MDSD in Practice

Xtext

What is Xtext?

- a suite of tools to assist MDSD
- particular focus: domain specific languages

- origin: openArchitectureWare (oAW)

- now part of Eclipse‘s Modeling Framework (EMF)
  - → migration of the project in 2008
  - → major contributions by itemis AG
  - → Open Source, Eclipse Public License (EPL)
- http://www.eclipse.org/Xtext

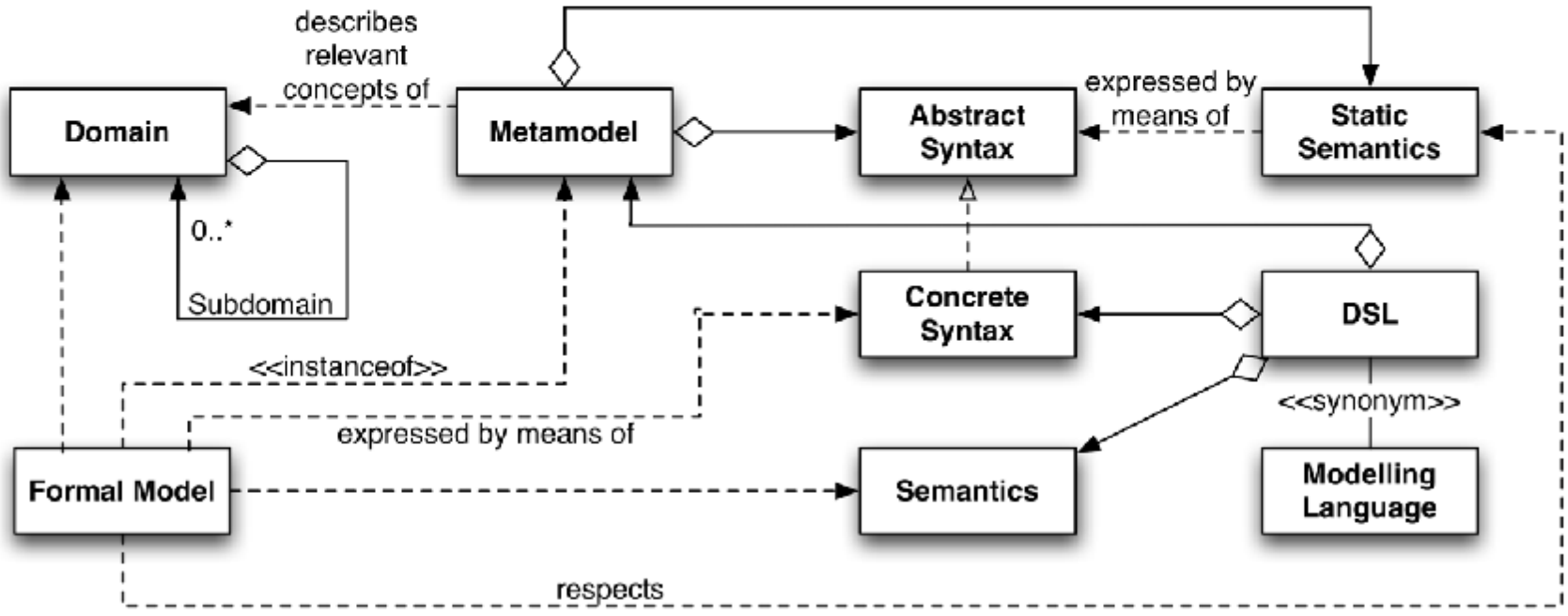
Xtext

Main features of Xtext

- Covers various DSL implementation aspects (parsers, editors, analyzers, code generation)
  - Parsers: ANTLR integrated
  - ASTs: Build on top of Ecore object meta model
  - Editors: automatically generated
    (syntax highlighting, code completion, static analyses, ...)
  - analyzers: grammars with references (ASTs are DAGs)
    allow static semantic checks
  - Powerful template engine
    Interpretation by JVM

MDSD recap

MDSD: Concepts & Notions in our project

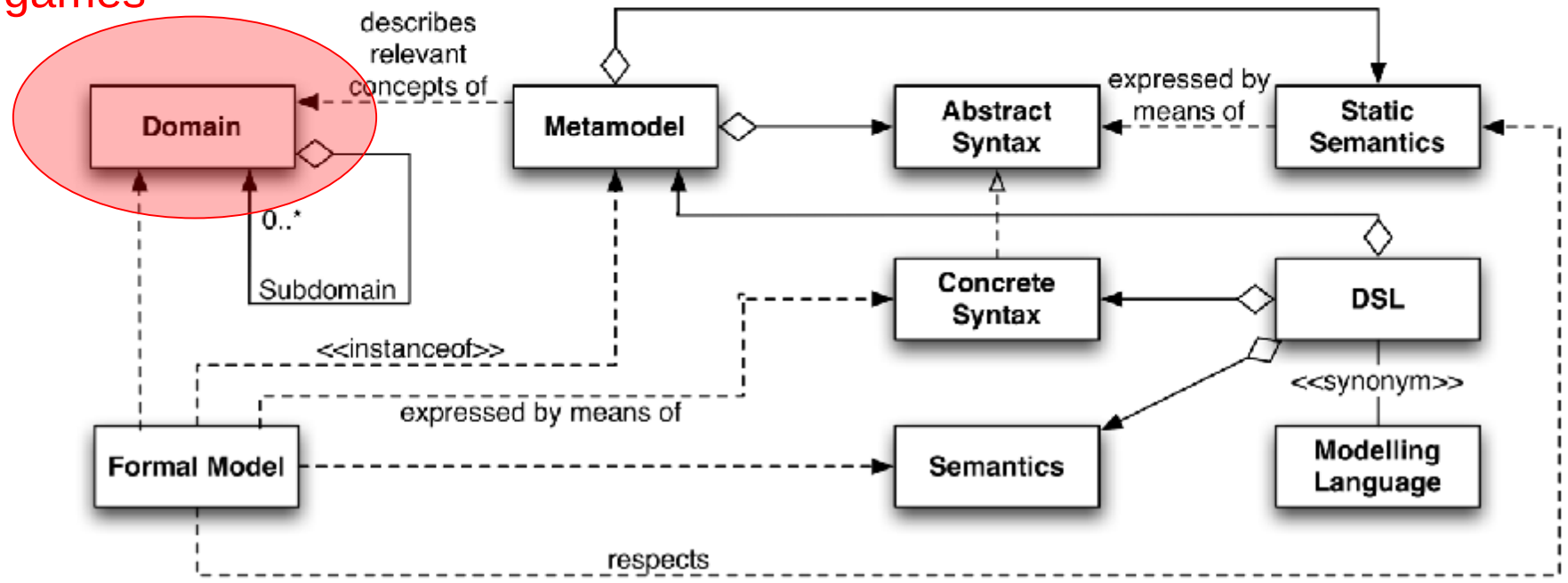


Source: [MDSD], S. 28

MDSD recap

MDSD: Concepts & Notions in our project

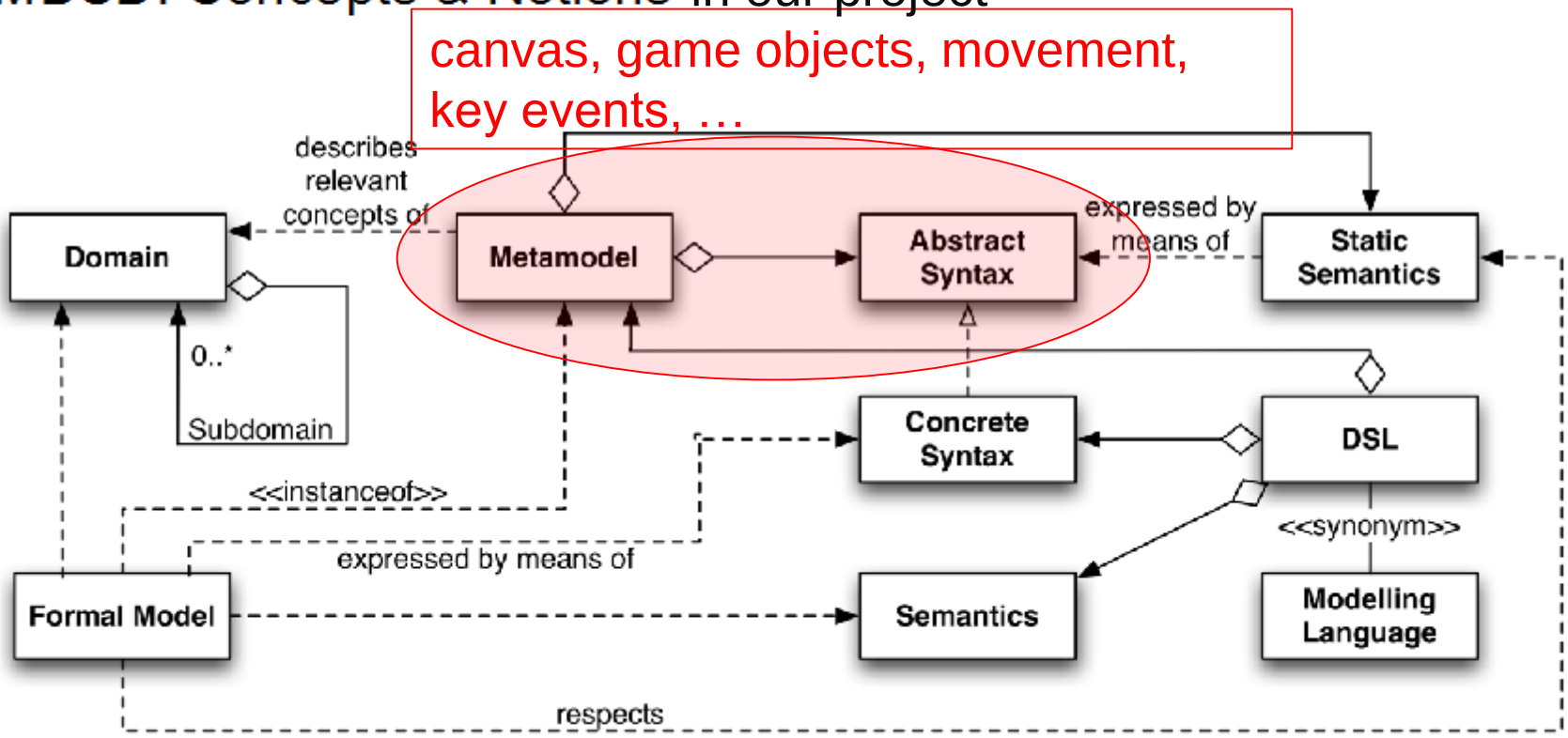
Basic arcade games



Source: [MDSD], S. 28

MDSD recap

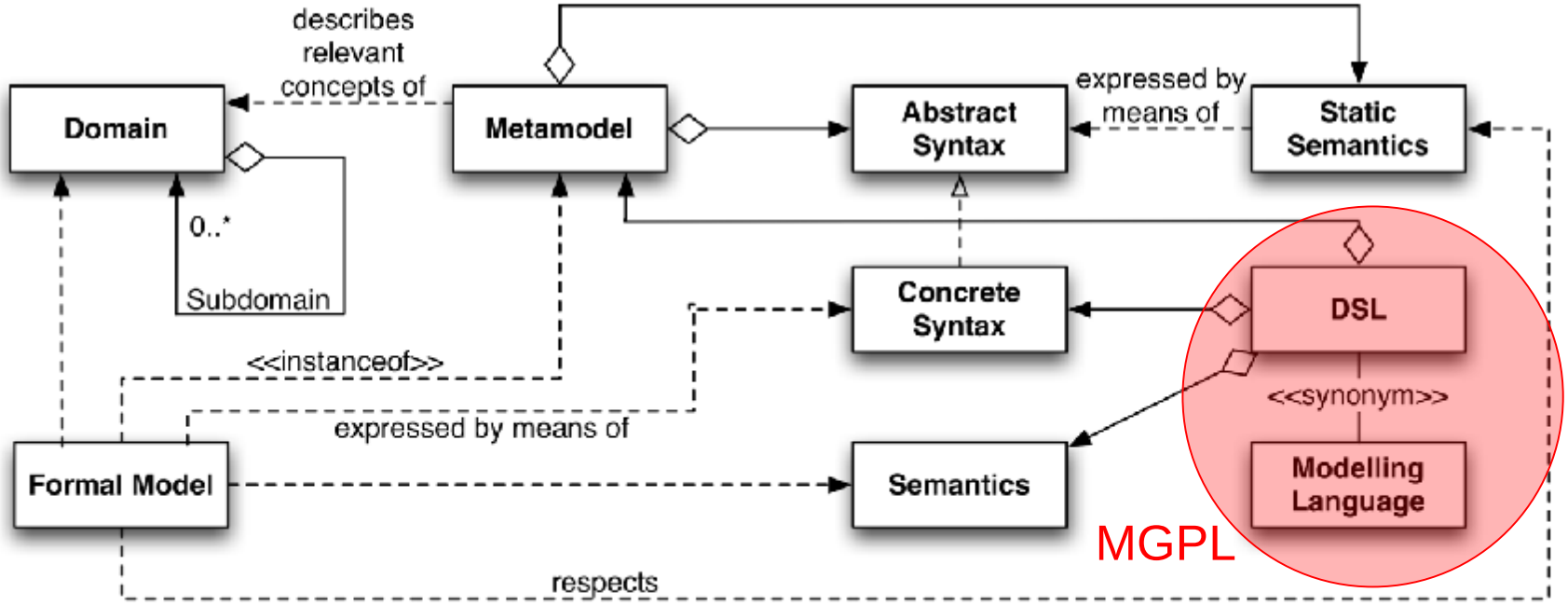
MDSD: Concepts & Notions in our project



Source: [MDSD], S. 28

MDSD recap

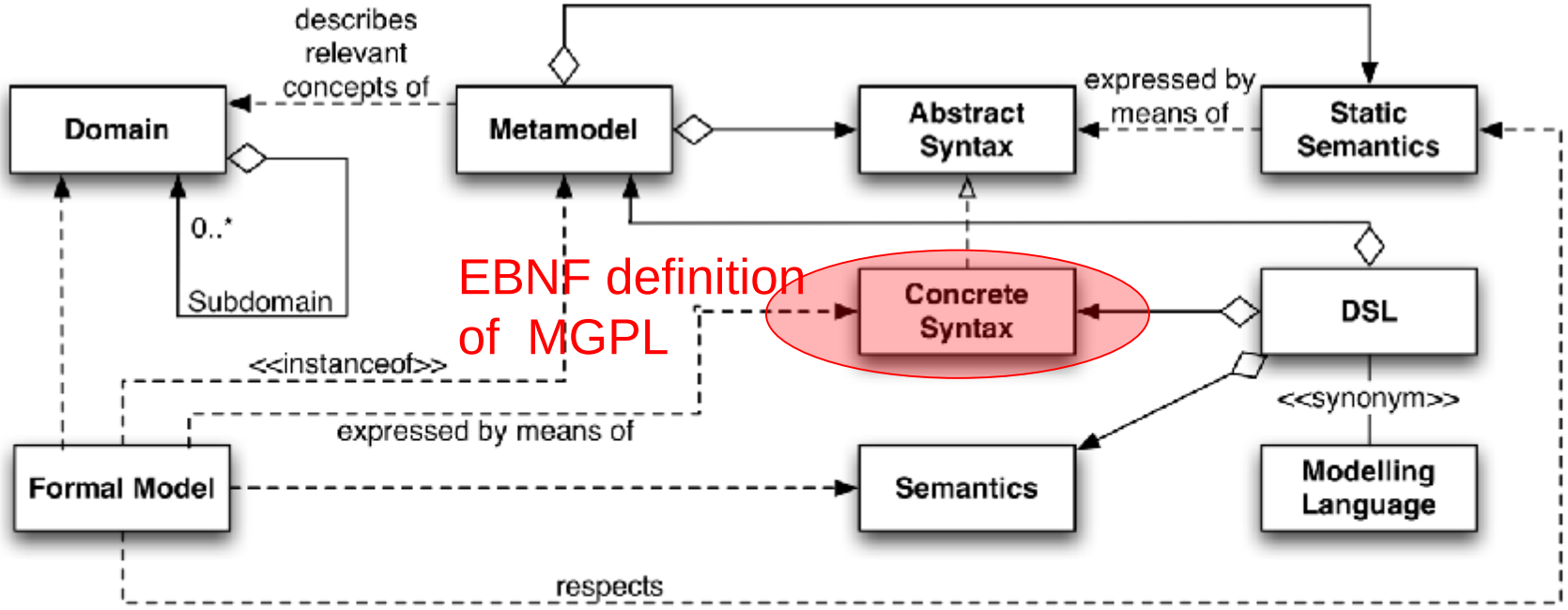
MDSD: Concepts & Notions in our project



Source: [MDSD], S. 28

MDSD recap

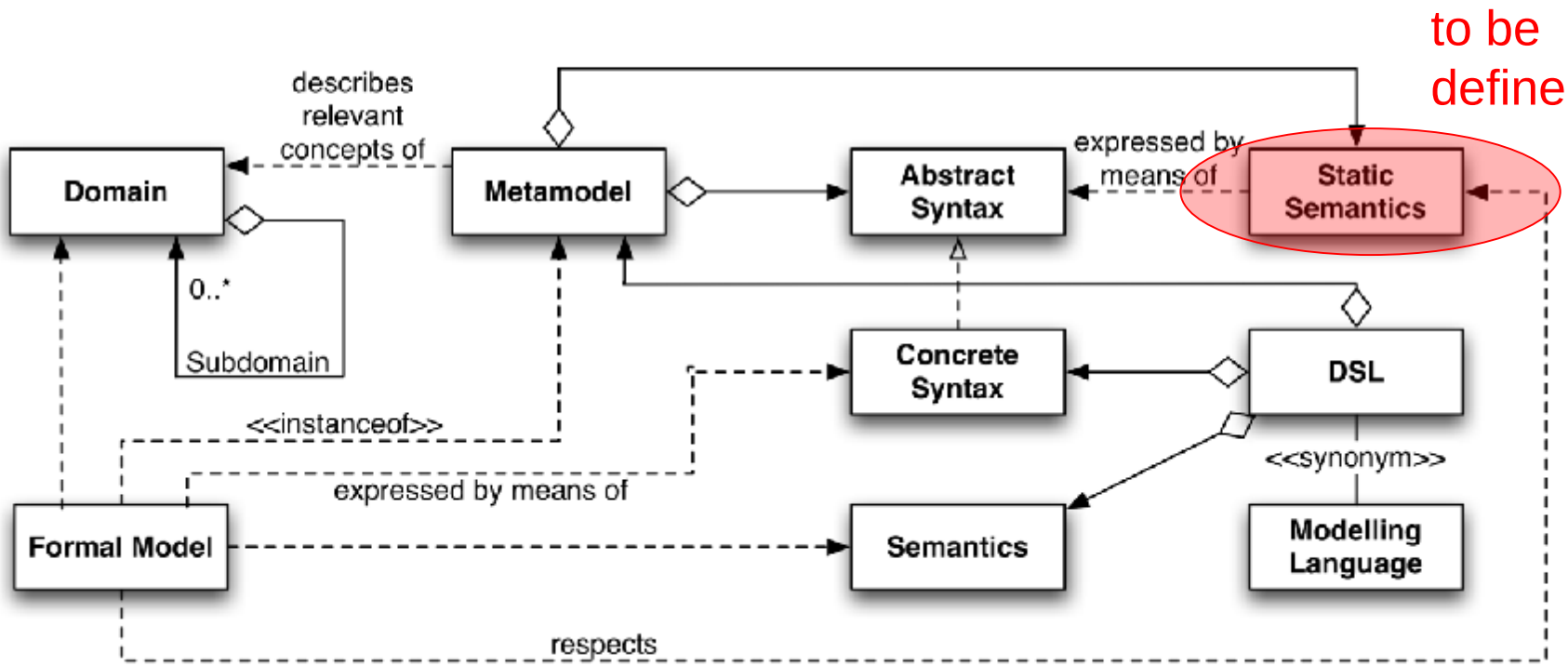
MDSD: Concepts & Notions in our project



Source: [MDSD], S. 28

MDSD recap

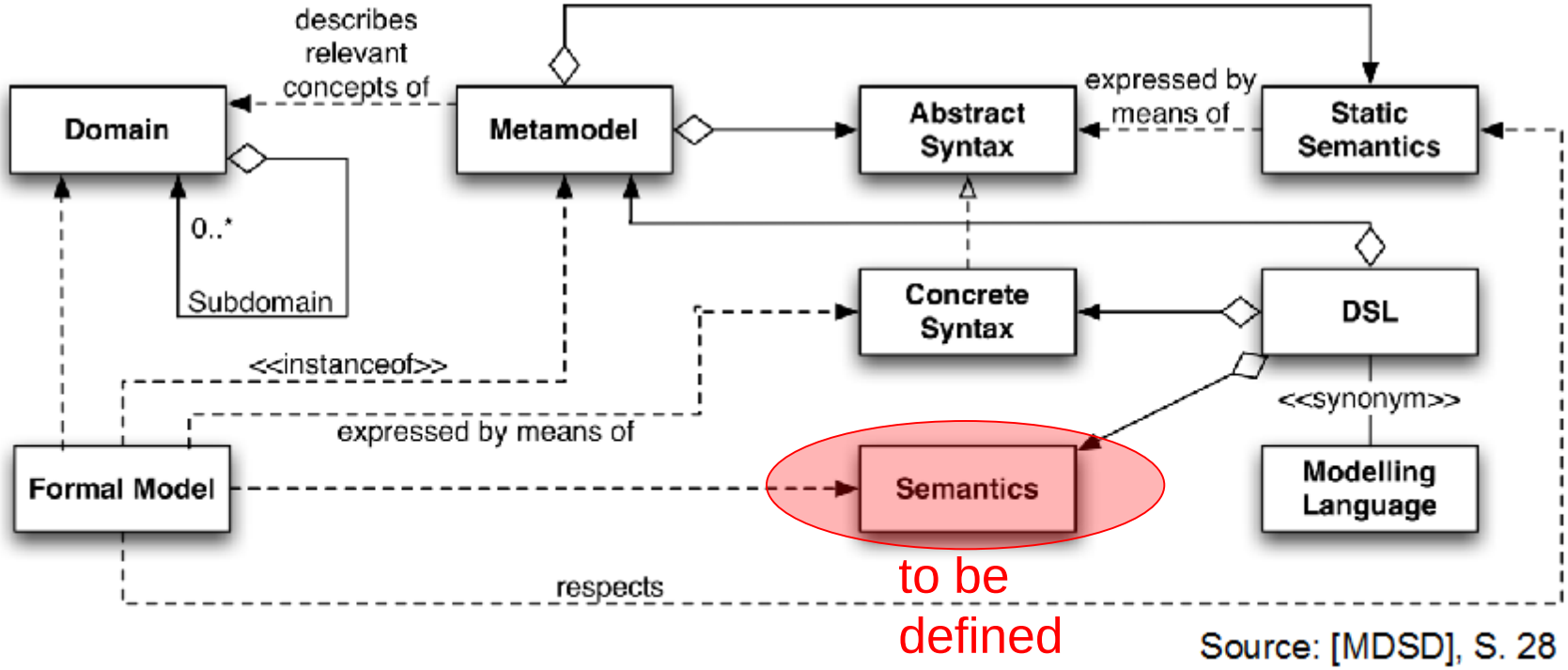
MDSD: Concepts & Notions in our project



Source: [MDSD], S. 28

MDSD recap

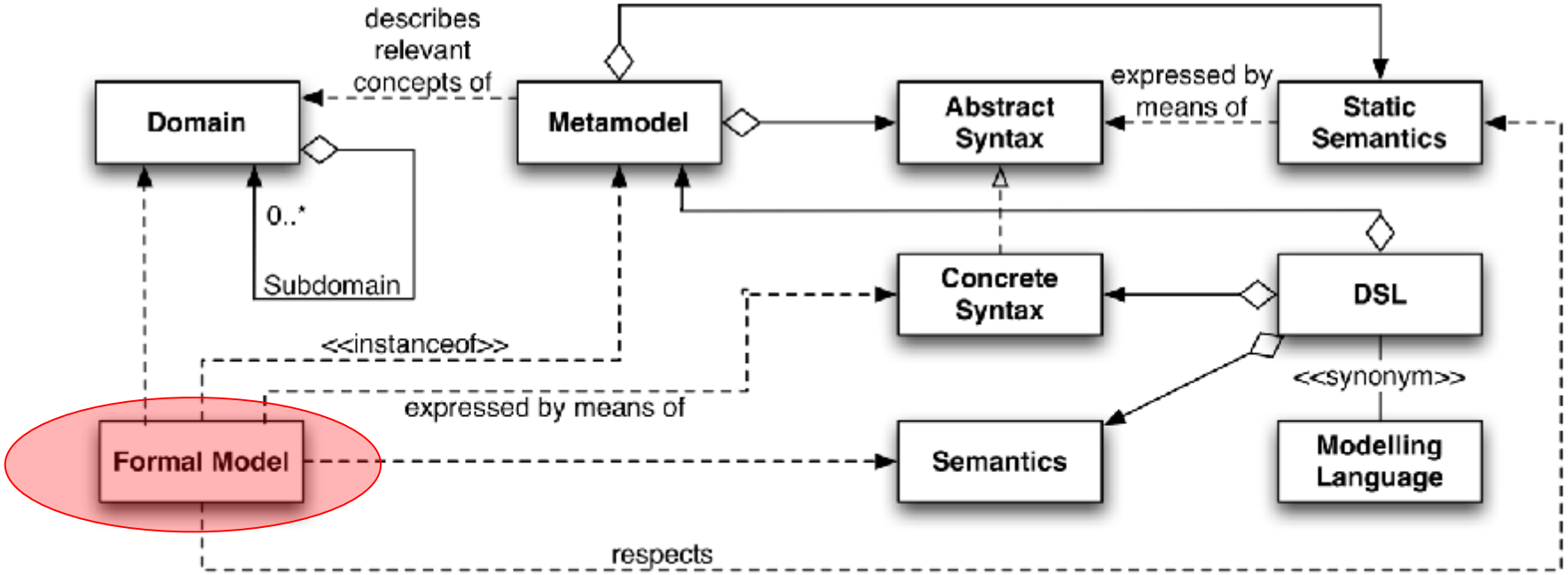
MDSD: Concepts & Notions in our project



Source: [MDSD], S. 28

MDSD recap

MDSD: Concepts & Notions in our project

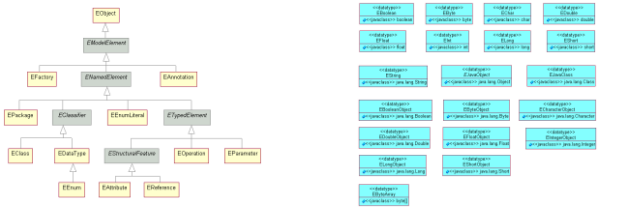


Source: [MDSD], S. 28

Pong,
Invaders,..

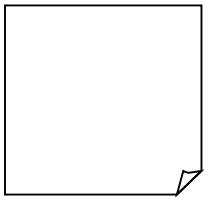
Model Hierarchy in our project

based on **Ecore**
+ EBNF facilities



M3: Metametamodel

MGPL grammar

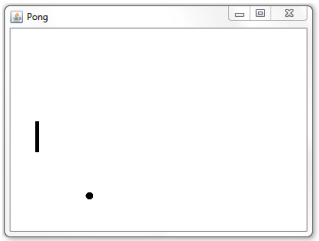


M2: Metamodel

MGPL program

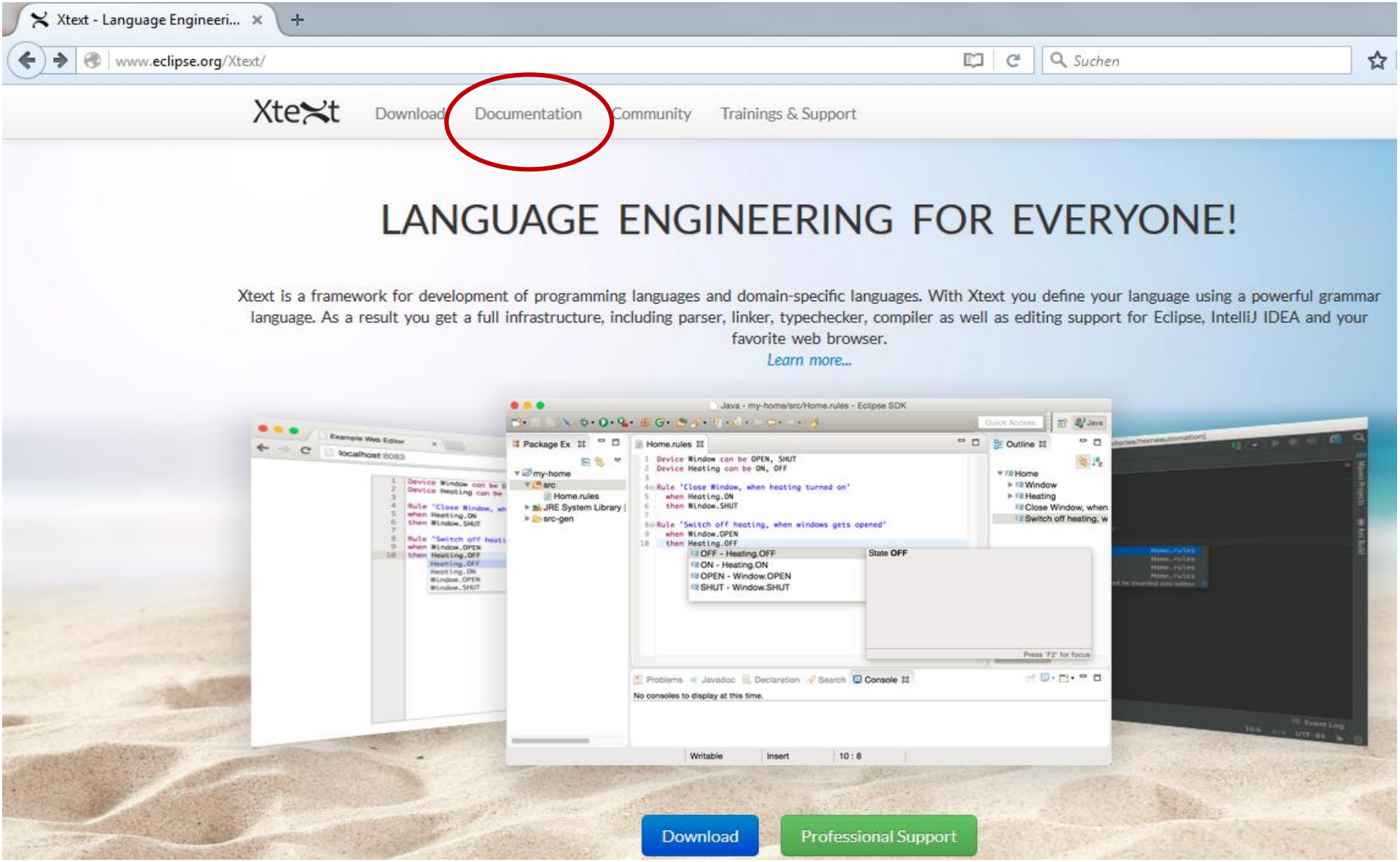
M1: Model

Java application



M0: Model instance

The tool – Xtext



Xtext documentation site

How to get started?

If you are not quite sure yet whether Xtext is an appropriate solution for your task, we recommend to watch the presentation below, as it gives you a good overview of what Xtext can do. If you want to dive in an learn, you should start with one of the getting started tutorials on the left. Also having the book as an additional resource to the online documentation is certainly a good idea.

Presentation - DSL Development With Xtext

An introduction and overview of Xtext, given in September 2015 at JavaZone in Oslo, Norway. In this presentation a rule-language for home automation is developed.

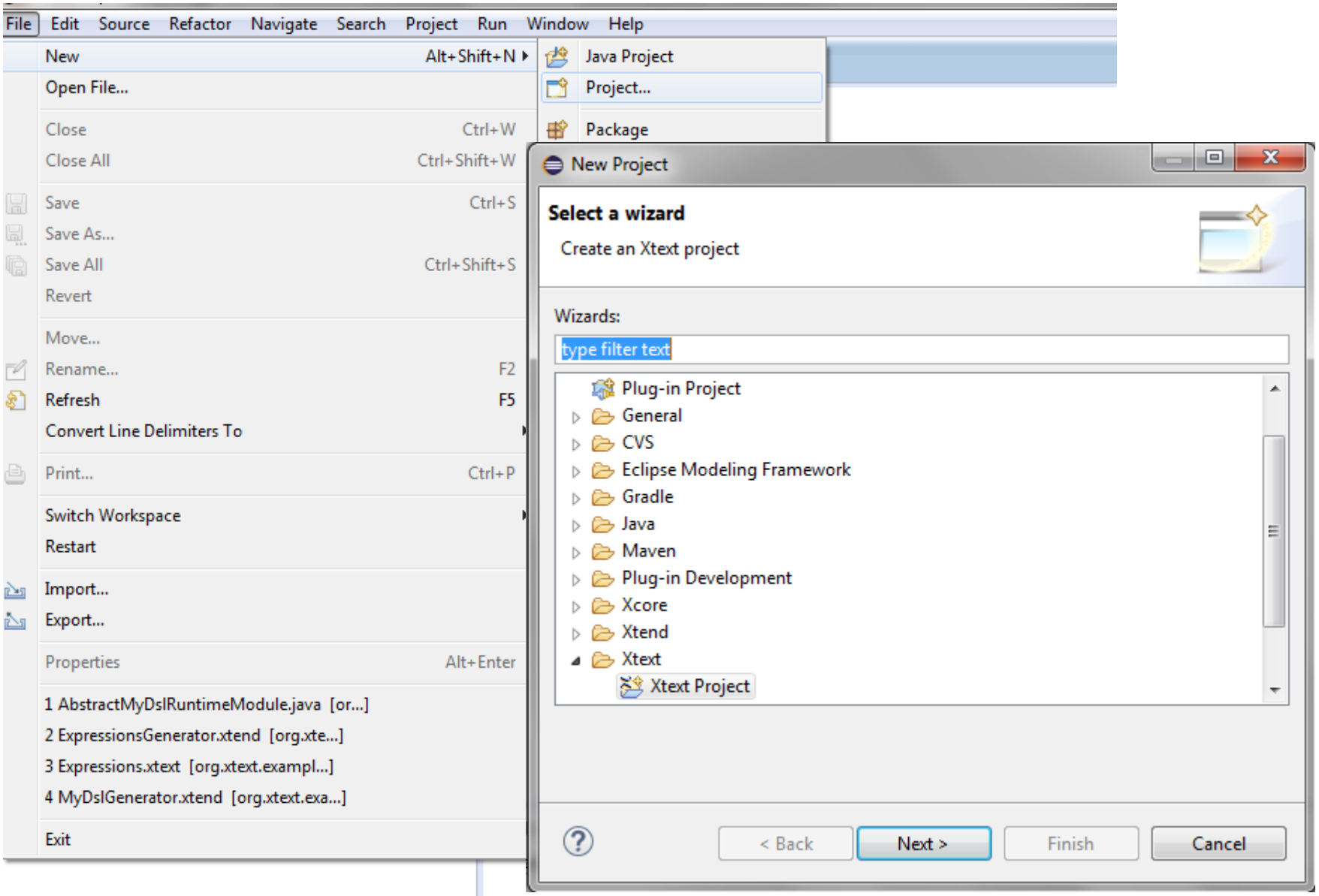
5 Minute Tutorials

We have written two tutorials that only take you 5 minutes to do

5 Minutes with Eclipse | 5 Minutes with Intellij

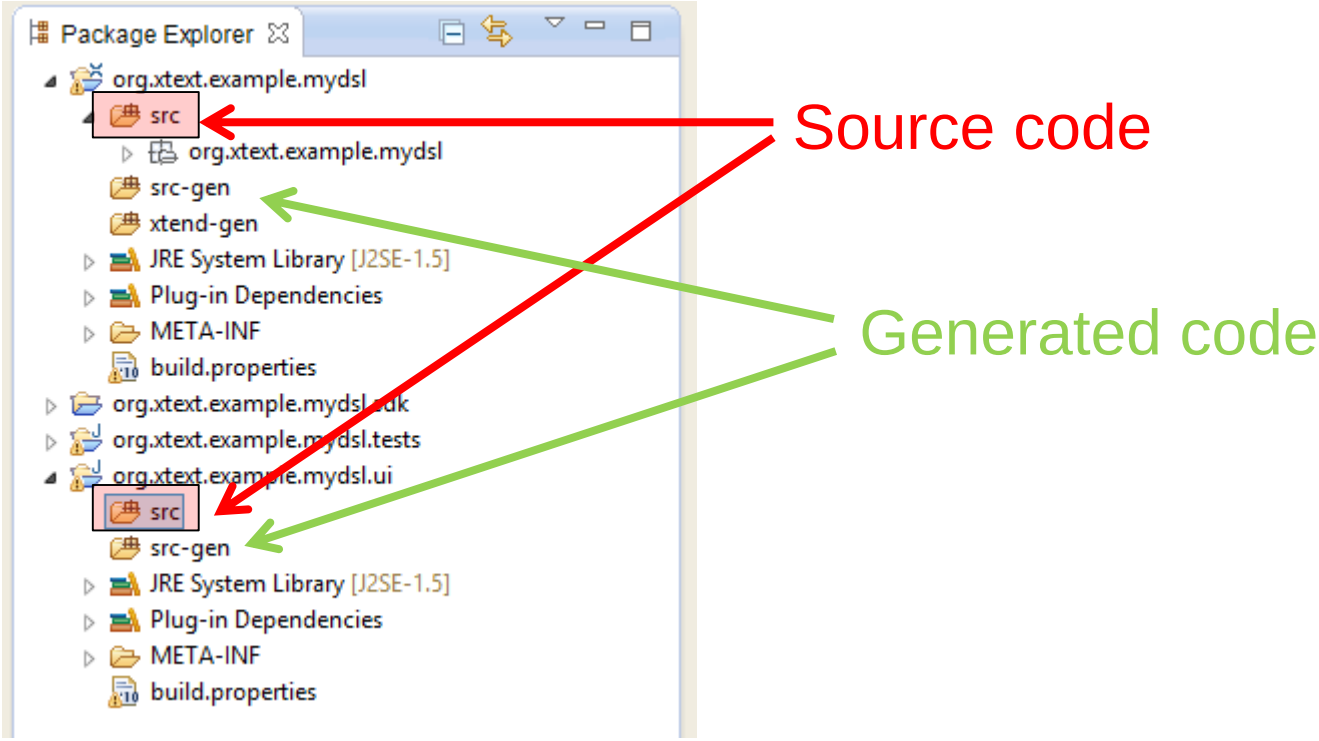
IF you have done one of these tutorials, you should check out the 15 minute one on the left.

Xtext Project

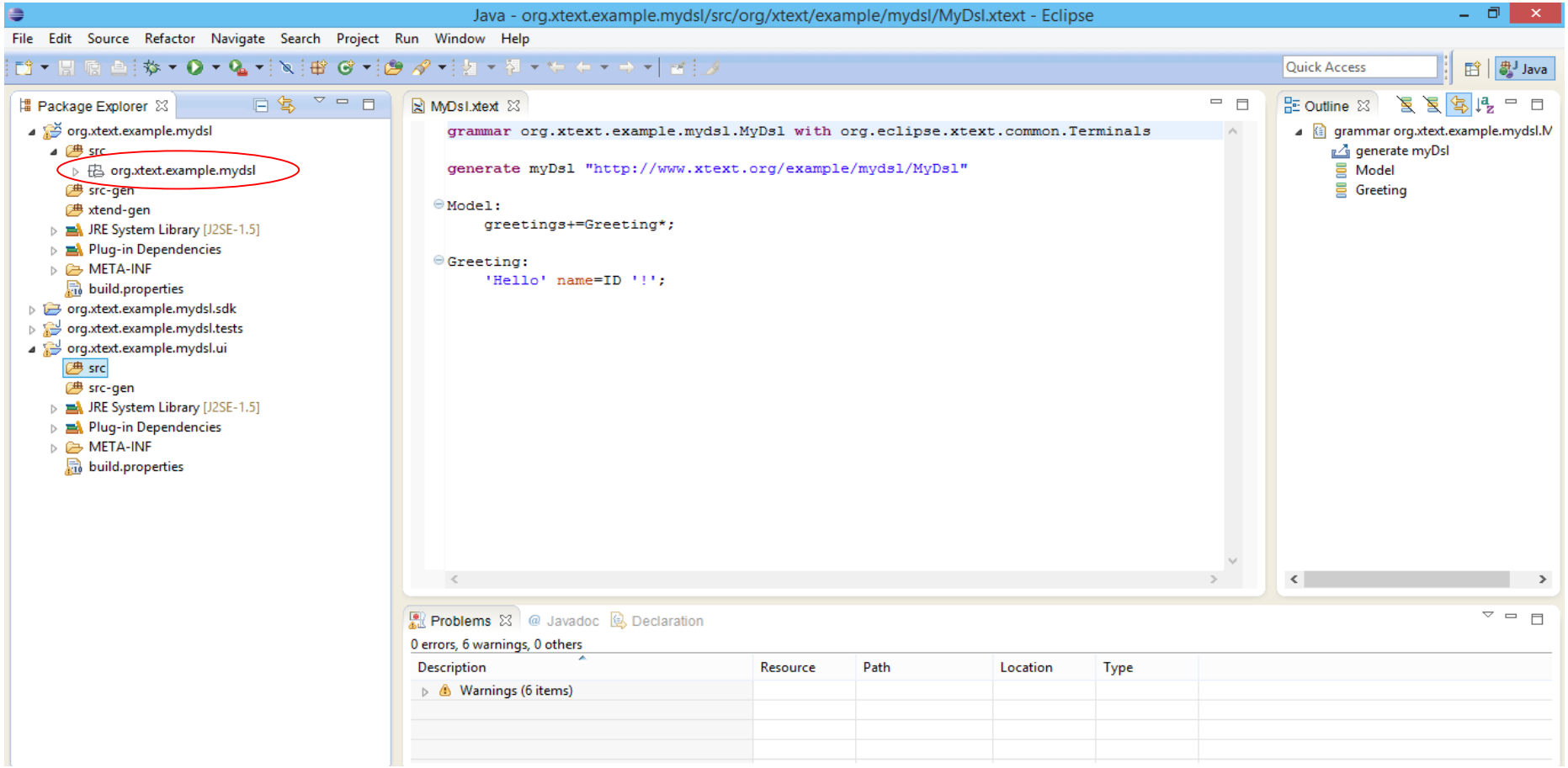


Xtext: Project overview

Using Xtext



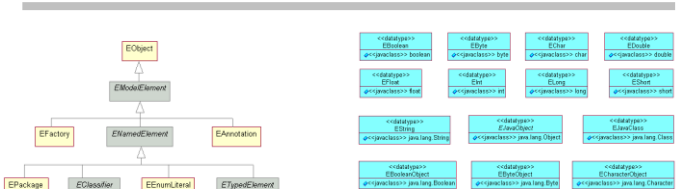
Using Xtext



Grammar file of DSL automatically opens editor preconfigured with "Hello world" grammar

recap: My Data Model

Metamodeling Example 3: Eclipse Modeling Framework



Metametamodel

platform:/resource/my.generator.project/src/metamodel/metamodel.ecore
metamodel
Model
types : Type
Type
name : EString
Datatype -> Type
Entity -> Type
features : Feature
Feature

lel

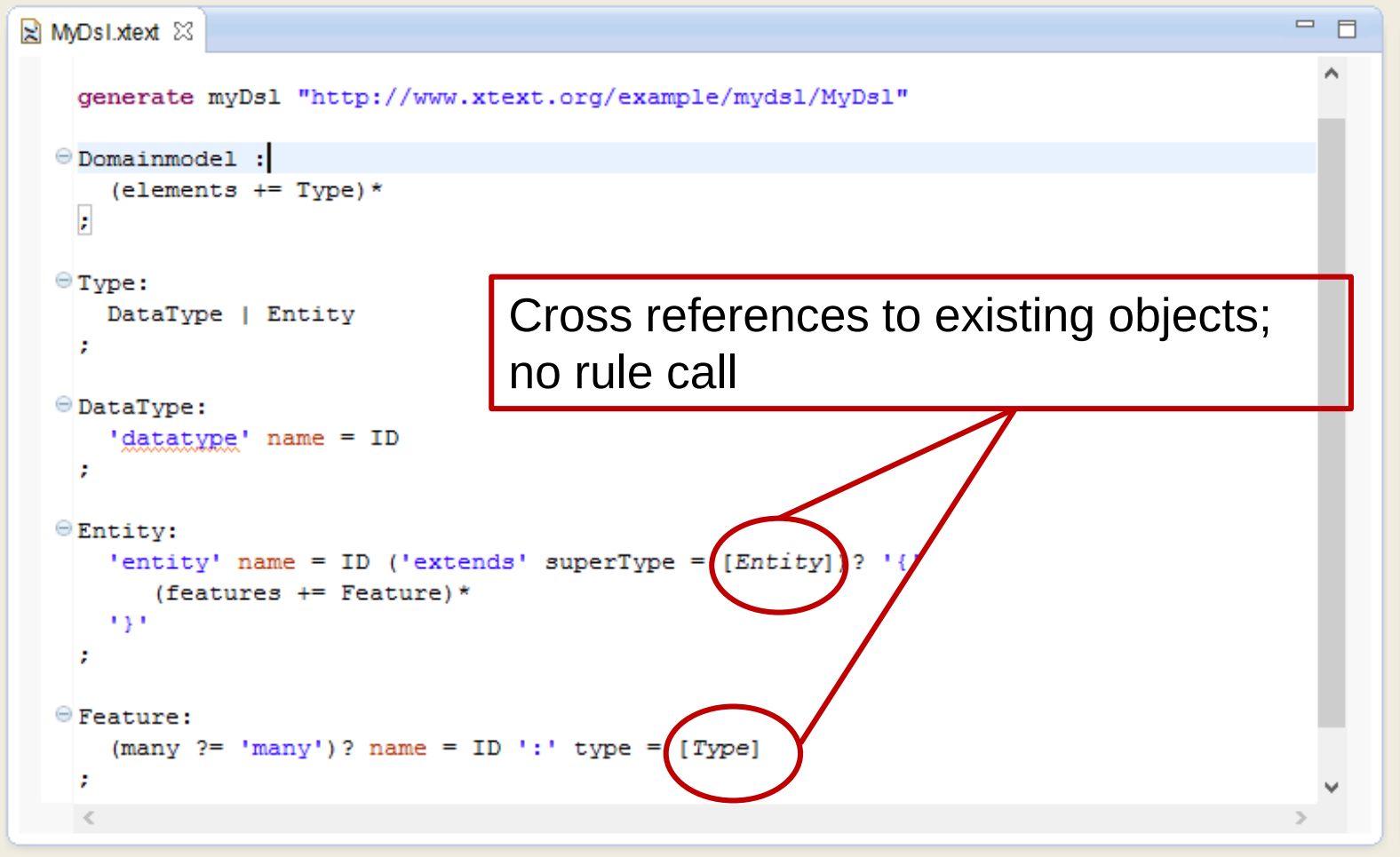
Entity Person
Feature name
Feature age
Feature address
Entity Address

Model

```
Person bob = new Person();
bob.setName("Bob");
bob.setAge(35);
Address address = new Address();
address.setStreet("Mulholland Drive");
address.setZip("8079");
address.setCity("Los Angeles");
```

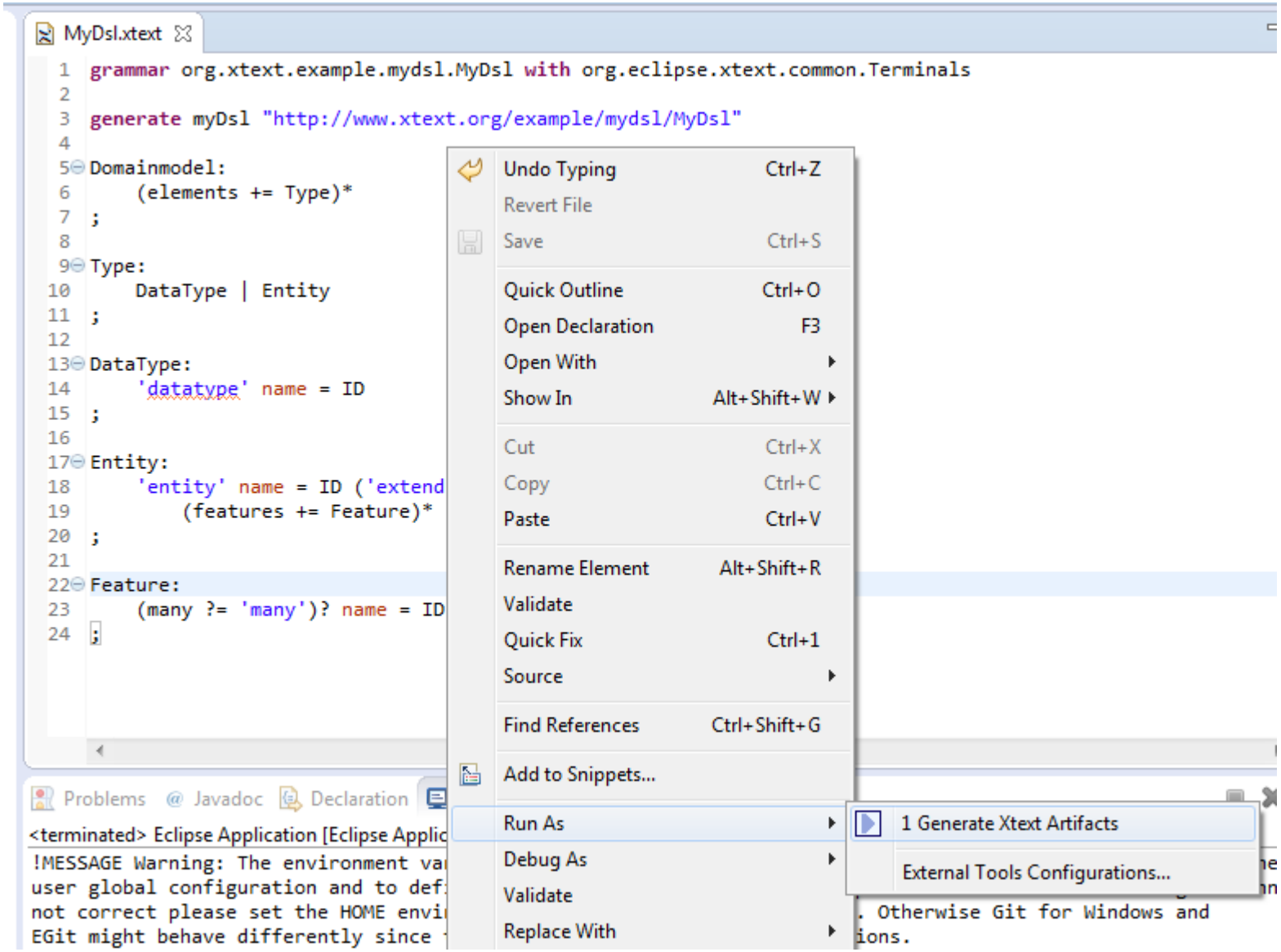
Model Instance

Syntax specification

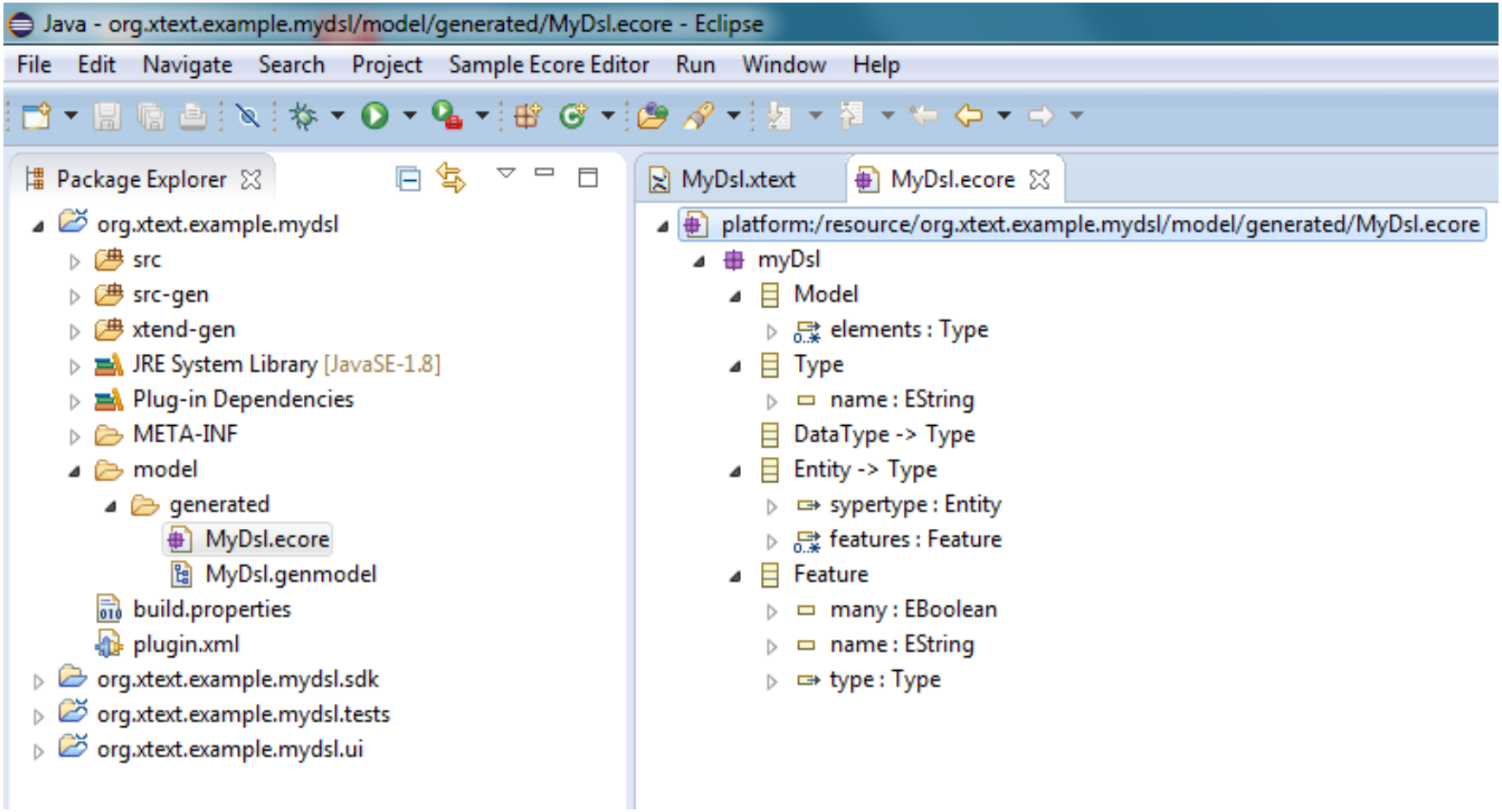


Edit grammar file by specifying your own DSL

Xtext: Generate editor, parser, Ecore model

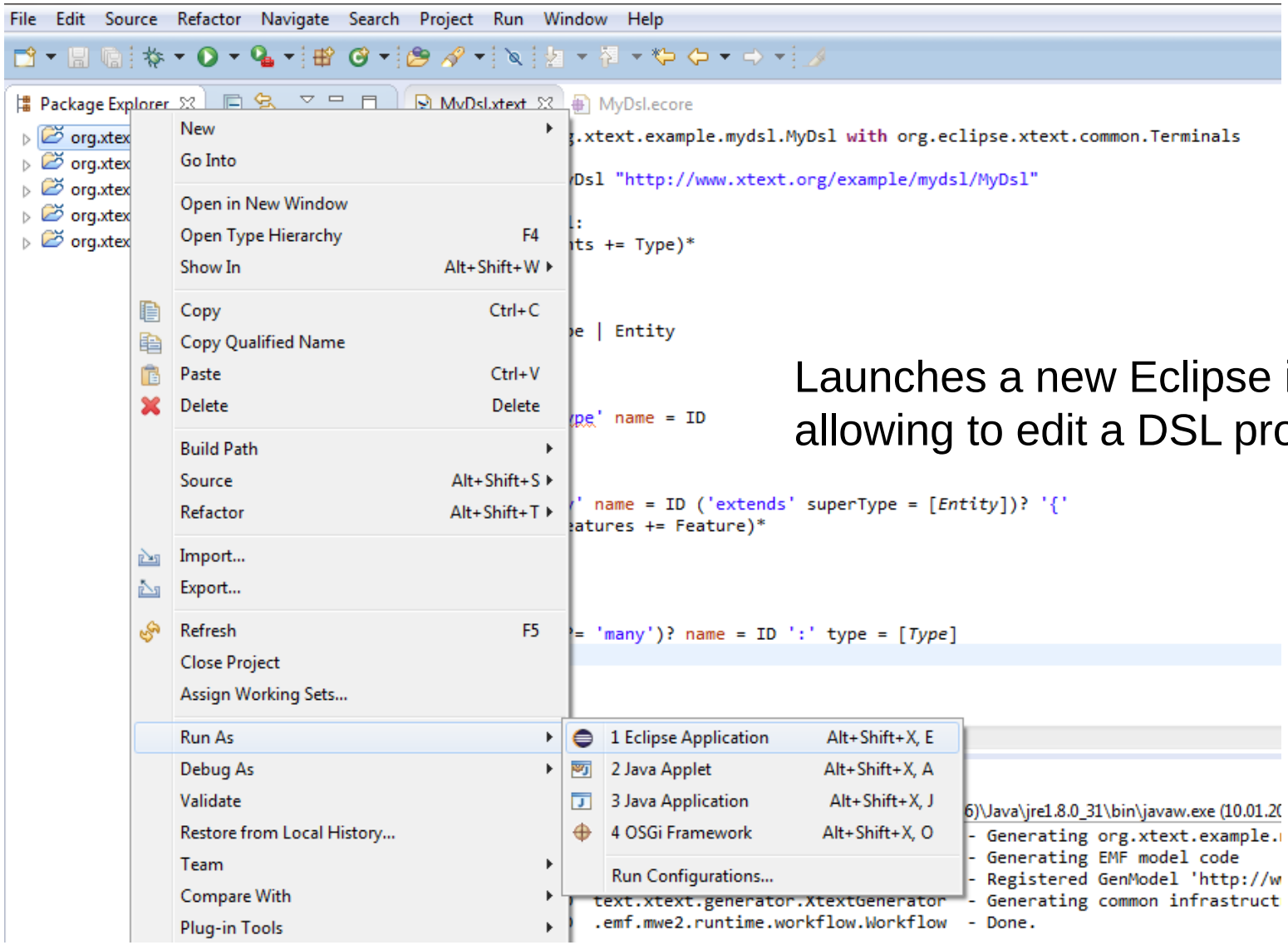


Xtext: Generated Ecore model



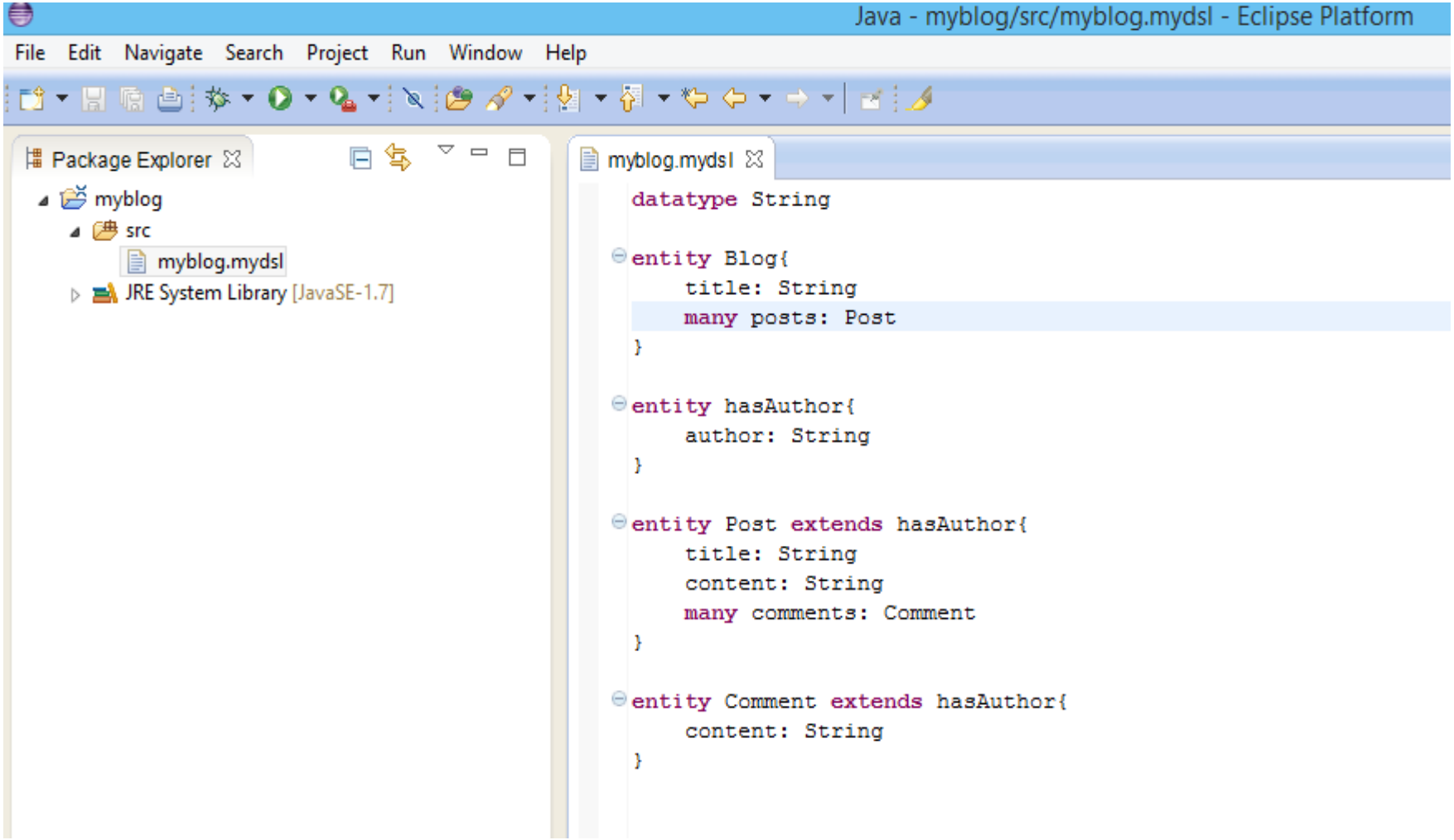
For every EClass, a Java class is generated, installing generalizations and references.

Xtext: Deploy generated artifacts



Launches a new Eclipse instance allowing to edit a DSL project

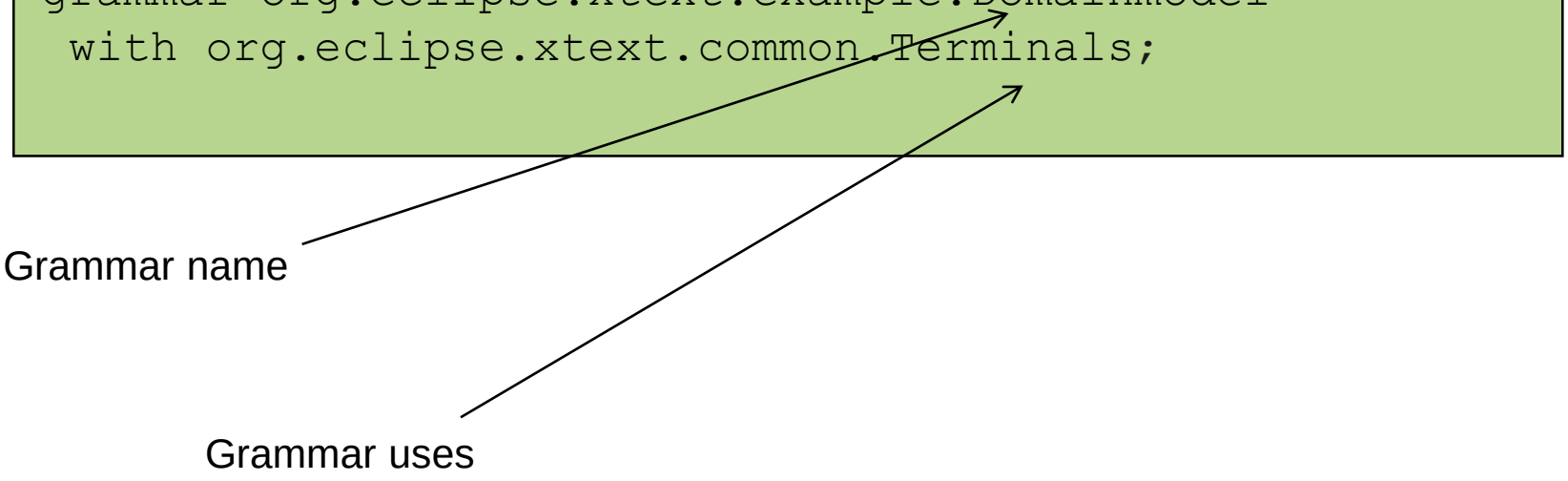
Xtext: Editing the new DSL-Project



Syntax Definition

Grammars in Xtext

```
grammar org.eclipse.xtext.example.Domainmodel
 with org.eclipse.xtext.common.Terminals;
```



Syntax Definition

Grammars in Xtext

```
DomainModel :
  (types+=Type)*;

Type:
  DataType | Entity;

DataType:
  'datatype' name=ID;

Entity:
  'entity' name=ID '{'
  (features+=Feature)*
  '}';

Feature:
 (many?='many')? name=ID ':' type=[Type];
```

Syntax Definition

Grammars in Xtext

```
DomainModel :
  (types+=Type)* ;

Type:
  DataType | Entity;

DataType:
  'datatype' name=ID;

Entity:
  'entity' name=ID '{'
  (features+=Feature)*
  '}';

Feature:
 (many?='many')? name=ID ':' type=[Type];
```

EBNF rules

Syntax Definition

Grammars in Xtext

```
DomainModel :
  (types+=Type)*;

Type:
  DataType | Entity;

DataType:
  'datatype' name=ID;

Entity:
  'entity' name=ID '{'
  (features+=Feature)*
  '}';

Feature:
 (many?='many')? name=ID ':' type=[Type];
```

terminal from `common.Terminals`

Newly defined terminal

Syntax Definition

Grammars in Xtext

```
DomainModel :
  (types+=Type)*;

Type:
  DataType | Entity;

DataType:
  'datatype' name=ID;

Entity:
  'entity' name=ID '{'
  (features+=Feature)*
  '}';

Feature:
 (many?='many')? name=ID ':' type=[Type];
```

Features of rules

→ attributes in the class

- single valued (String by default)
- list valued ([1..*])
- Boolean valued

Syntax Definition

Grammars in Xtext

```
DomainModel :
  (types+=Type)*;

Type:
  DataType | Entity;

DataType:
  'datatype' name=ID;

Entity:
  'entity' name=ID '{'
  (features+=Feature)*
  '}';

Feature:
 (many?='many')? name=ID ':' type=[Type];
```

Cross referencere

- link to some existing referenced object
- no rule call
- resolved by linking
- advanced variant
  e.g., [Type|Token]

Syntax Definition

Terminal rules

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

From `common.Terminals`

```
terminal INT returns ecore::EInt :
  ('0'..'9')+;
```

Return type
(default: `ecore::EString`)

→ An `EInt`-Object will be created during the parsing process.

Syntax Definition

Return types in parser rules (Object creation during the parsing)

```
MyRule returns TypeA :
  'A' name=ID |
  MyOtherRule;

MyOtherRule returns TypeB :  // B subtypes A
  'B' name = ID;
```

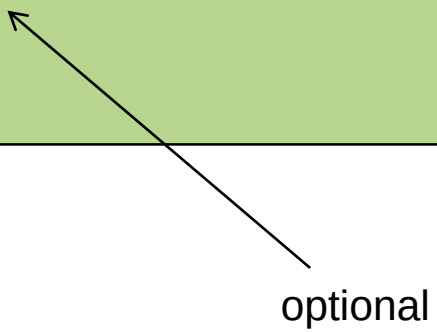
Rule names and type names are identical by default, but they can be different using `returns´.

Causes the rule name class to be subclass of its return-type class.

Syntax Definition

Datatype Rules

```
QualifiedName returns ecore::EString :
  ID ('.' ID)*
;
```

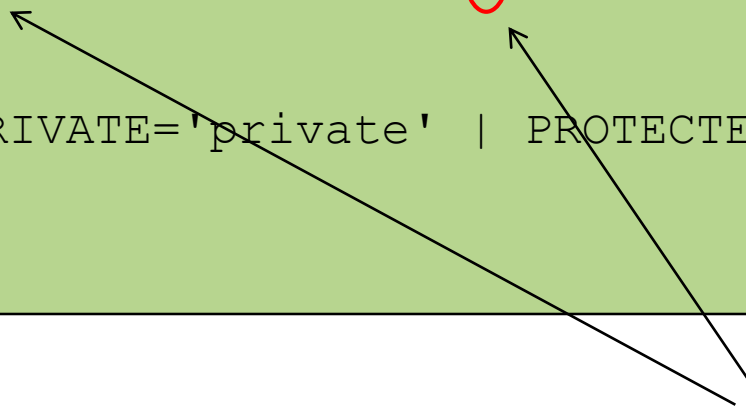


- Rules return DataTypes (EDataType) rather than classes
- Requirement
  - No calls of parse rules
  - No assignments (to features), no actions
  - Standard return type: `EString`

Syntax Definition

(Unordered Groups / Enums)

```
Modifier:
static?='static'? & final?='final'? & visibility=Visibility;

enum Visibility:
PUBLIC='public' | PRIVATE='private' | PROTECTED='protected';
```



Any order, but exactly one occurrence of each

Syntax Definition

Simple Actions

```
MyRule returns TypeA :
  'A' name=ID |
  MyOtherRule;

MyOtherRule returns TypeB :  // B subtypes A
  'B' name = ID;
```

```
MyRule returns TypeA :
  'A' name=ID |
  'B' {Type B} name=ID;
```

Creation of B-object enforced

Syntax Definition

Unassigned rule calls

```
AbstractToken :
  ( TokenA |
    TokenB |
    TokenC ) (cardinality=('?'|'+'|'*'))?
;
```

- Rule calls without assignments to features
- Types are passed through

Syntax Definition

AST construction via assigned features/actions

```
Addition :
  Multiplication ('+'  Multiplication)*;

Multiplication :
   Primary ('*' Primary)*;

Primary :
    '(' Addition ')' |
    NumberLiteral;

NumberLiteral:
     INT;
```

All unassigned rules

→ types inferred by Xtext (`EString` by default)

→ no AST nodes

Syntax Definition

AST construction via assigned actions (naive approach)

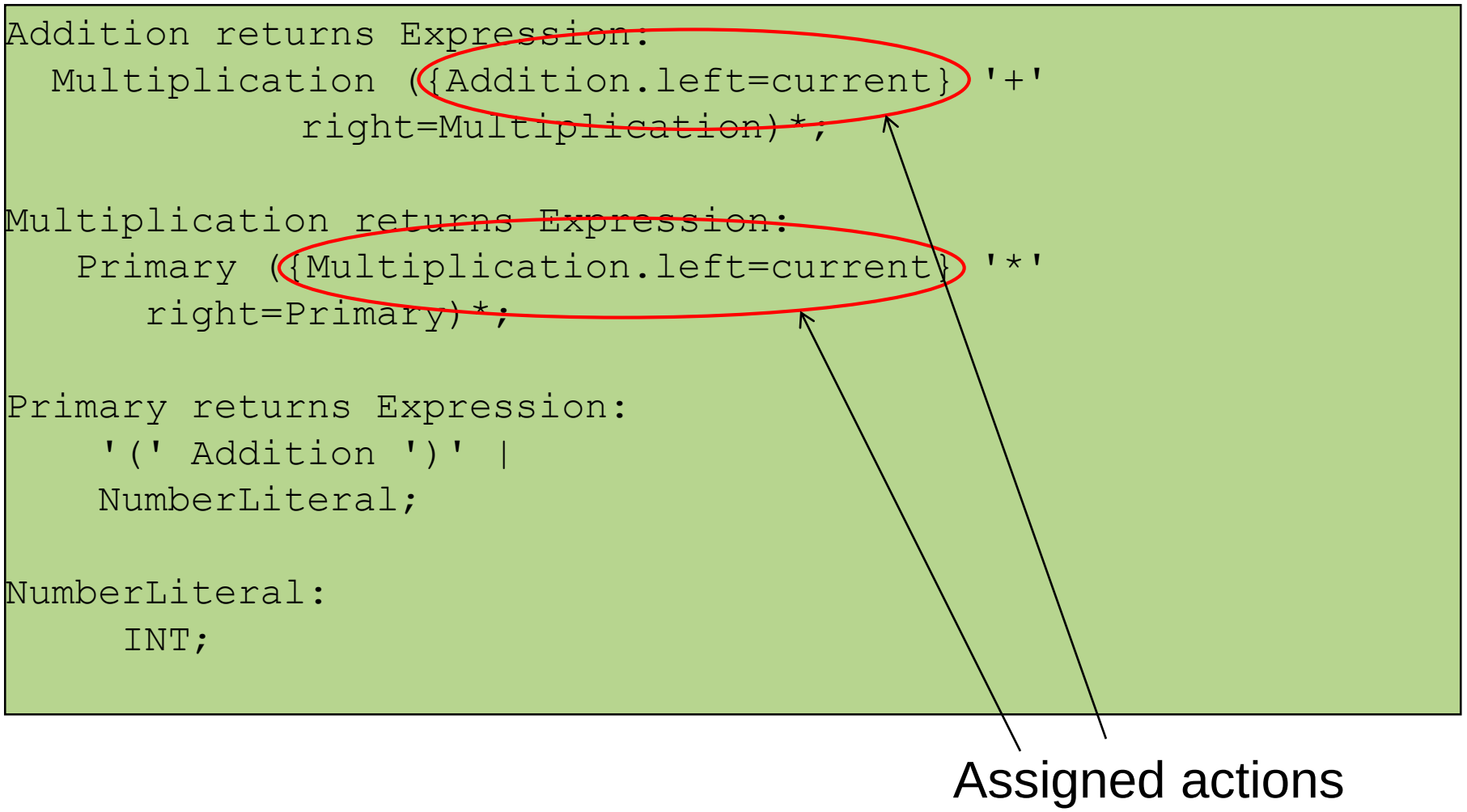
```
Addition returns Expression:
  left=Multiplication ('+' right=Multiplication)*;

Multiplication returns Expression:
   left=Primary ('*' right=Primary)*;

Primary returns Expression:
    '(' Addition ')' |
    NumberLiteral;

NumberLiteral:
     INT;
```

Unnecessary AST nodes!

Syntax Definition

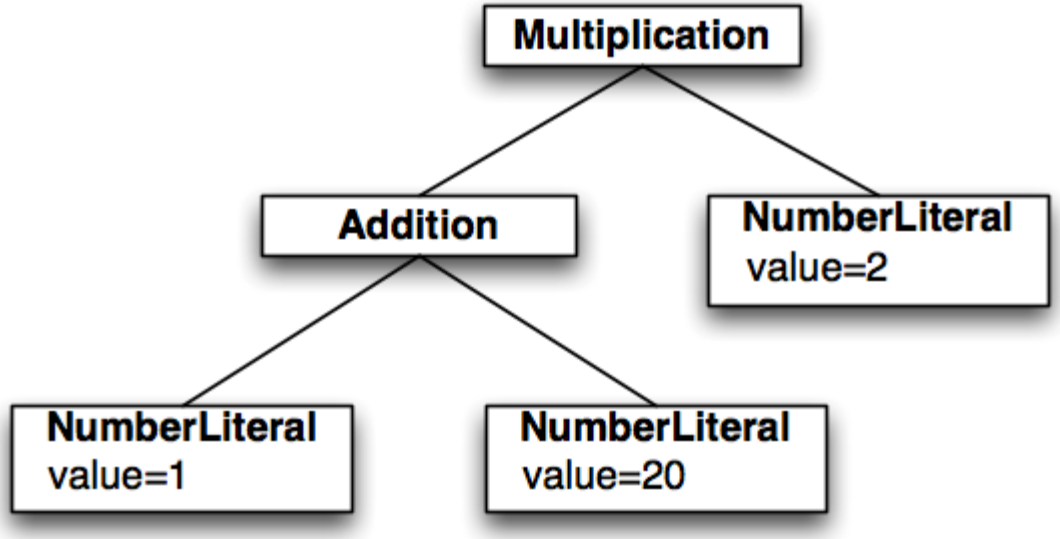
AST construction via assigned actions

```
Addition returns Expression:
  Multiplication ({Addition.left=current} '+'
            right=Multiplication)*;

Multiplication returns Expression:
   Primary ({Multiplication.left=current} '*'
      right=Primary)*;

Primary returns Expression:
    '(' Addition ')' |
    NumberLiteral;

NumberLiteral:
     INT;
```



Syntax Definition

AST tree for expression (1+20)*2



Xtext

So far …

- overview of the Xtext project
- meta models in terms of grammars
  - EBNF facilities + Ecore
  - ANTLR 3 used as parser generator
  - cross-links to capture basic bindings
- generated artifacts
  - deployed as Ecplise plugins
  - parser
  - syntax oriented editor (…)

Xtext

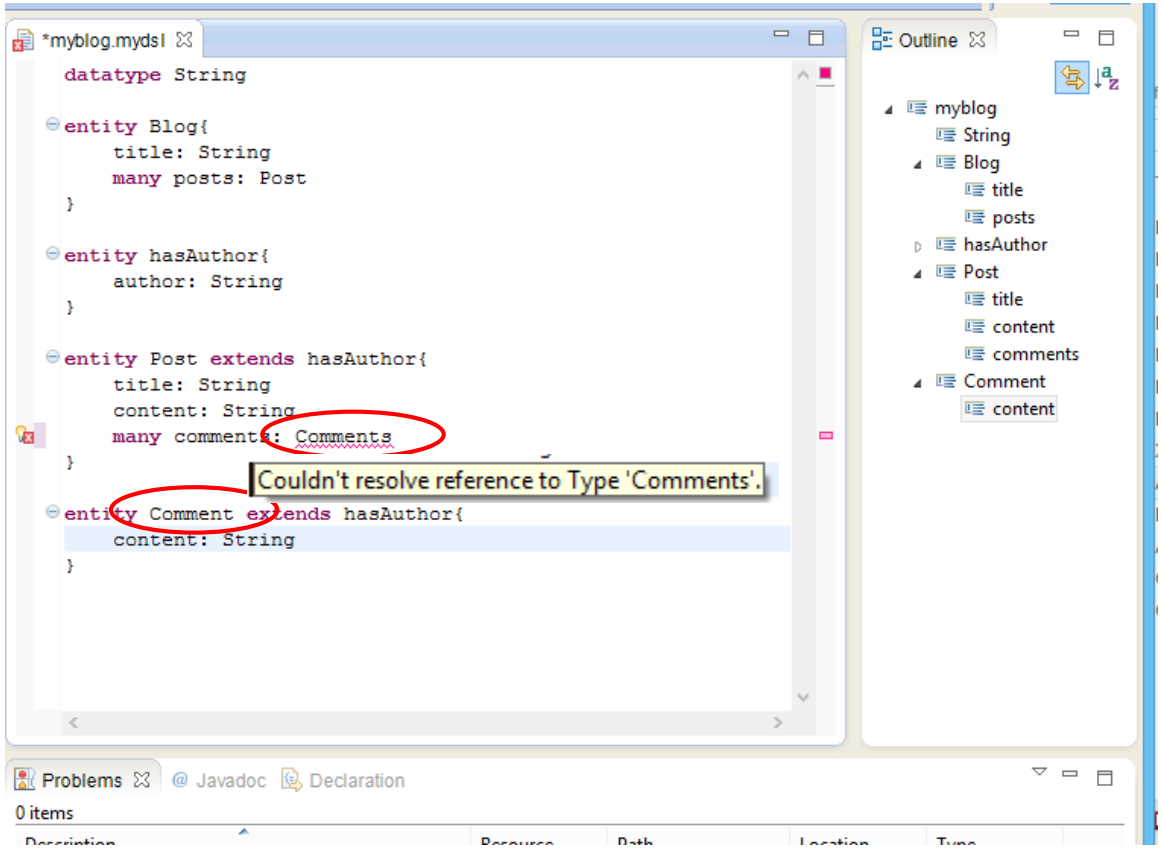
Now …

- Adding static validations
  - via cross-links
  - preconfigured
  - customized
- Adding a code generator
  - template based
  - refinement based approach

    stub → beans → executable code
- Using **Xtend** (a Java-like declarative language)

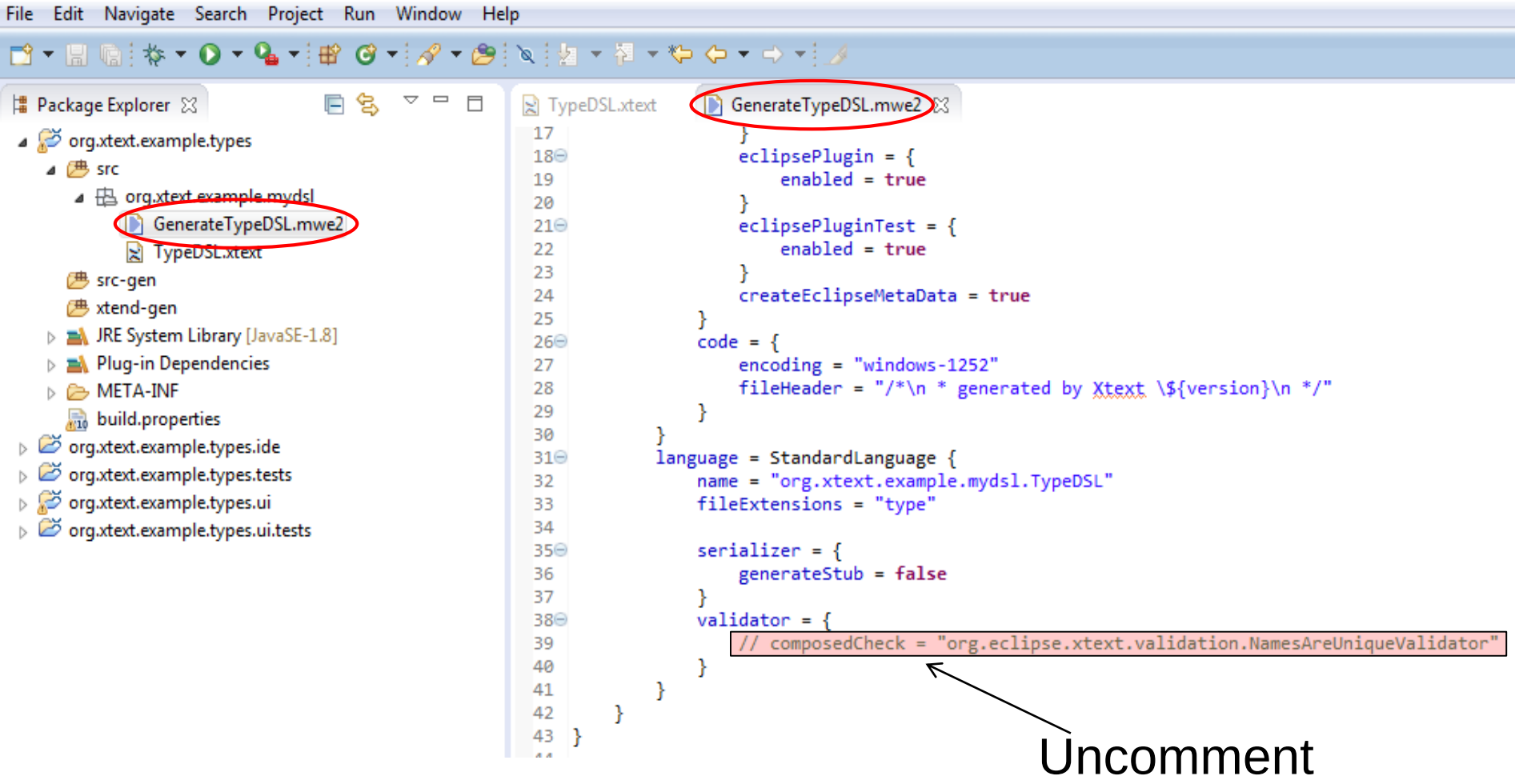
Static Semantics

Validating bindings via cross-links



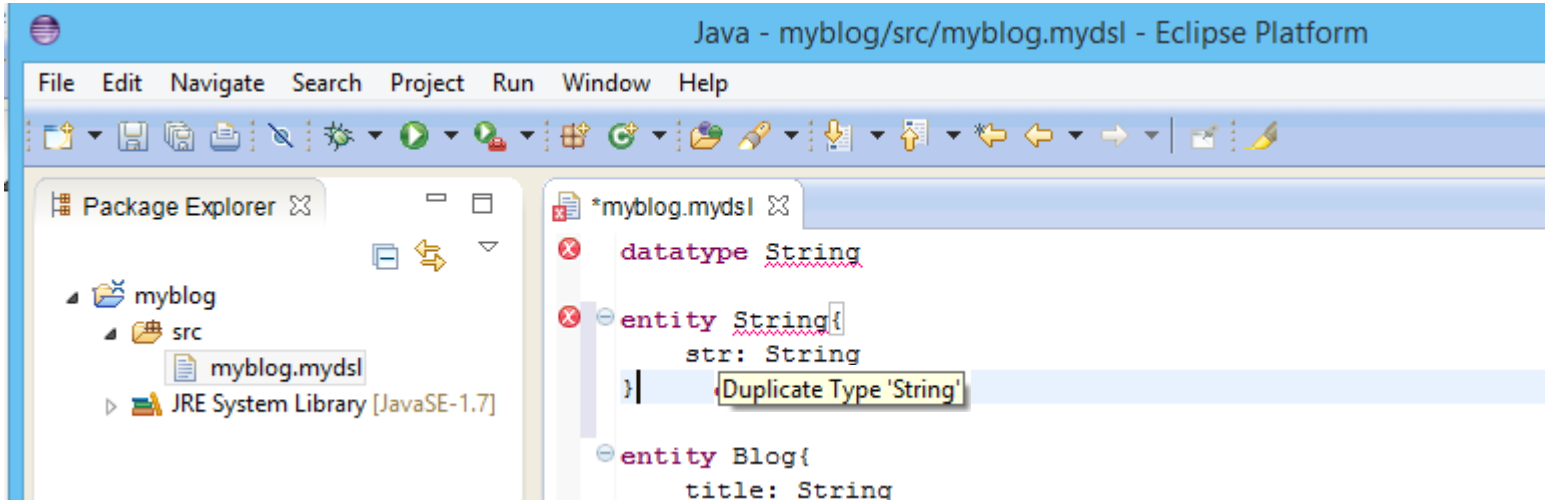
Static Semantics

Preconfigured validations



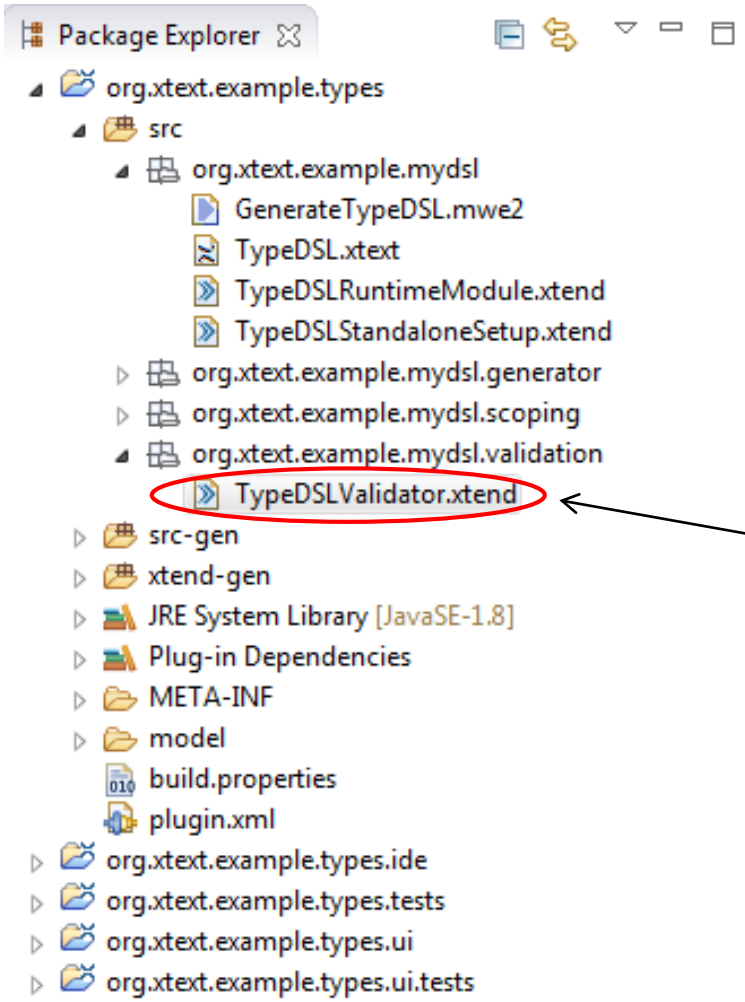
Static Semantics

Preconfigured validations



Static Semantics

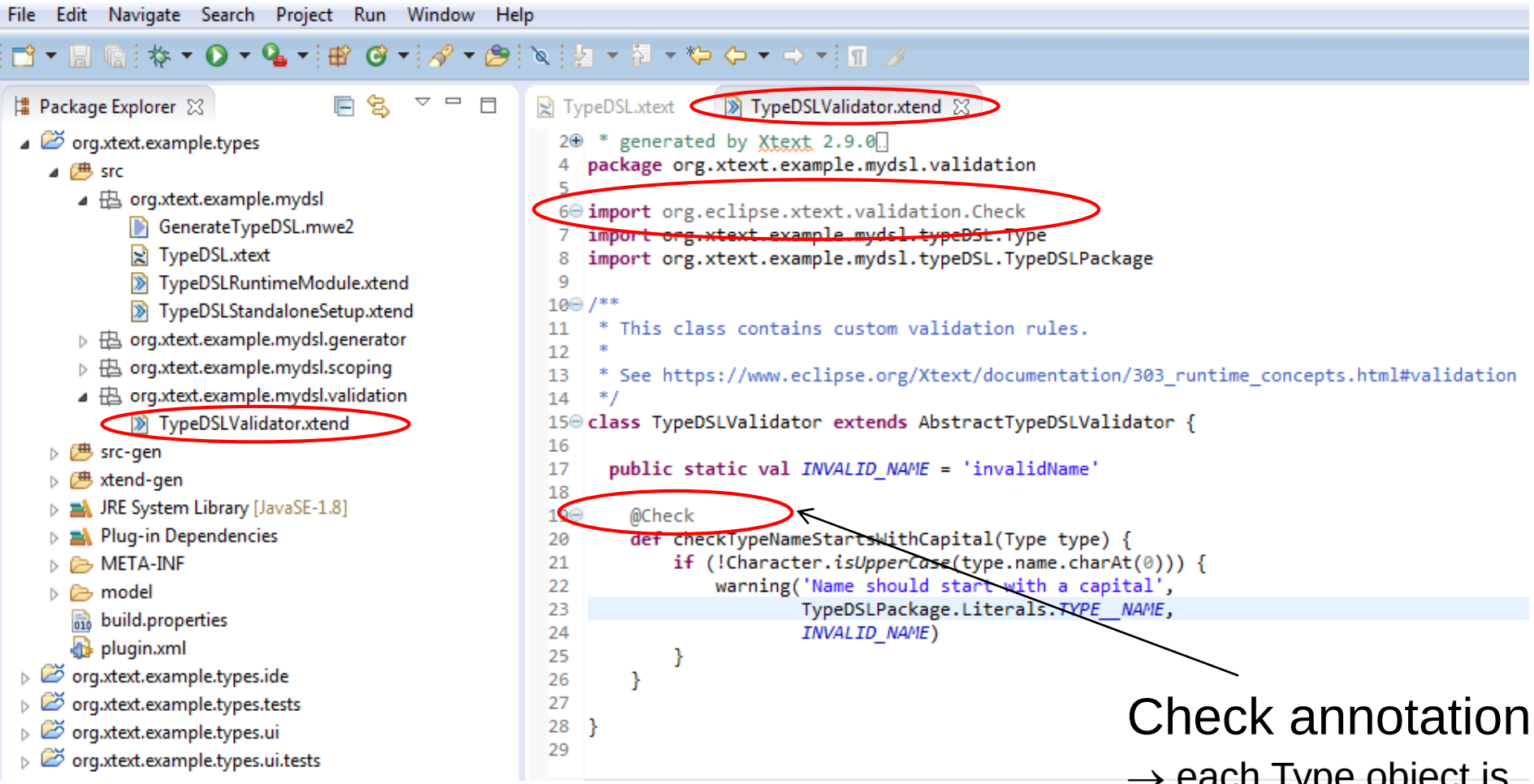
Customized validations



Edit source file

Static Semantics

Customized validations

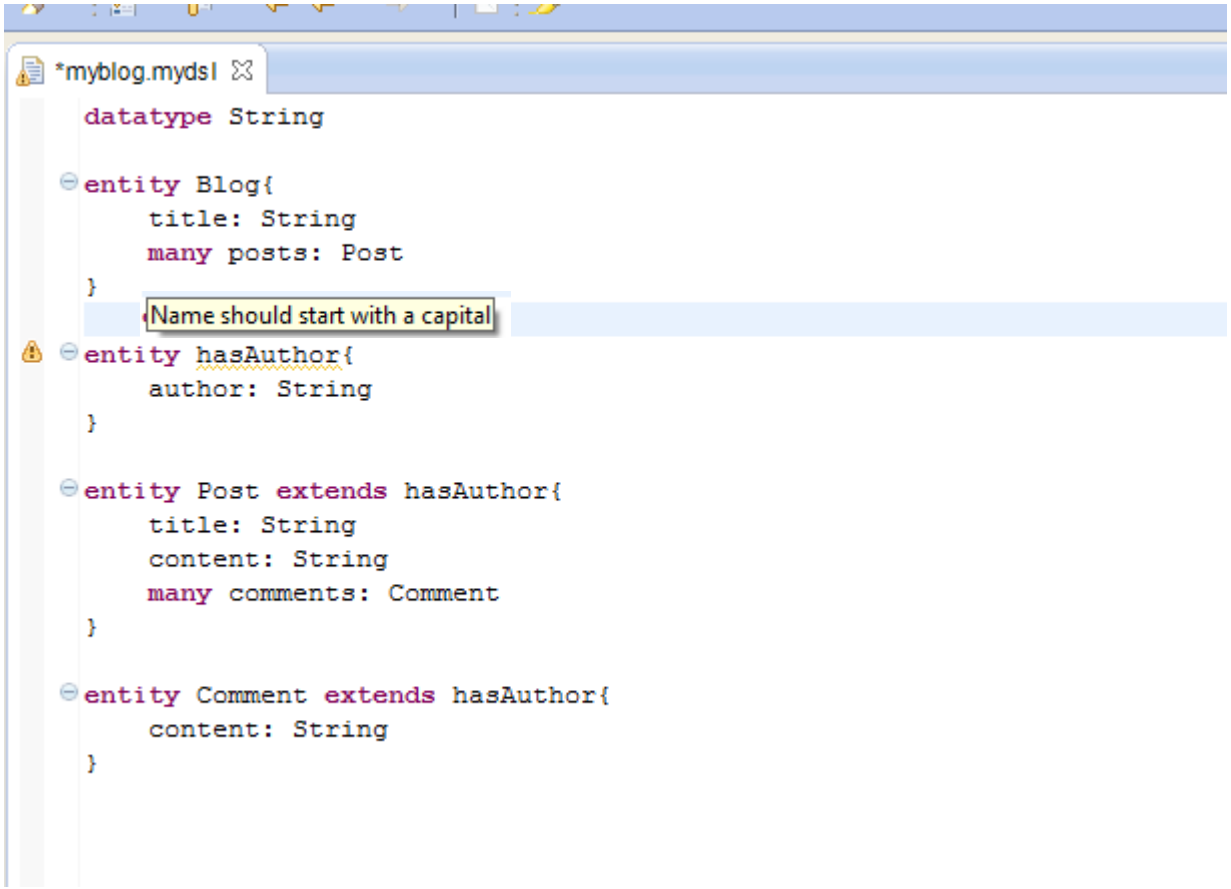


Check annotation

→ each Type object is checked accordingly

Static Semantics

Customized validations



Static Semantics

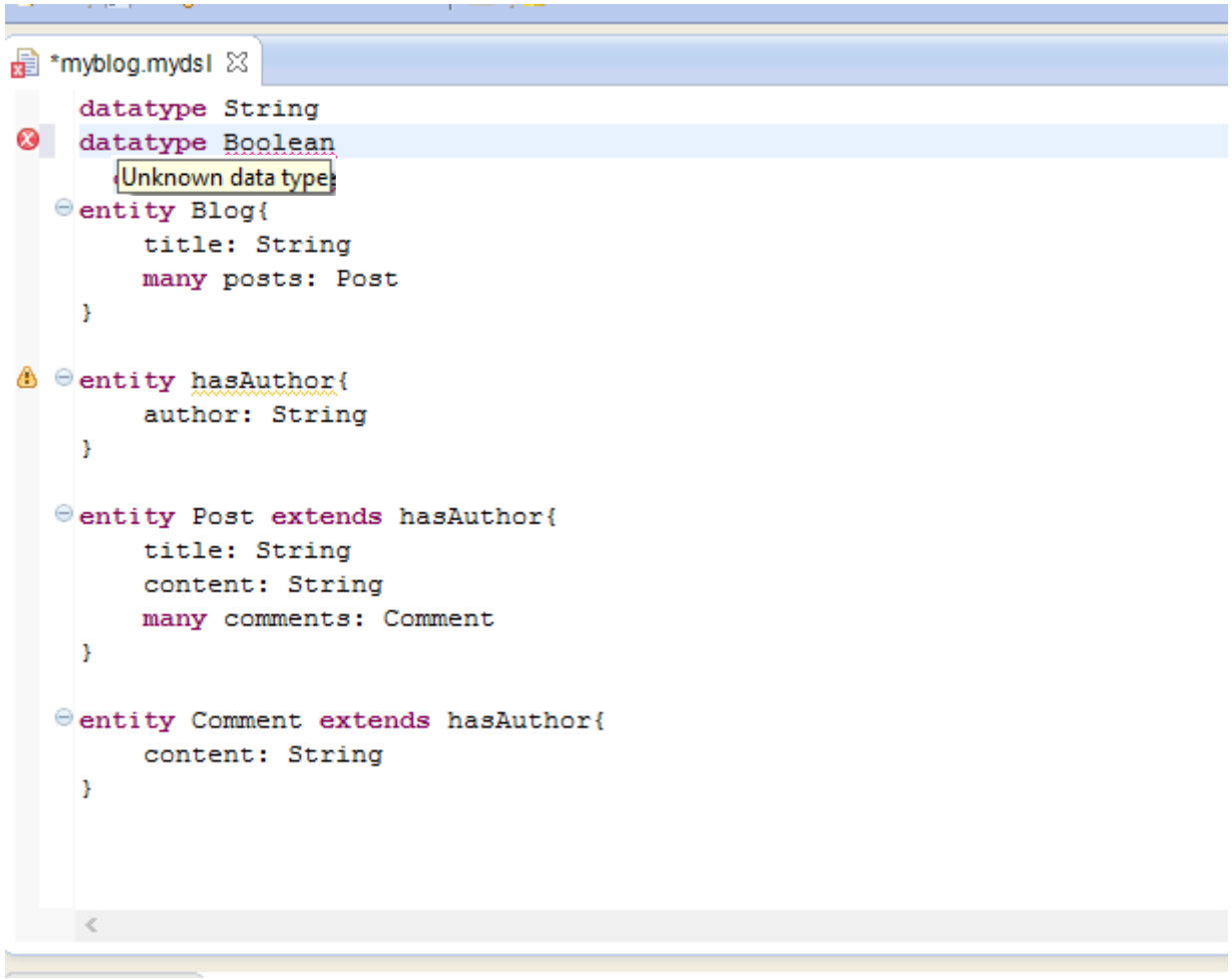
Customized validations

TypeDSL.xtext | TypeDSLValidator.xtend

```
 2⊕ * generated by Xtext 2.9.0
 4 package org.xtext.example.mydsl.validation
 5
 6⊖ import org.eclipse.xtext.validation.Check
 7 import org.xtext.example.mydsl.typeDSL.DataType
 8 import org.xtext.example.mydsl.typeDSL.Type
 9 import org.xtext.example.mydsl.typeDSL.TypeDSLPackage
10
11⊖ /**
12  * This class contains custom validation rules.
13  *
14  * See https://www.eclipse.org/Xtext/documentation/303_runtime_concepts.html#validation
15  */
16⊖ class TypeDSLValidator extends AbstractTypeDSLValidator {
17
18   public static val INVALID_NAME = 'invalidName'
19
20⊖    @Check
21     def checkTypeNameStartsWithCapital(Type type) {
22         if (!Character.isUpperCase(type.name.charAt(0))) {
23             warning('Name should start with a capital',
24                     TypeDSLPackage.Literals.TYPE__NAME,
25                     INVALID_NAME)
26         }
27     }
28
29⊖    @Check
30     def checkDataTypes(DataType type) {
31         if (!(type.getName().equals("String") || type.getName().equals("Integer") ))
32           error("Unknown Datatype", TypeDSLPackage.Literals.TYPE__NAME)
33     }
34 }
35
```

Static Semantics

Customized validations



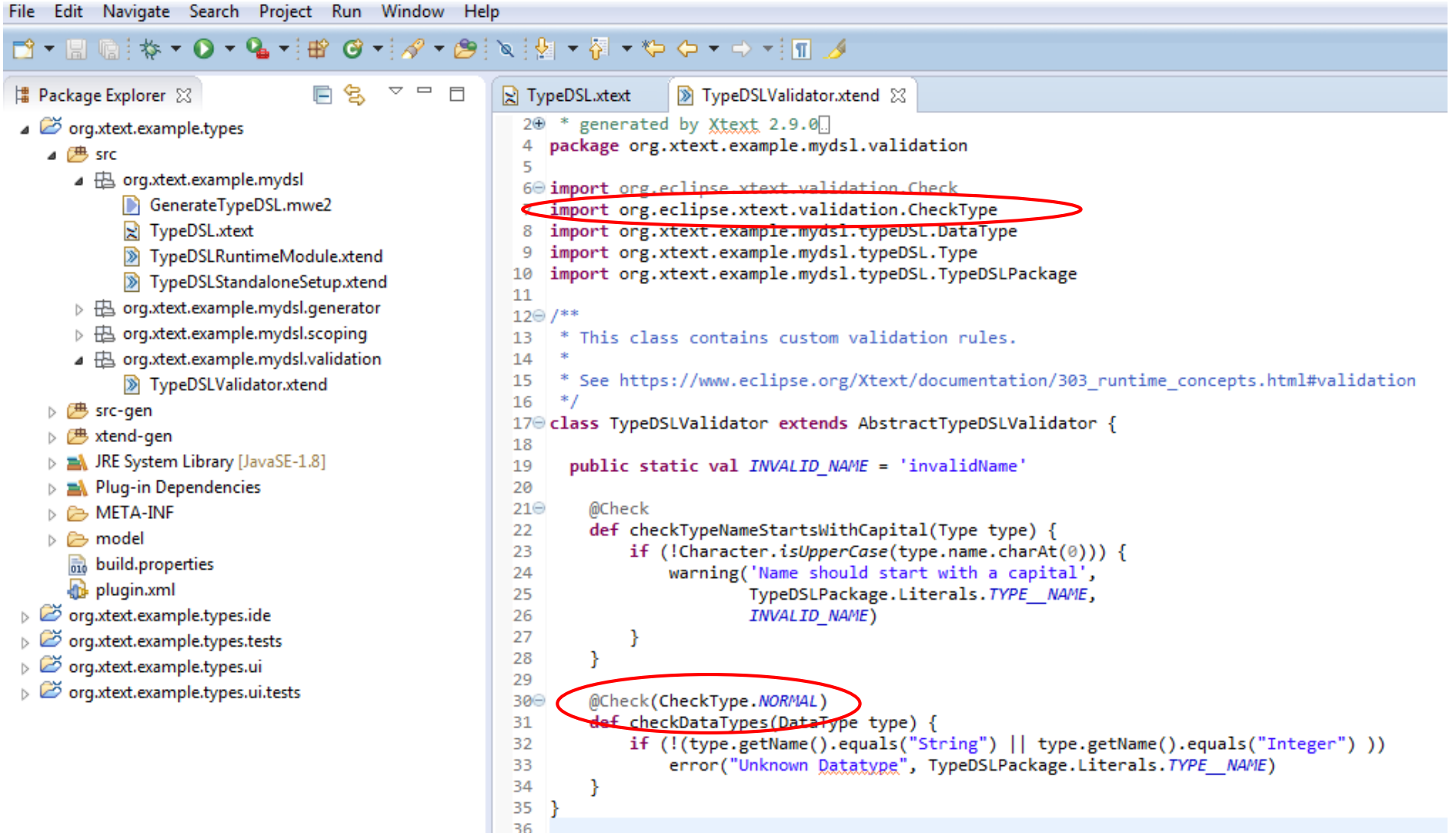
Static Semantics

Customized validations

- Trigger of validation checks
  determined by (optional) parameter of Check annotation
  - FAST immediately during editing
  - NORMAL at every save
  - EXPENSIVE to be explicitely generated for the DSL
- Default is FAST

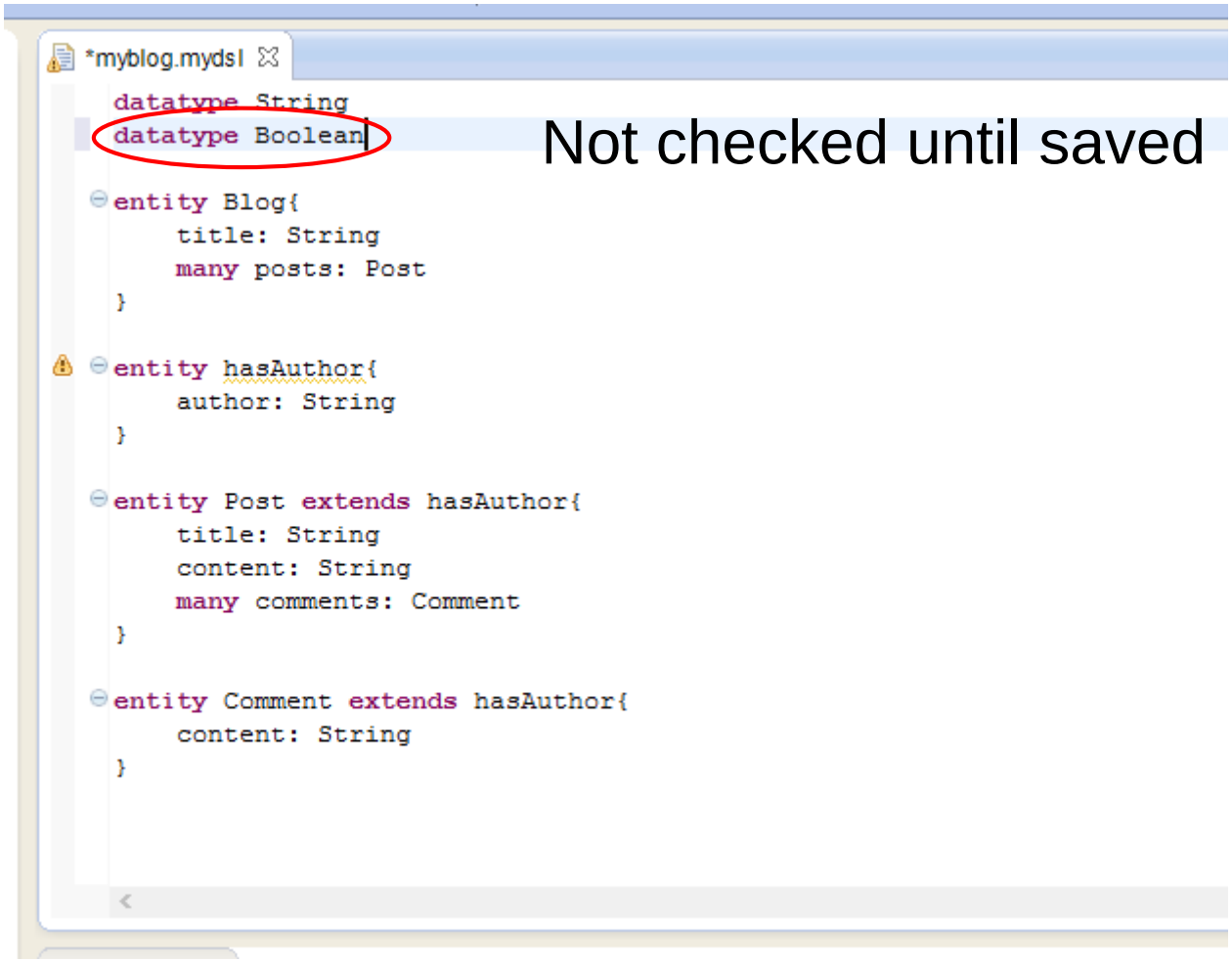
Static Semantics

Customized validations



Static Semantics

Customized validations

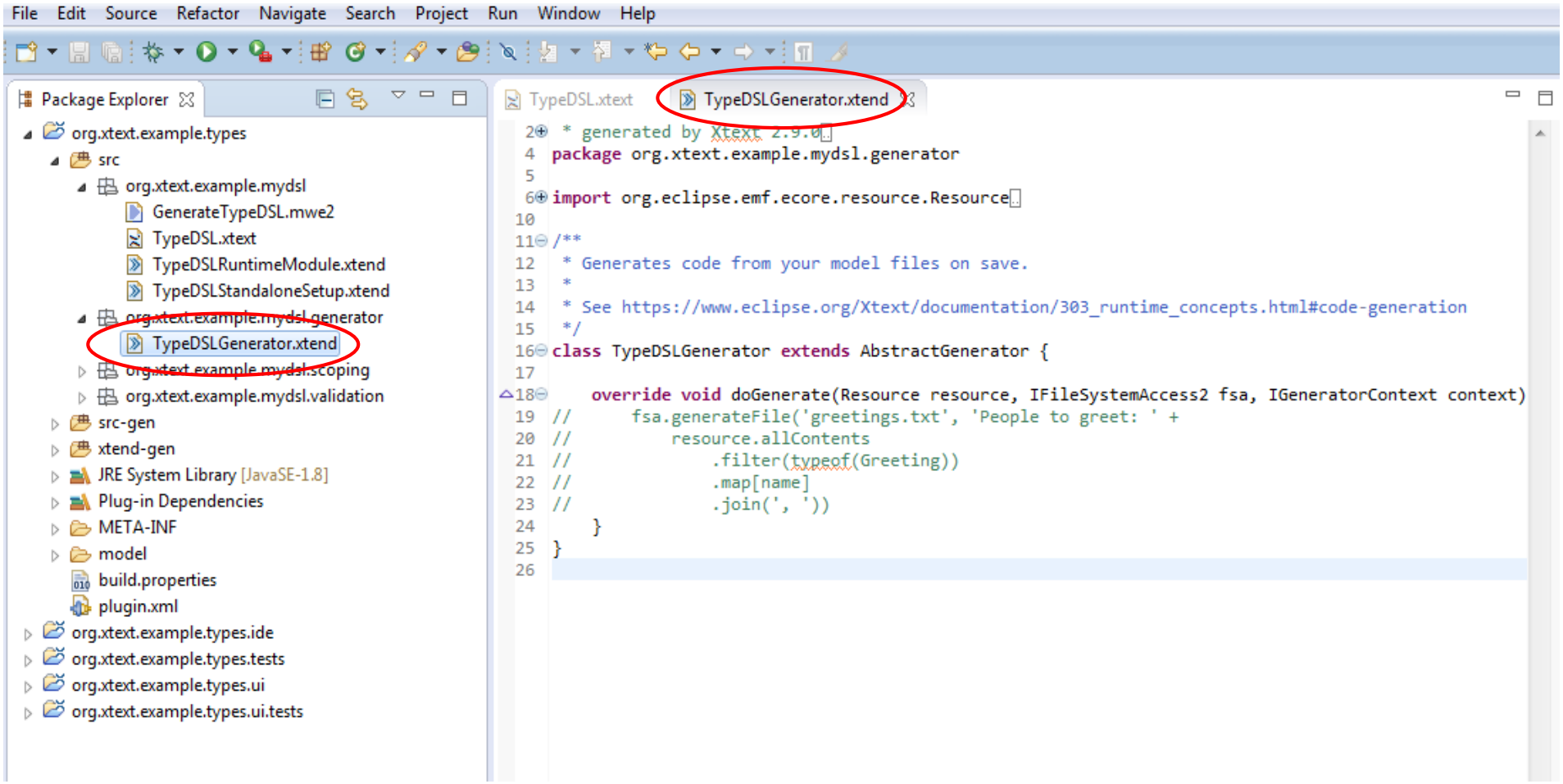


Code Generation

Code generation with Xtend

- Xtend is a Java-like language for specifying code generators
  - sets upon an Ecore model of the AST
  - integrates concepts of declarative programming
  - powerful type inference
  - own template language
  - ..
- Here generation of Java beans for our blog example
- For more information $\rightarrow$ documentation!!!

Code Generation with Xtend



Code generation stub to be modified

Code Generation with Xtend

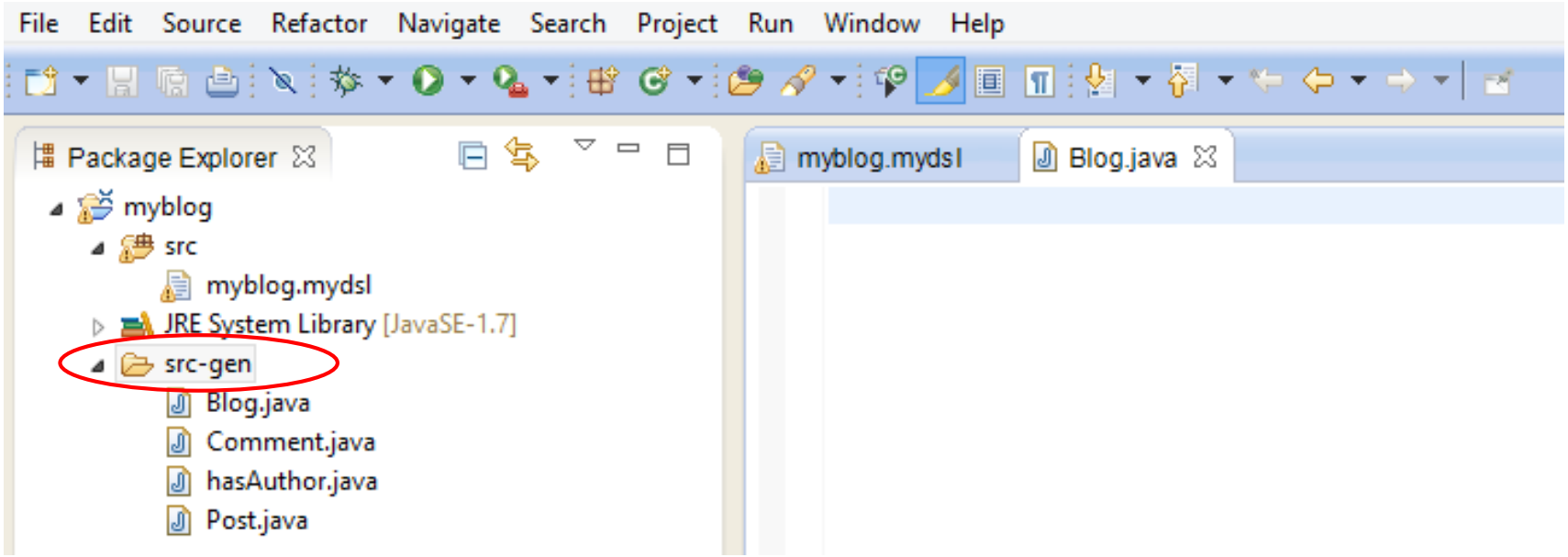
TypeDSL.xtext | TypeDSLGenerator.xtend

```
 2  * generated by Xtext 2.9.0
 4  package org.xtext.example.mydsl.generator
 5
 6  import com.google.inject.Inject
 7  import org.eclipse.emf.ecore.resource.Resource
 8  import org.eclipse.xtext.generator.AbstractGenerator
 9  import org.eclipse.xtext.generator.IFileSystemAccess2
10  import org.eclipse.xtext.generator.IGeneratorContext
11  import org.eclipse.xtext.naming.IQualifiedNameProvider
12  import org.xtext.example.mydsl.typeDSL.Entity
13
14  /**
15   * Generates code from your model files on save.
16   *
17   * See https://www.eclipse.org/Xtext/documentation/303_runtime_concepts.html#code-generation
18   */
19  class TypeDSLGenerator extends AbstractGenerator {
20
21      @Inject extension IQualifiedNameProvider
22
23      override void doGenerate(Resource resource, IFileSystemAccess2 fsa, IGeneratorContext context)
24          for(e: resource.allContents.toIterable.filter(Entity))
25              fsa.generateFile(e.fullyQualifiedName.toString("/") + ".java",
26              '''''' // empty template
27              )
28      }
29  }
30
```

First refinement:
generating empty java classes of models

Code Generation

Give it a try!



Generate Xtext artifacts, launch Eclipse application,
create a DSL file in a Java project. *Now:*
create src-gen folder → classes are automatically generated

Code Generation

Adding attributes, getter and setter

```
override void doGenerate(Resource resource, IFileSystemAccess2 fsa, IGeneratorContext context) {
    for(e: resource.allContents.toIterable.filter(Entity))
        fsa.generateFile(e.fullyQualifiedName.toString("/") + ".java",
        e.compile
        )
}

def compile(Entity e) '''
    public class «e.name»
    «IF e.superType != null» extends «e.superType.fullyQualifiedName» «ENDIF» {
        «FOR f:e.features»
            «f.compile»
        «ENDFOR»
    }
'''

def compile(Feature f) '''
    private «f.type.fullyQualifiedName» «f.name»;

    public «f.type.fullyQualifiedName» get«f.name.toFirstUpper»() {
        return «f.name»;
    }

    private void set«f.name.toFirstUpper»(«f.type.fullyQualifiedName» «f.name») {
        this.«f.name» = «f.name»;
    }
'''
```

Second refinement: generating java beans of models

Code Generation

Code generation with Xtend