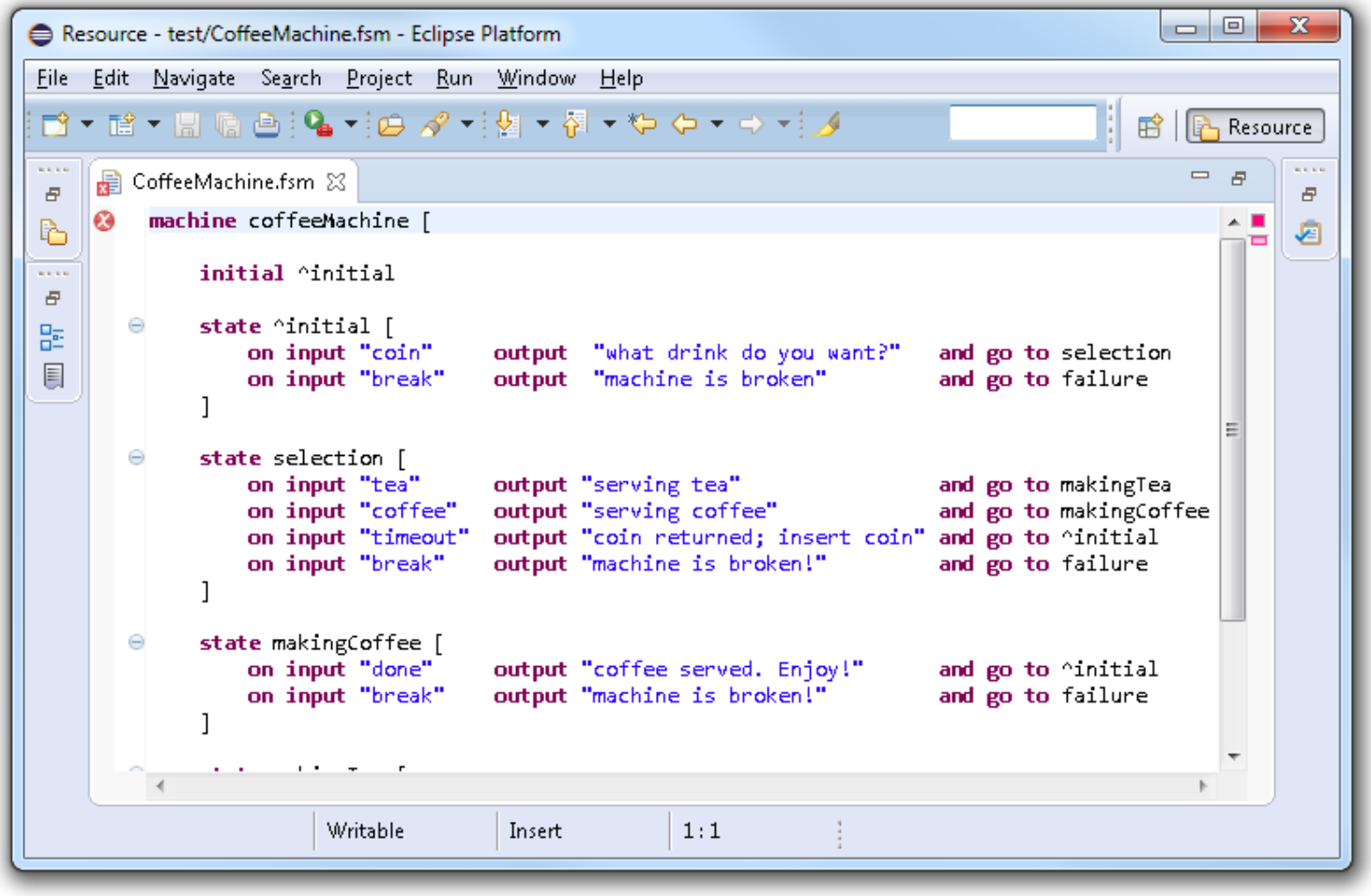
A Grammar for Finite State Machines

- The previous snippets were automatically generated by Xtext
- Let's modify that a bit to obtain our preferred grammar:

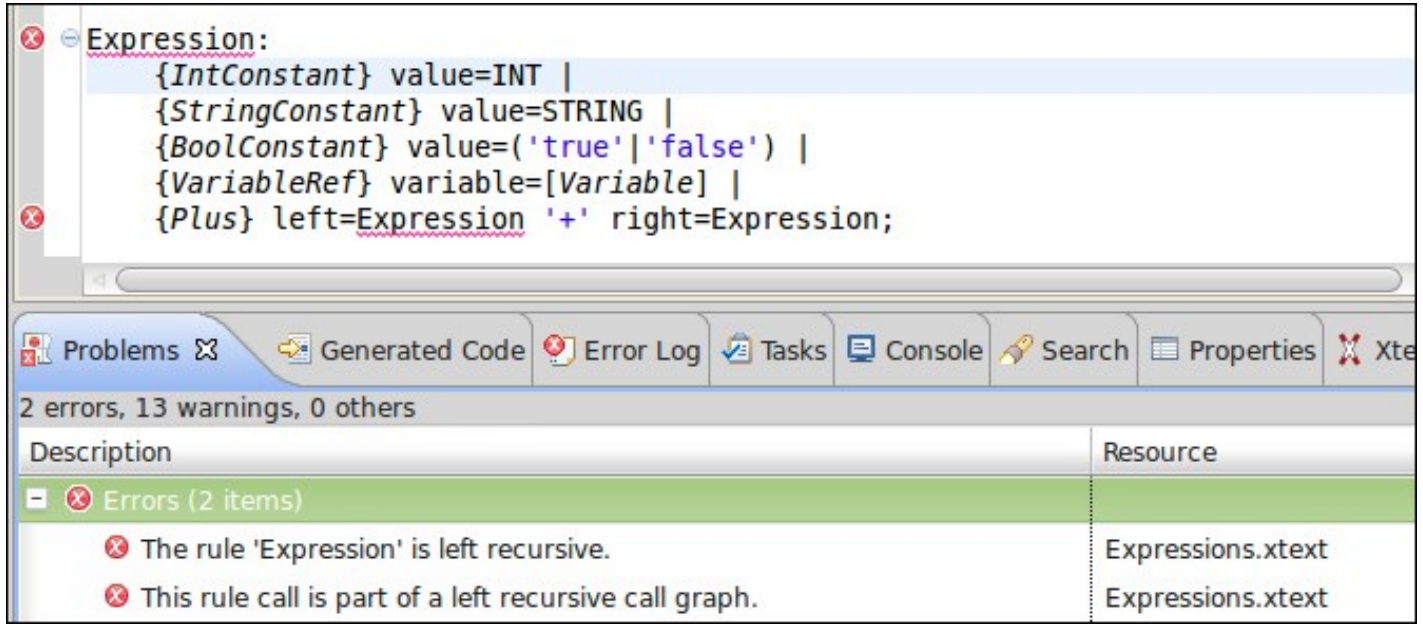
```
1  // (c) mdsebook, wasowski, berger
2  grammar mdsebook.Fsm with org.eclipse.xtext.common.Terminals
3
4  import "http://www.mdsebook.org/mdsebook.fsm"
5  import "http://www.eclipse.org/emf/2002/Ecore" as ecore
6
7  FiniteStateMachine:
8      'machine' name=ID
9      ('[' 'initial' initial=[State|ID]
10         (states+=State)*
11     ']')?;
12
13 State:
14     'state' name=ID ('[' leavingTransitions+=Transition* ']')?;
15
16 Transition:
17     'on' 'input' input=STRING ('output' output=STRING)?
18     'and' 'go' 'to' target=[State];
```

Syntax for Finite State Machines

- And launch our generated editor



Left Recursion



[Be16]

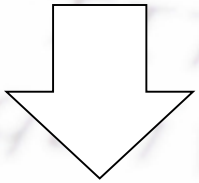
Left-Recursion

left-recursion

- That grammar was not only ambiguous but also left-recursive
- A **nonterminal X is left-recursive** if the left-most symbol in any of its productions is X itself, or rewrites to X again.
- A **grammar is left-recursive** if it has a left-recursive nonterminal.
- **LL parsers can't** deal with left-recursive grammars (incl. ANTLR and hence Xtext)

```
expr → expr ('+'|'*') expr
expr → INT | FLOAT | ID
expr → '(' expr ')'
```

expr is both left- and right-recursive above (but right-recursive is fine for LL-parsers)



```
term → factor '*' factor
term → factor
factor → INT | FLOAT | ID
factor → '(' expr ')'

expr → term ( '+' term )*
```

Xtext (15 minutes)

In the handout you find the Xtext grammar and an example instance for the first Lego DSL (see Metamodels from the earlier handout).

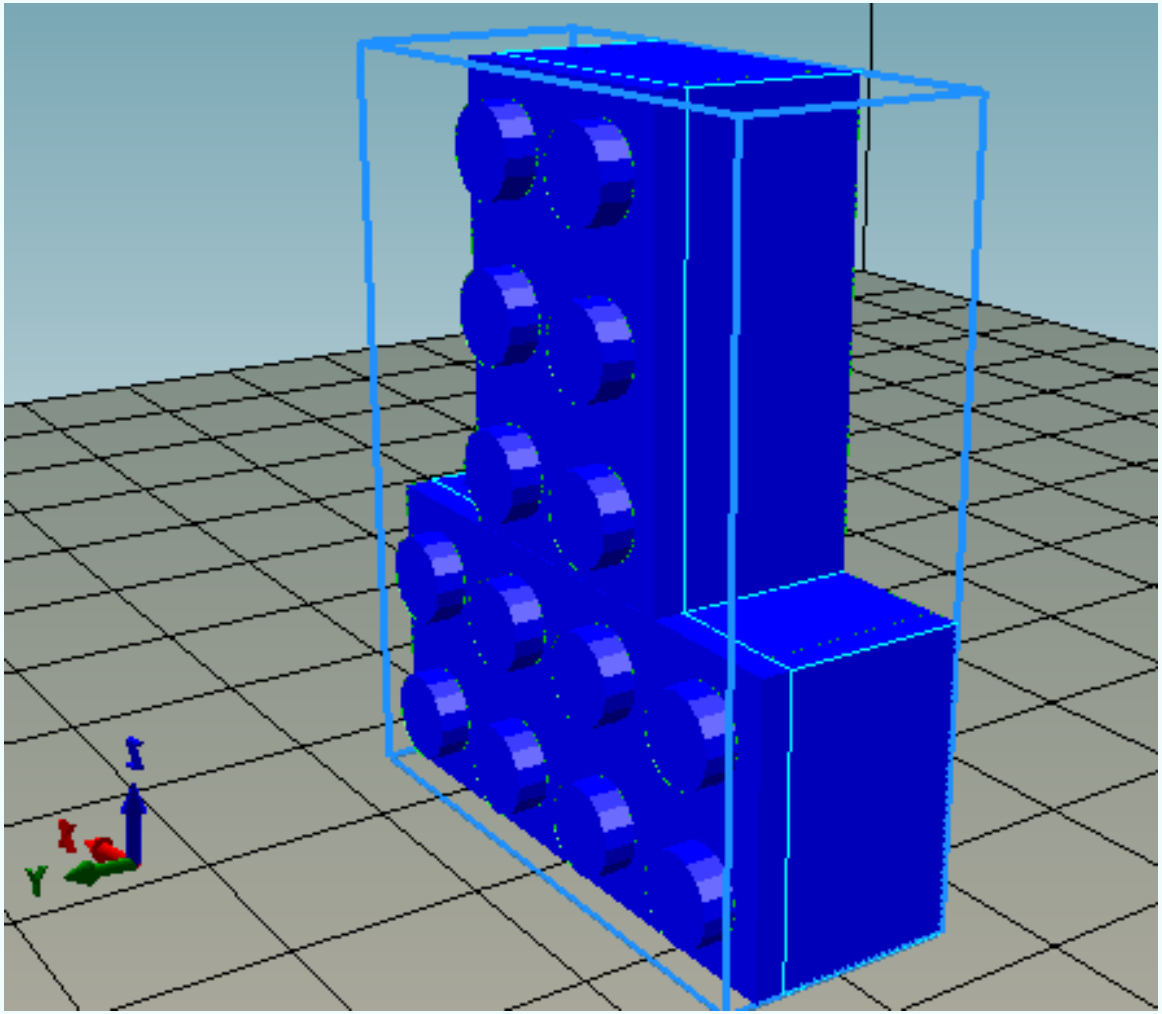
Together with your neighbor:

1. Try to map metamodel classes and associations to elements in the grammar. Which elements do you find reflected and how? How are associations and containment relationship represented?

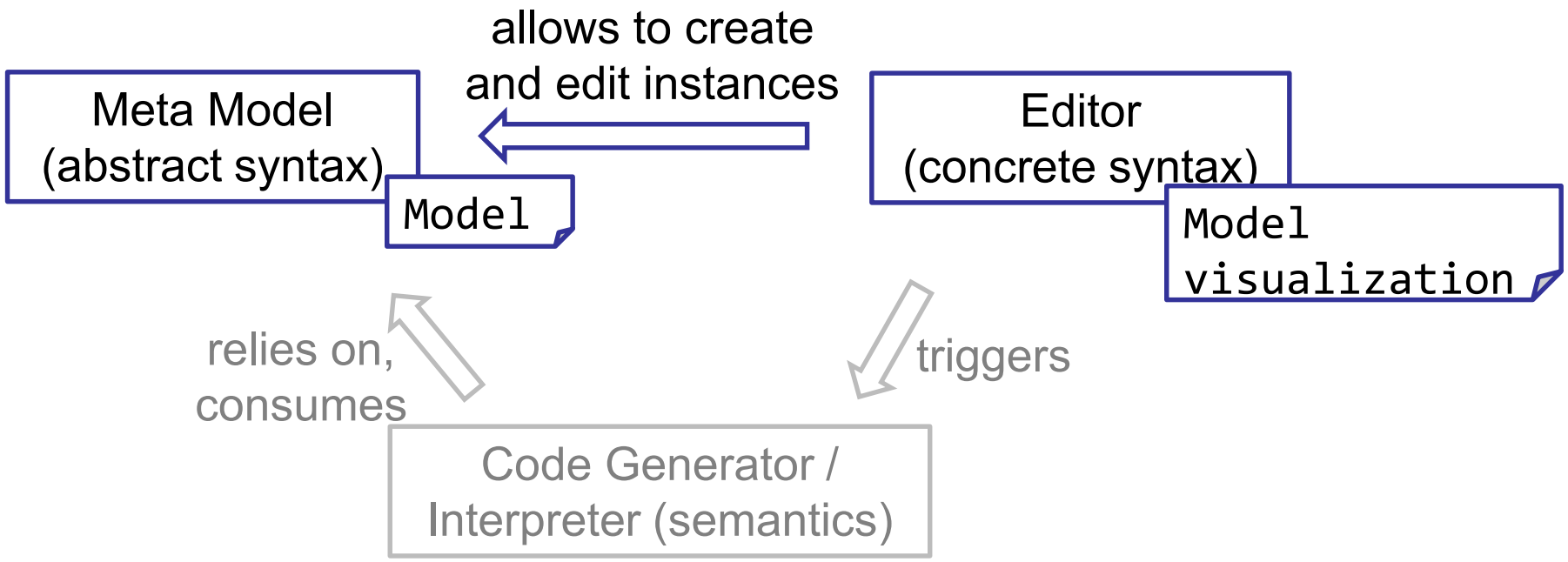
2. Have a look at the instance below. Make a guess what it could describe.

https://eclipse.org/Xtext/documentation/301_grammarlanguage.html

Xtext (15 minutes)



Wrap-up Syntax & Outlook Semantics

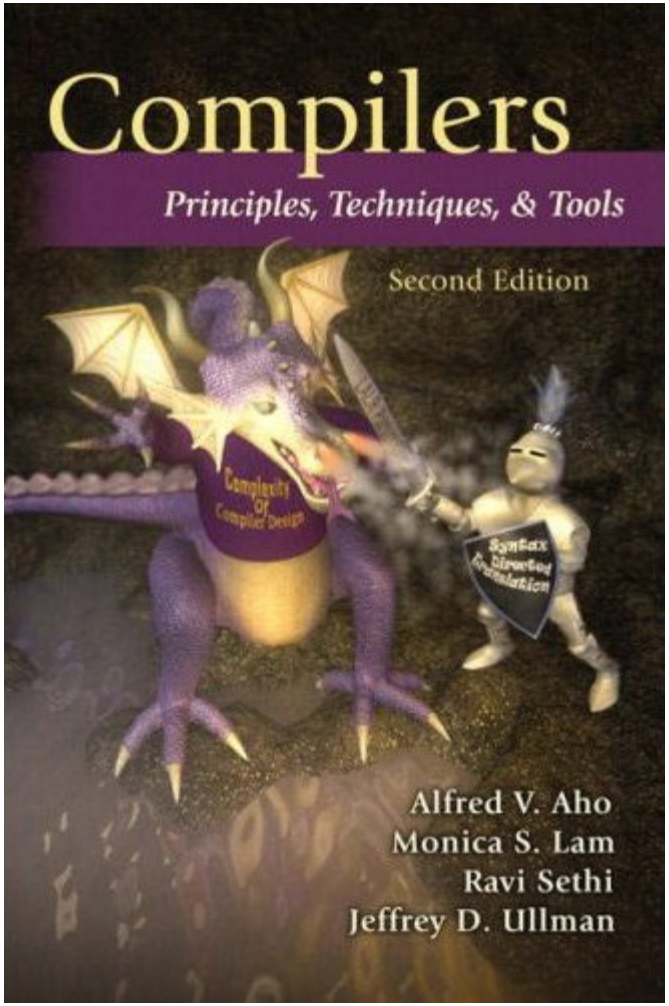


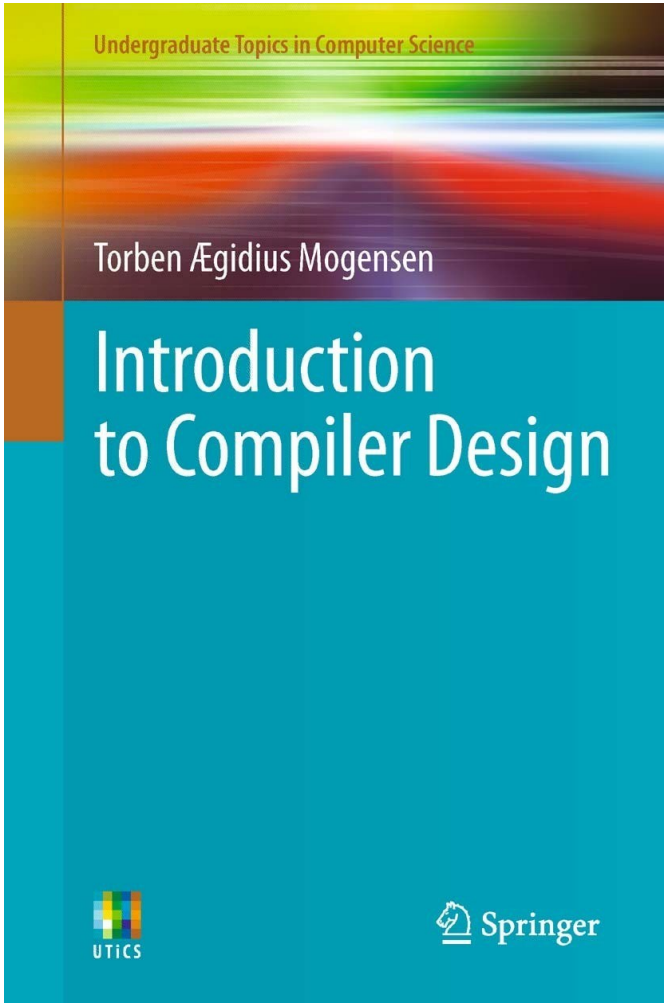
- Semantics

- How to prepare:
  - Download the assignment and code already – we will have a look at the assignment and you can ask questions

Model-Driven Engineering (MDE)

Regina Hebig, Thorsten Berger


Compilers
Principles, Techniques, & Tools
Second Edition
Alfred V. Aho
Monica S. Lam
Ravi Sethi
Jeffrey D. Ullman

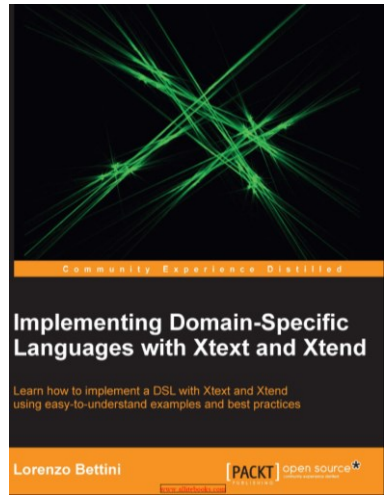

Undergraduate Topics in Computer Science
Torben Ægidius Mogensen
Introduction
to Compiler Design
UTiCS
Springer

Literature - Metamodeling

- Book
  - 03 - meta-modeling.pdf
  - appendix b - class modeling.pdf
  - appendix c - using eclipse emf.pdf

Literature

- Book
  - Chapter 6 (Concrete Syntax)
    - contains an appendix with an Xtext tutorial



- Xtext
  - [Be16] Bettini, Lorenzo. *Implementing domain-specific languages with Xtext and Xtend*. Packt Publishing Ltd, 2016.
    - freely available ebook
    - very hands-on description of Xtext
- Xtext documentation and guide
  - https://www.eclipse.org/Xtext/documentation/2.5.0/Xtext%20Documentation.pdf