



MDSD in Practice

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>

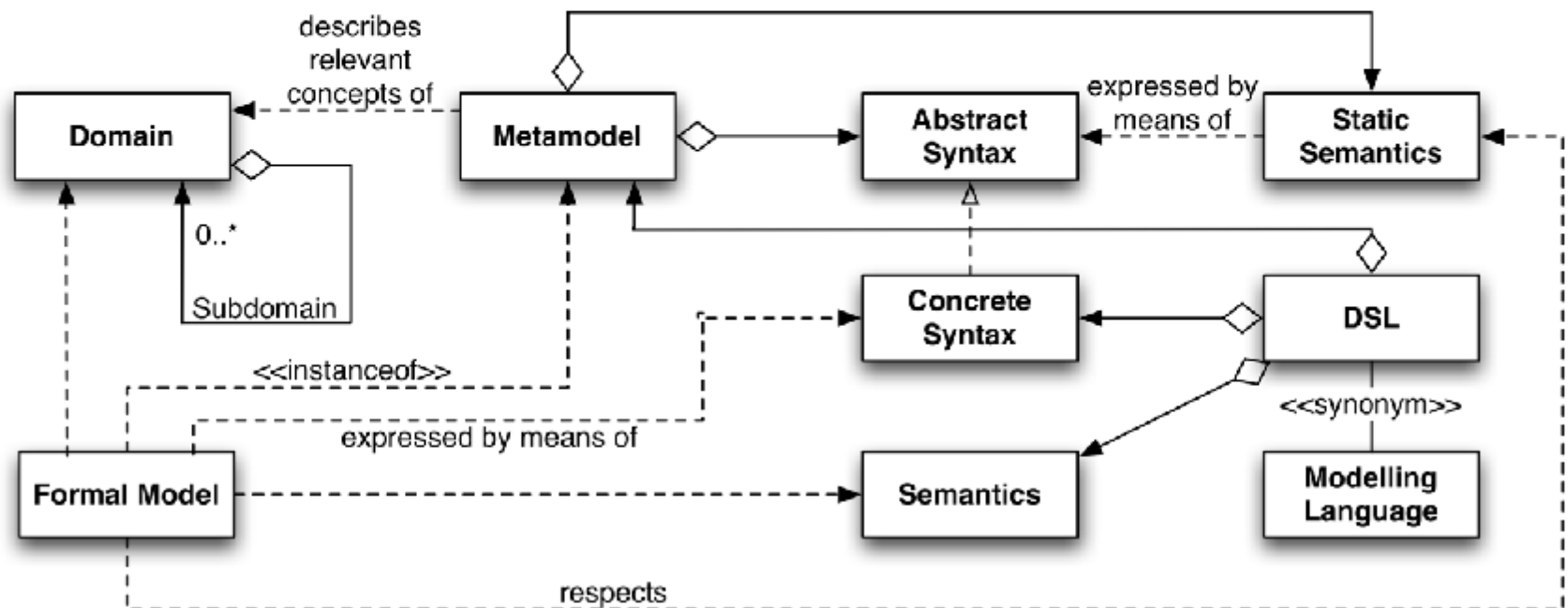
Xtext

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

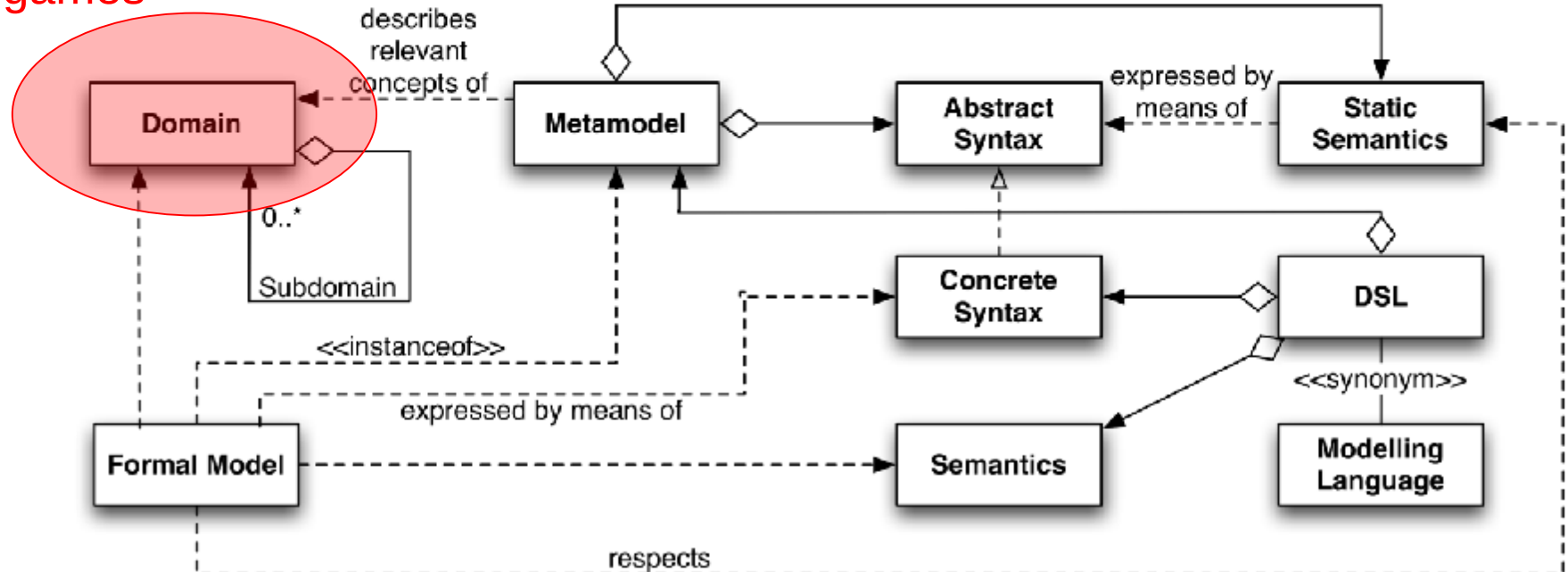


Source: [MDSD], S. 28

MDSD recap

MDSD: Concepts & Notions in our project

Basic arcade
games

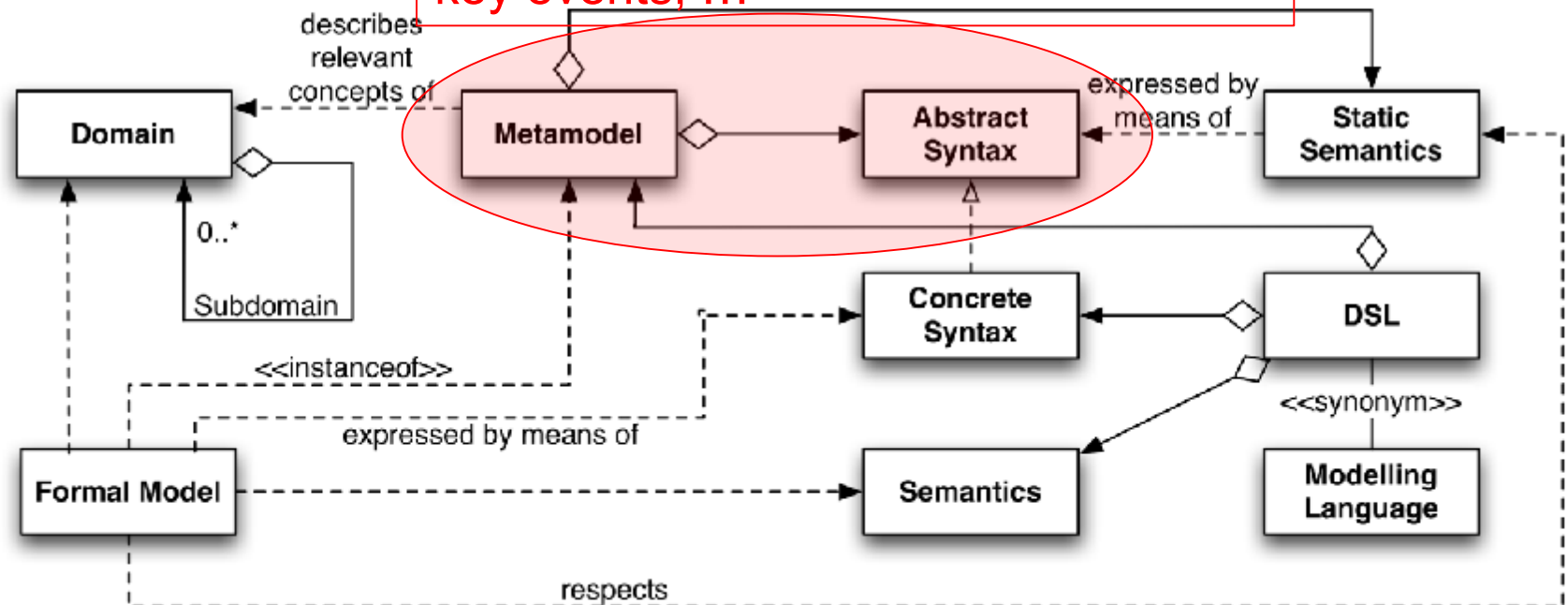


Source: [MDSD], S. 28

MDSD recap

MDSD: Concepts & Notions in our project

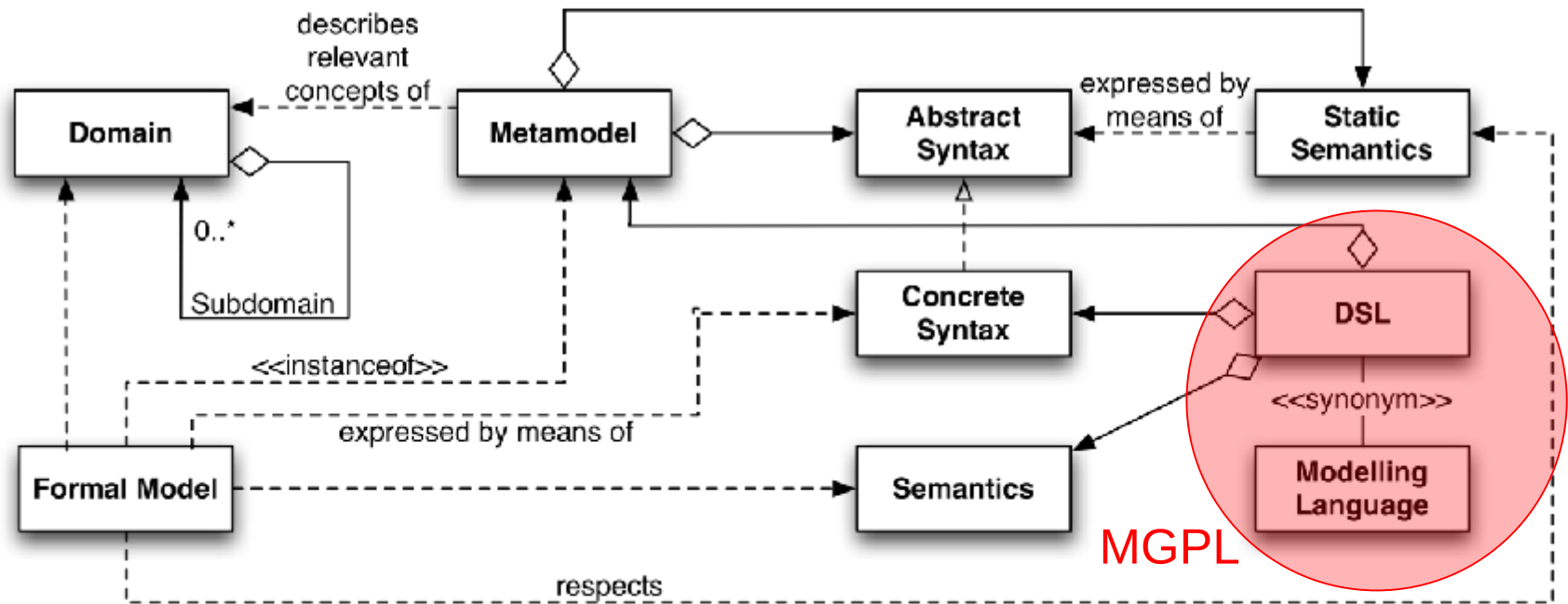
canvas, game objects, movement,
key events, ...



Source: [MDSD], S. 28

MDSD recap

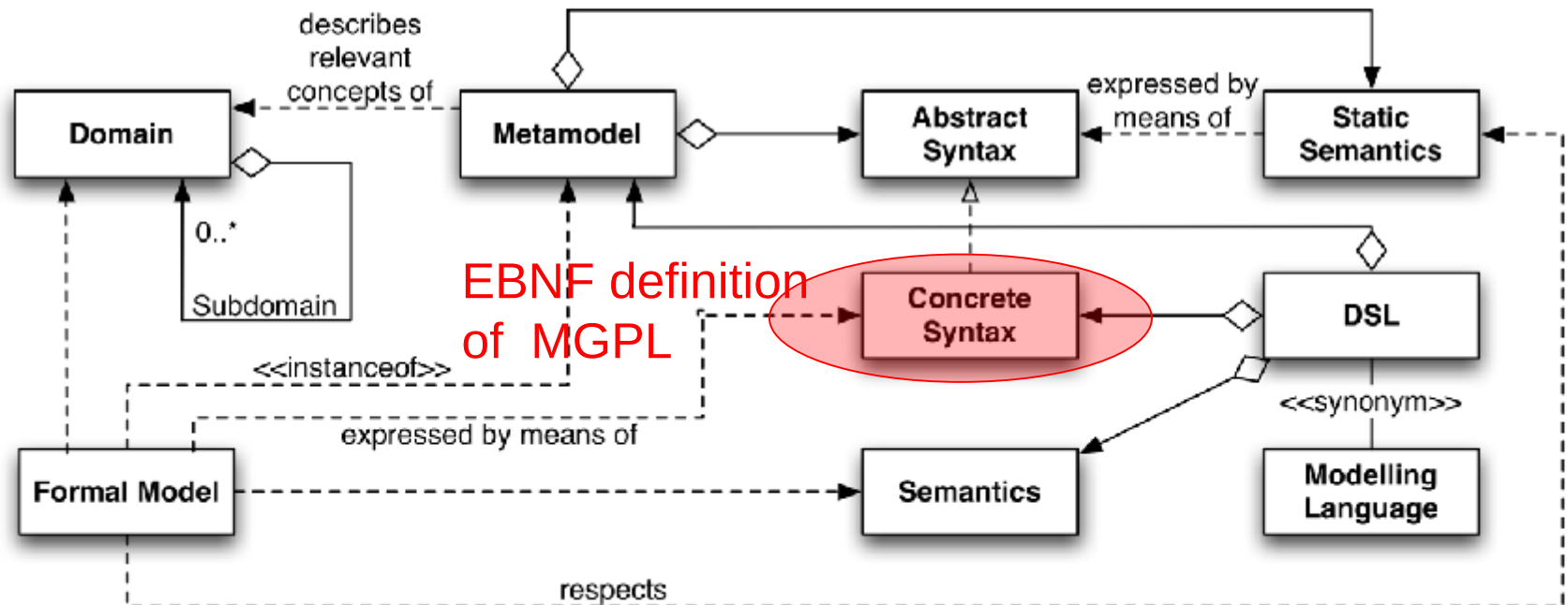
MDSD: Concepts & Notions in our project



Source: [MDSD], S. 28

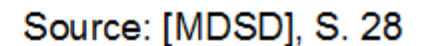
MDSD recap

MDSD: Concepts & Notions in our project



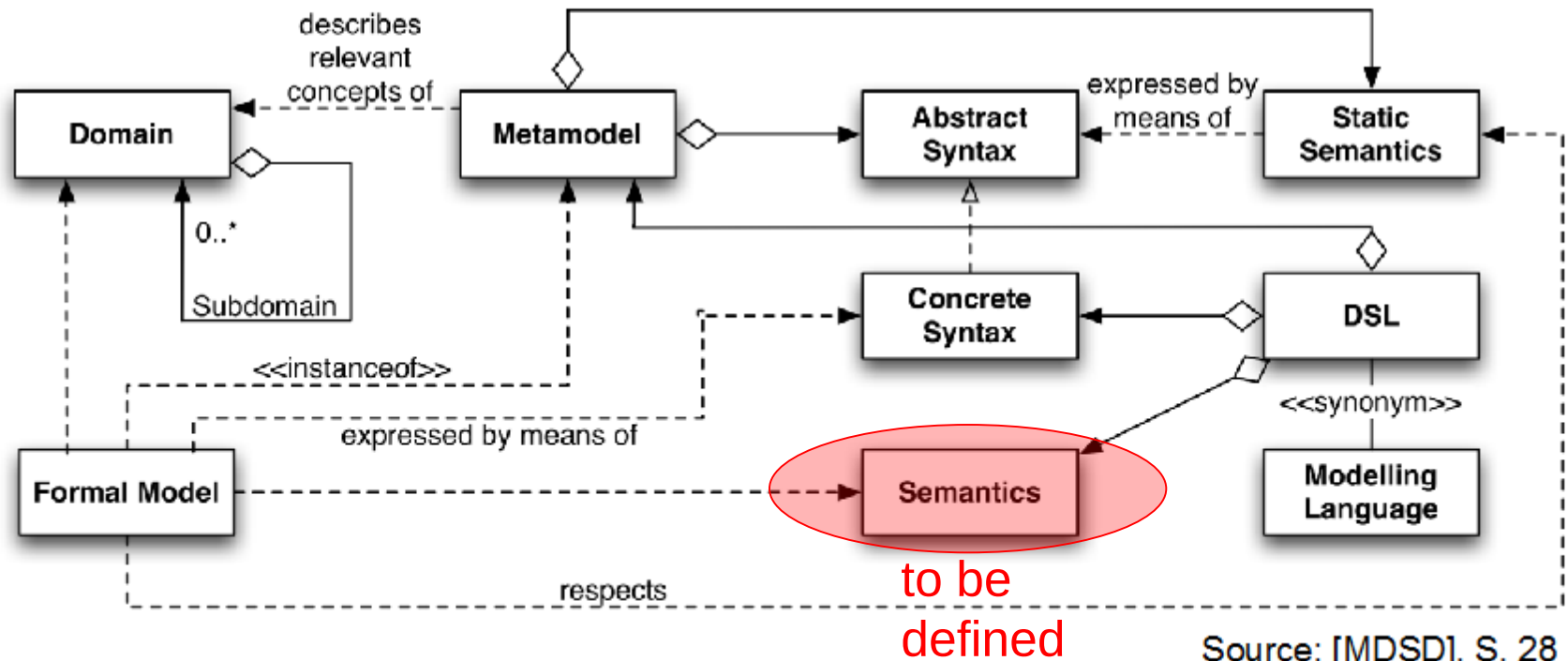
Source: [MDSD], S. 28

MDSD: Concepts & Notions in our project



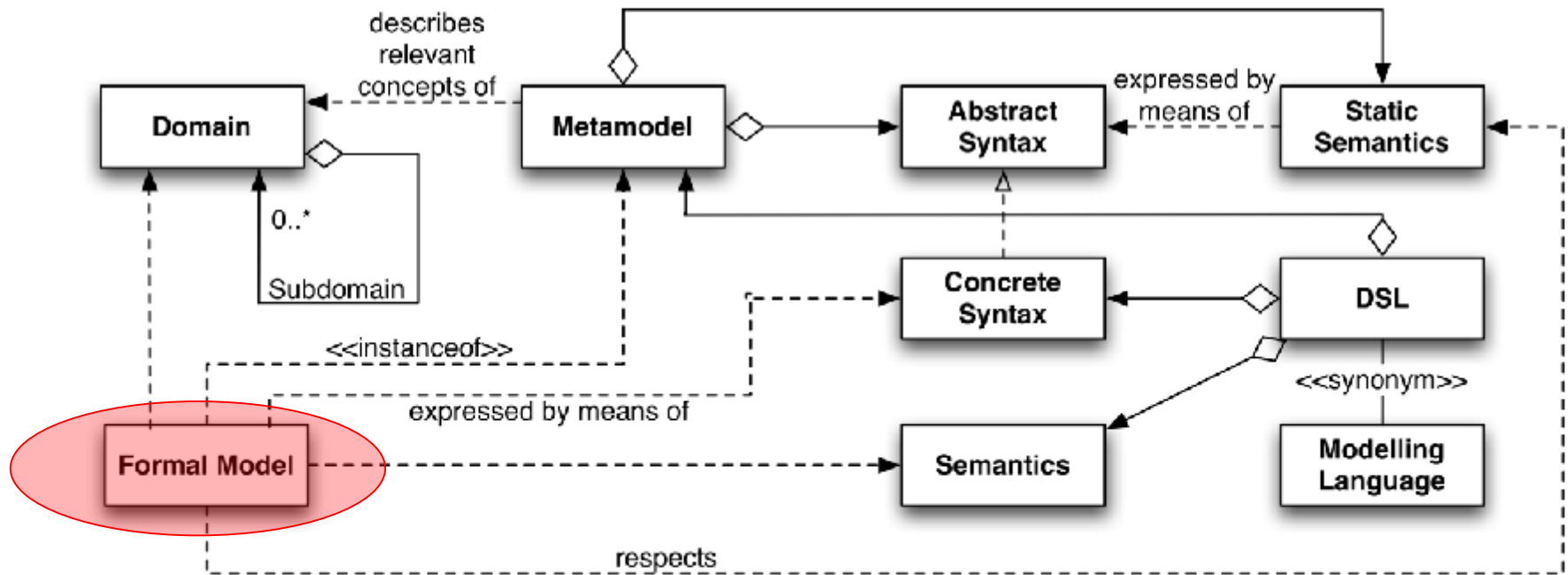
MDSD recap

MDSD: Concepts & Notions in our project



MDSD recap

MDSD: Concepts & Notions in our project

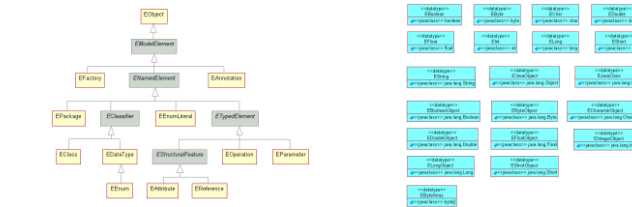


Pong,
Invaders,...

Source: [MDSD], S. 28

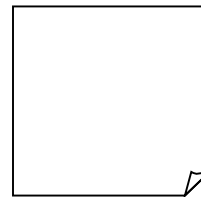
Model Hierarchy in our project

based on Ecore
+ EBNF facilities



M3: Metametamodel

MGPL grammar



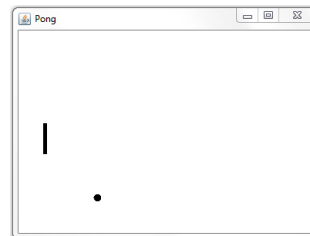
M2: Metamodel

MGPL program

game Pong(..)

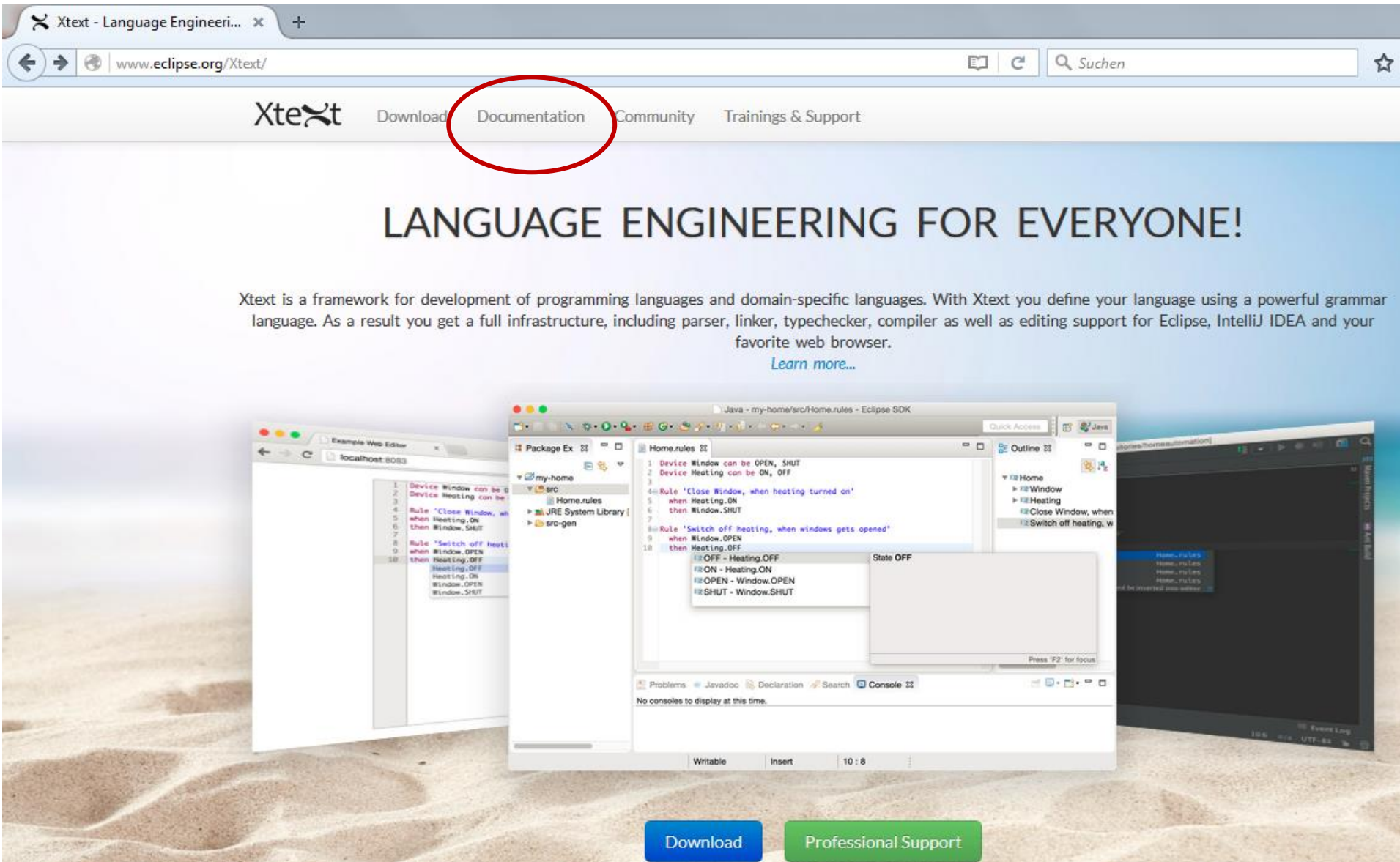
M1: Model

Java application



M0: Model instance

The tool – Xtext



The image is a composite graphic for the Xtext tool. At the top, a browser window shows the Xtext website (www.eclipse.org/Xtext/) with the 'Documentation' link circled in red. Below the website header, the text 'LANGUAGE ENGINEERING FOR EVERYONE!' is displayed. A paragraph describes Xtext as a framework for developing programming and domain-specific languages, listing its components (parser, linker, typechecker, compiler) and supported editors (Eclipse, IntelliJ IDEA). A 'Learn more...' link is provided. The bottom half of the image features a collage of screenshots from the Xtext IDE. One screenshot shows a web editor at localhost:8083 displaying a domain-specific language for a home automation system. Another screenshot shows the Eclipse IDE with a project named 'my-home' containing a 'Home.rules' file with similar domain-specific language rules. A third screenshot shows a console window with an event log. At the bottom, there are two buttons: 'Download' and 'Professional Support'.

Xtext - Language Engineeri... x +

www.eclipse.org/Xtext/

Suchen

Xtext Download Documentation Community Trainings & Support

LANGUAGE ENGINEERING FOR EVERYONE!

Xtext is a framework for development of programming languages and domain-specific languages. With Xtext you define your language using a powerful grammar language. As a result you get a full infrastructure, including parser, linker, typechecker, compiler as well as editing support for Eclipse, IntelliJ IDEA and your favorite web browser.

[Learn more...](#)

Example Web Editor
localhost:8083

```
1 Device Window can be OPEN, SHUT
2 Device Heating can be ON, OFF
3
4 Rule 'Close Window, when heating turned on'
5 when Heating.ON
6 then Window.SHUT
7
8 Rule 'Switch off heating, when windows gets opened'
9 when Window.OPEN
10 then Heating.OFF
11 Heating.OFF
12 Heating.ON
13 Window.OPEN
14 Window.SHUT
```

Package Explorer
my-home
src
Home.rules
JRE System Library
src-gen

Home.rules

```
1 Device Window can be OPEN, SHUT
2 Device Heating can be ON, OFF
3
4 Rule 'Close Window, when heating turned on'
5 when Heating.ON
6 then Window.SHUT
7
8 Rule 'Switch off heating, when windows gets opened'
9 when Window.OPEN
10 then Heating.OFF
11 Heating.OFF
12 Heating.ON
13 Window.OPEN
14 Window.SHUT
```

Outline
Home
Window
Heating
Close Window, when
Switch off heating, w

State OFF

Press 'F2' for focus

Problems Javadoc Declaration Search Console

No consoles to display at this time.

Writable Insert 10 : 8

Download Professional Support

Xtext documentation site

Getting Started

[15 Minutes Tutorial](#)

[15 Minutes Tutorial -
Extended](#)

[Five simple steps to your
JVM language](#)

Reference Documentation

[The Grammar Language](#)

[Configuration](#)

[Runtime Concepts](#)

[IDE Concepts](#)

[Xtext and Java](#)

[MWE2](#)

[Typical Language](#)

[Configurations](#)

[Integration with EMF and](#)

[Other EMF Editors](#)

[Web Editor Support](#)

[Continuous Integration](#)

Additional Resources

[API Documentation](#)

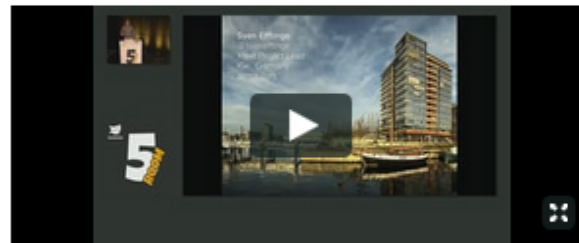
[\(JavaDoc\)](#)

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 and 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

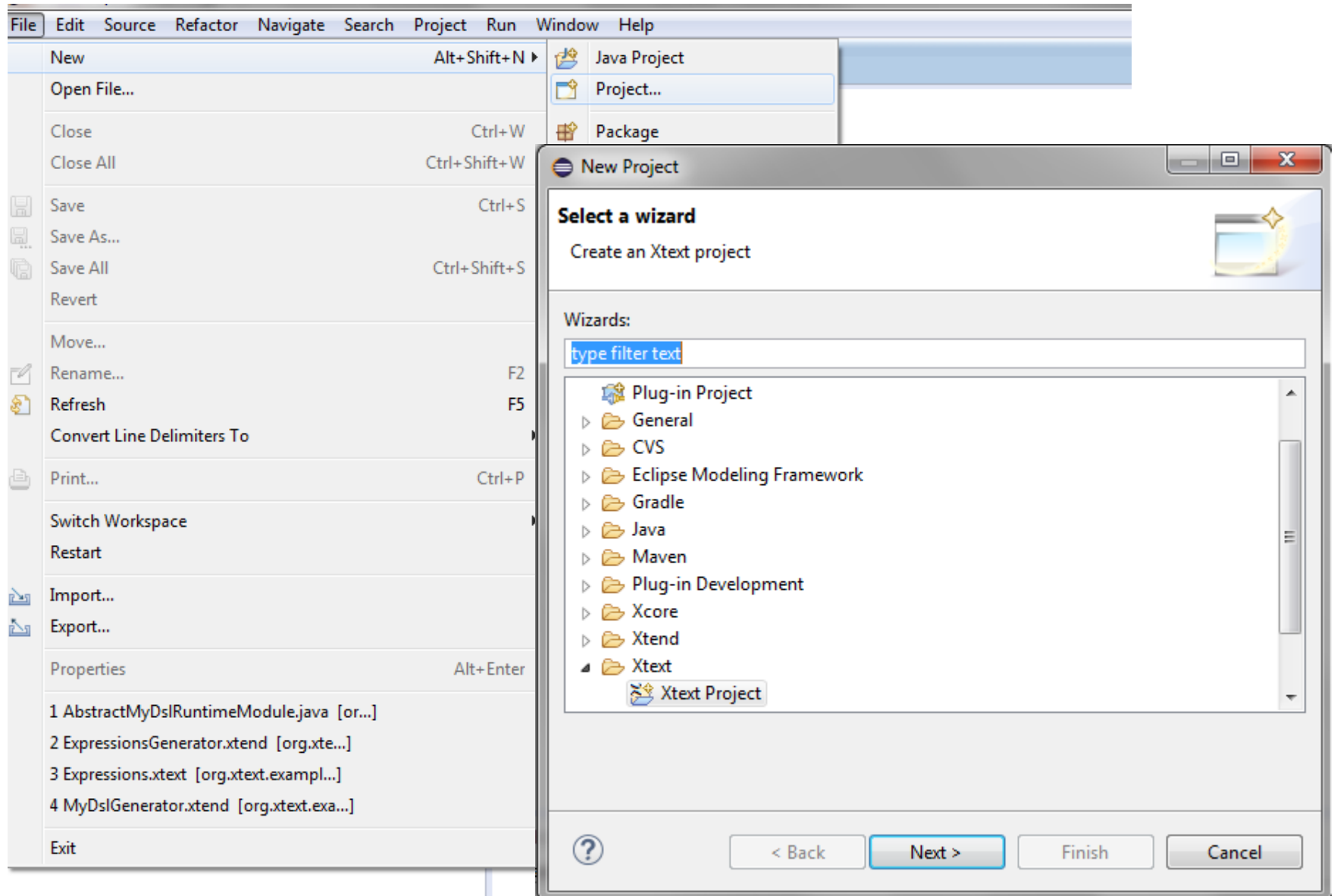
We have written two tutorials that only take you 5 minutes to do

[5 Minutes with Eclipse](#)

[5 Minutes with IntelliJ](#)

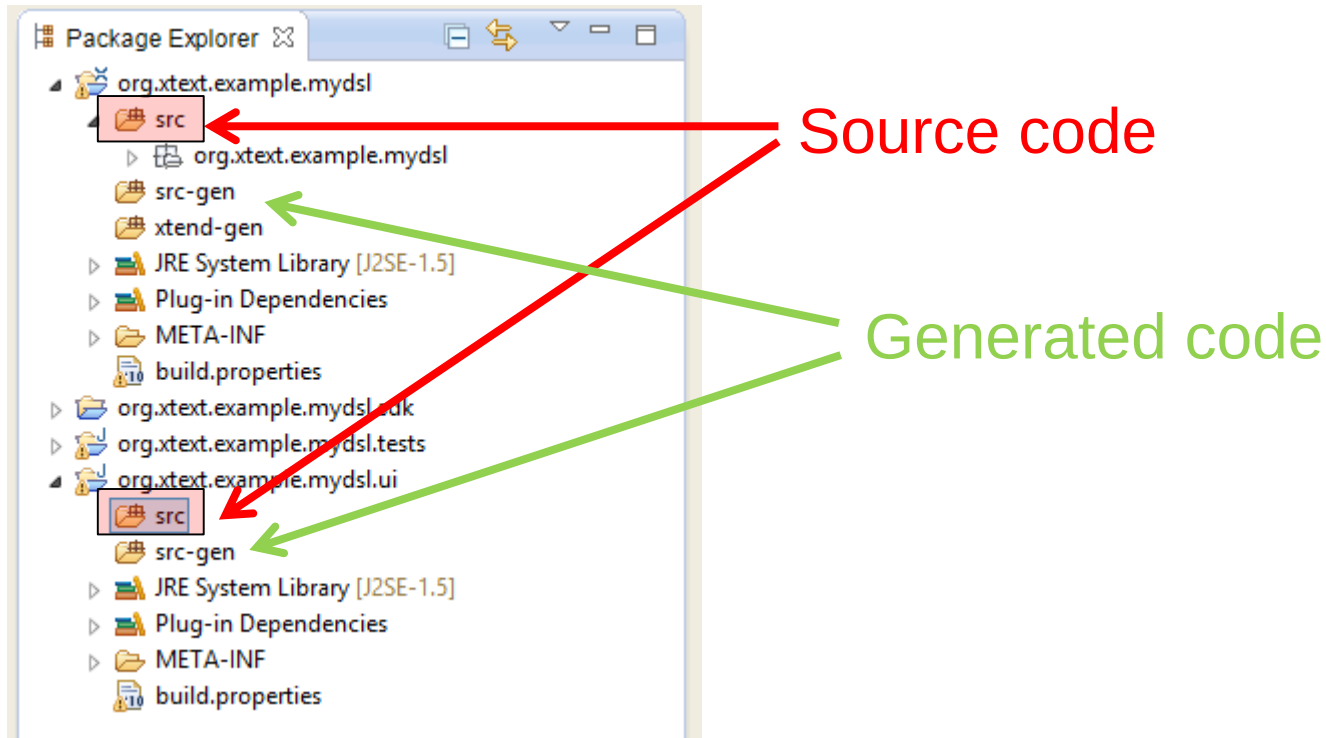
If you have done one of these tutorials, you should check out the 15 minute one on the left.

Xtext Project

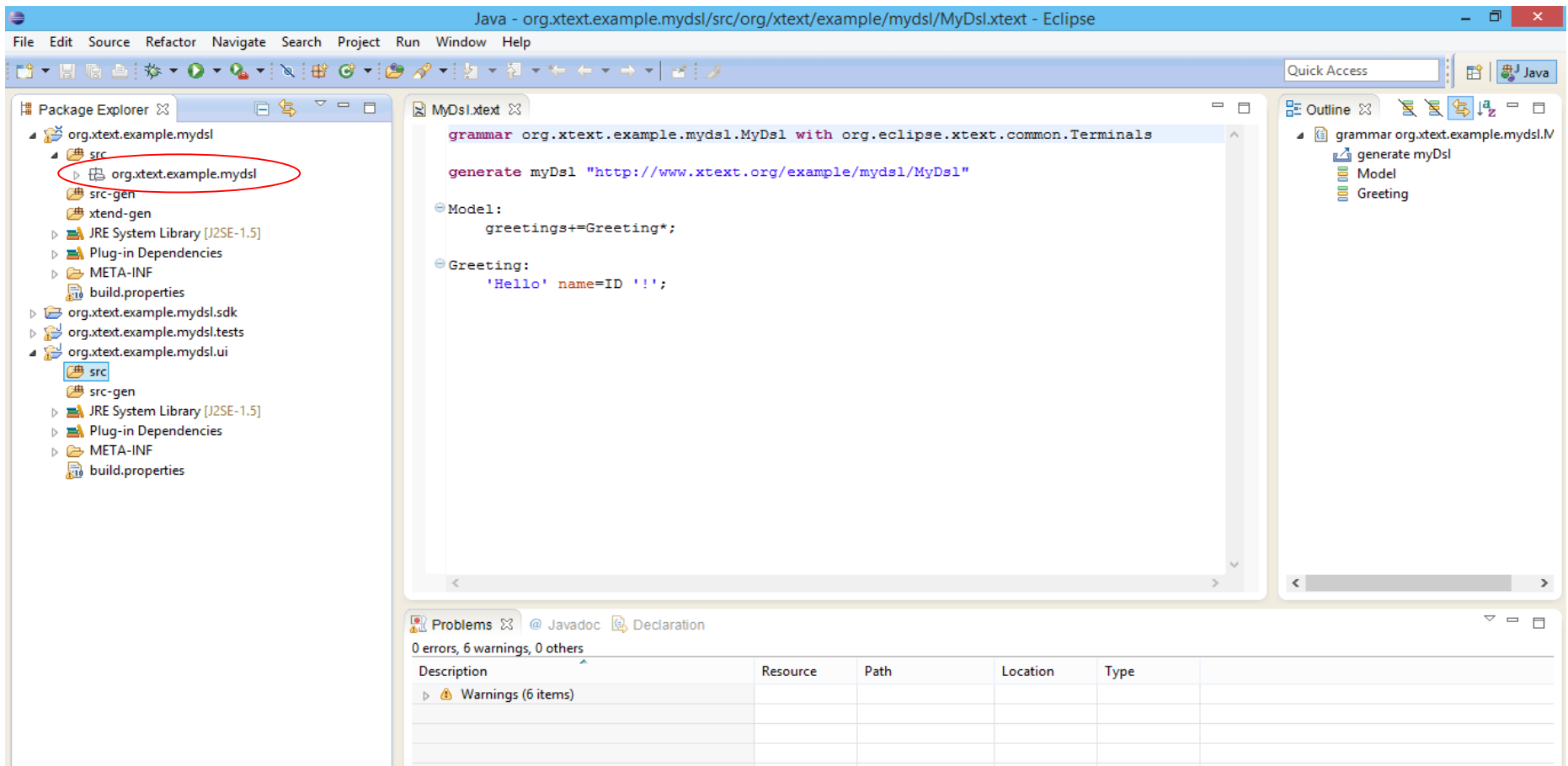


Xtext: Project overview

Using Xtext



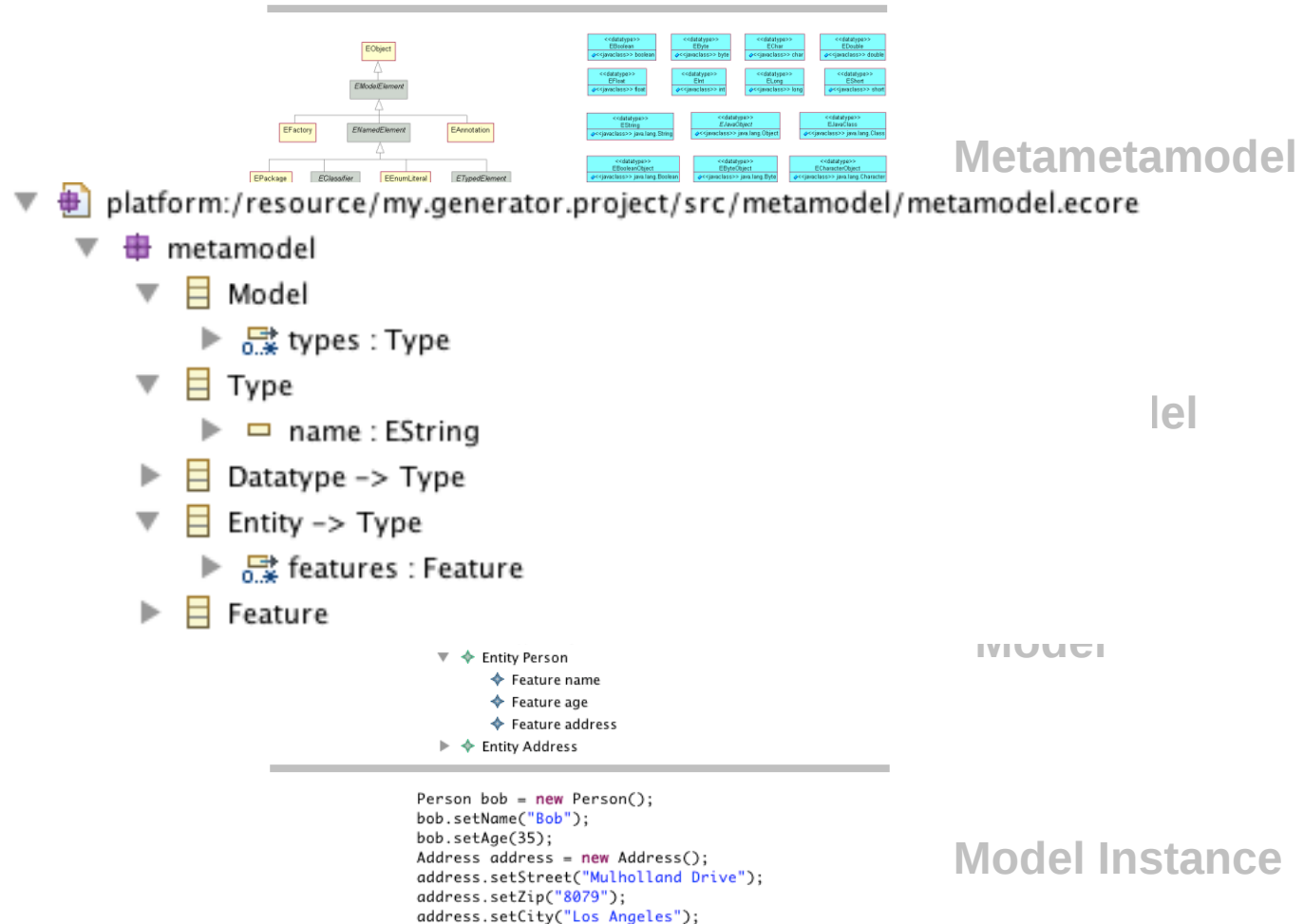
Using Xtext



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

recap: My Data Model

Metamodeling Example 3: Eclipse Modeling Framework



Syntax specification

```
generate myDsl "http://www.xtext.org/example/mydsl/MyDsl"

Domainmodel :|
  (elements += Type)*
;

Type:
  DataType | Entity
;

DataType:
  'datatype' name = ID
;

Entity:
  'entity' name = ID ('extends' superType = [Entity]? '{'
    (features += Feature)*
  '}'
;

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

Cross references to existing objects;
no rule call

Edit grammar file by specifying your own DSL

Xtext: Generate editor, parser, Ecore model

The screenshot shows the Eclipse IDE with a file named `MyDsl.xtext` open in the editor. The grammar content is as follows:

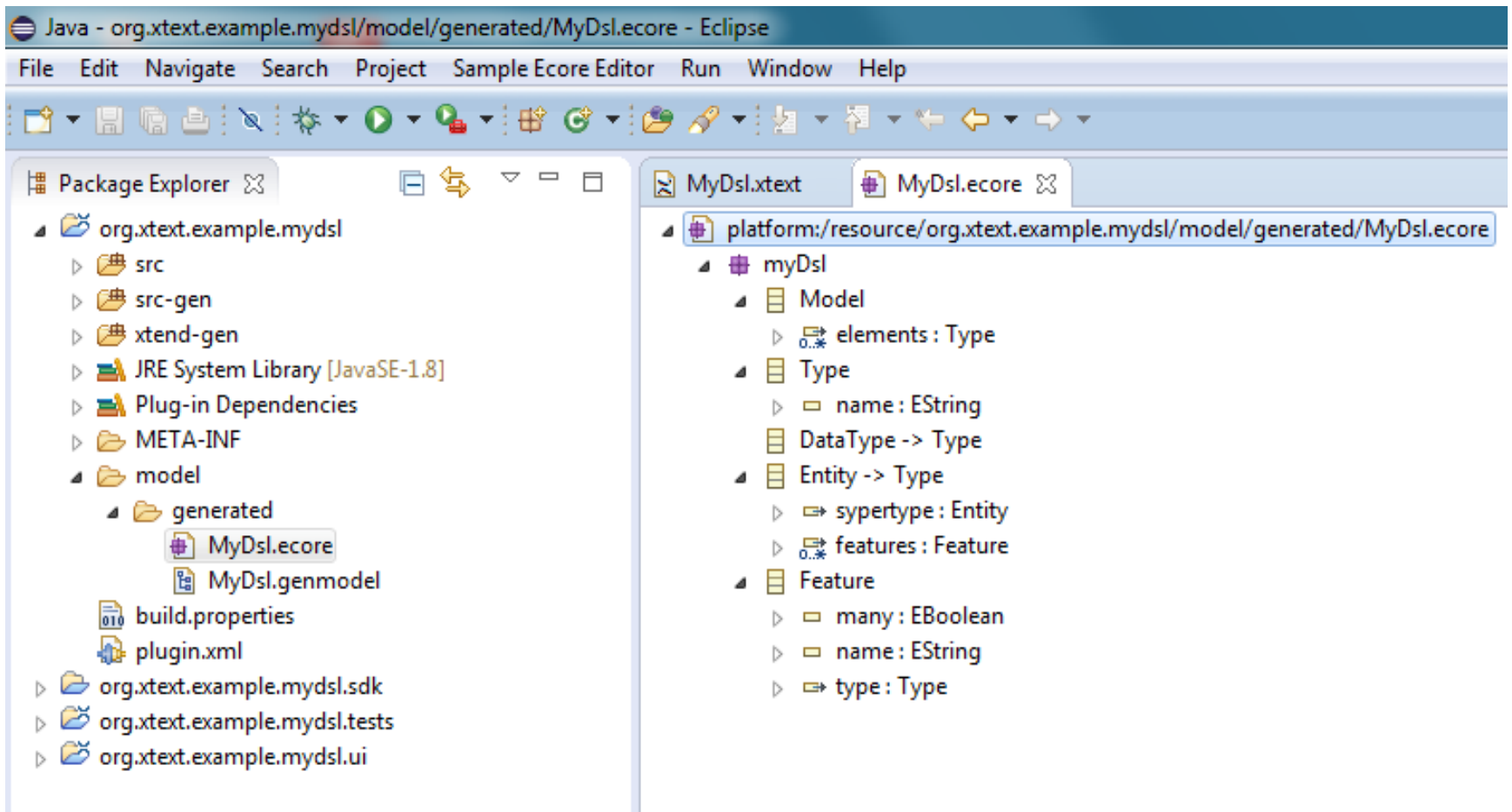
```
1 grammar org.xtext.example.mydsl.MyDsl with org.eclipse.xtext.common.Terminals
2
3 generate myDsl "http://www.xtext.org/example/mydsl/MyDsl"
4
5 Domainmodel:
6   (elements += Type)*
7 ;
8
9 Type:
10  DataType | Entity
11 ;
12
13 DataType:
14   'datatype' name = ID
15 ;
16
17 Entity:
18   'entity' name = ID ('extend'
19     (features += Feature)*
20 ;
21
22 Feature:
23   (many ?= 'many')? name = ID
24 ;
```

A context menu is open over the `Entity` rule, listing various actions and their shortcuts:

- Undo Typing (Ctrl+Z)
- Revert File
- Save (Ctrl+S)
- Quick Outline (Ctrl+O)
- Open Declaration (F3)
- Open With
- Show In (Alt+Shift+W)
- Cut (Ctrl+X)
- Copy (Ctrl+C)
- Paste (Ctrl+V)
- Rename Element (Alt+Shift+R)
- Validate
- Quick Fix (Ctrl+1)
- Source
- Find References (Ctrl+Shift+G)
- Add to Snippets...
- Run As (1 Generate Xtext Artifacts)
- Debug As
- Validate
- Replace With

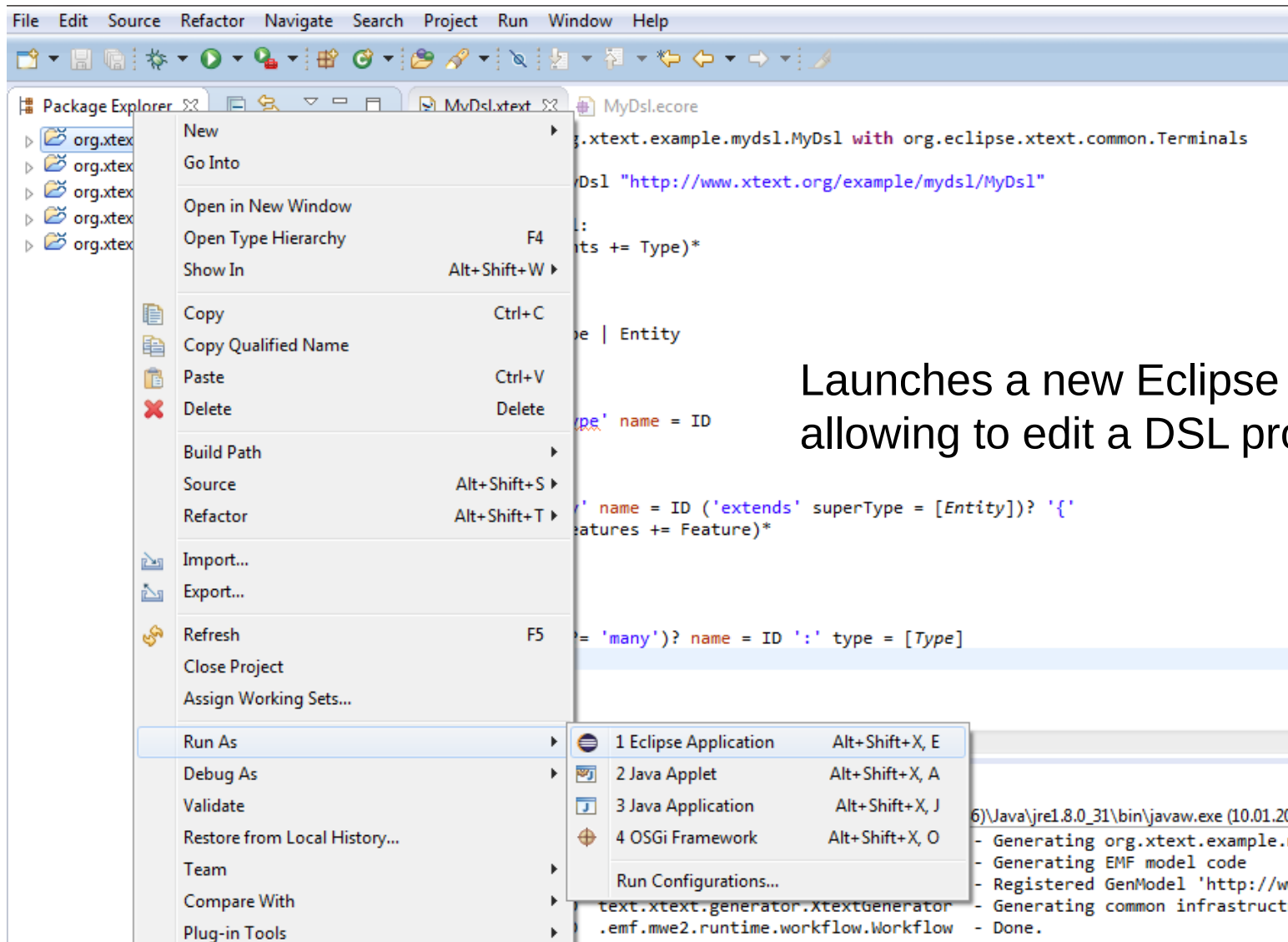
The bottom of the screen shows the `Problems` view with a warning message: `!MESSAGE Warning: The environment variable user global configuration and to def not correct please set the HOME envi Egit might behave differently since`.

Xtext: Generated Ecore model



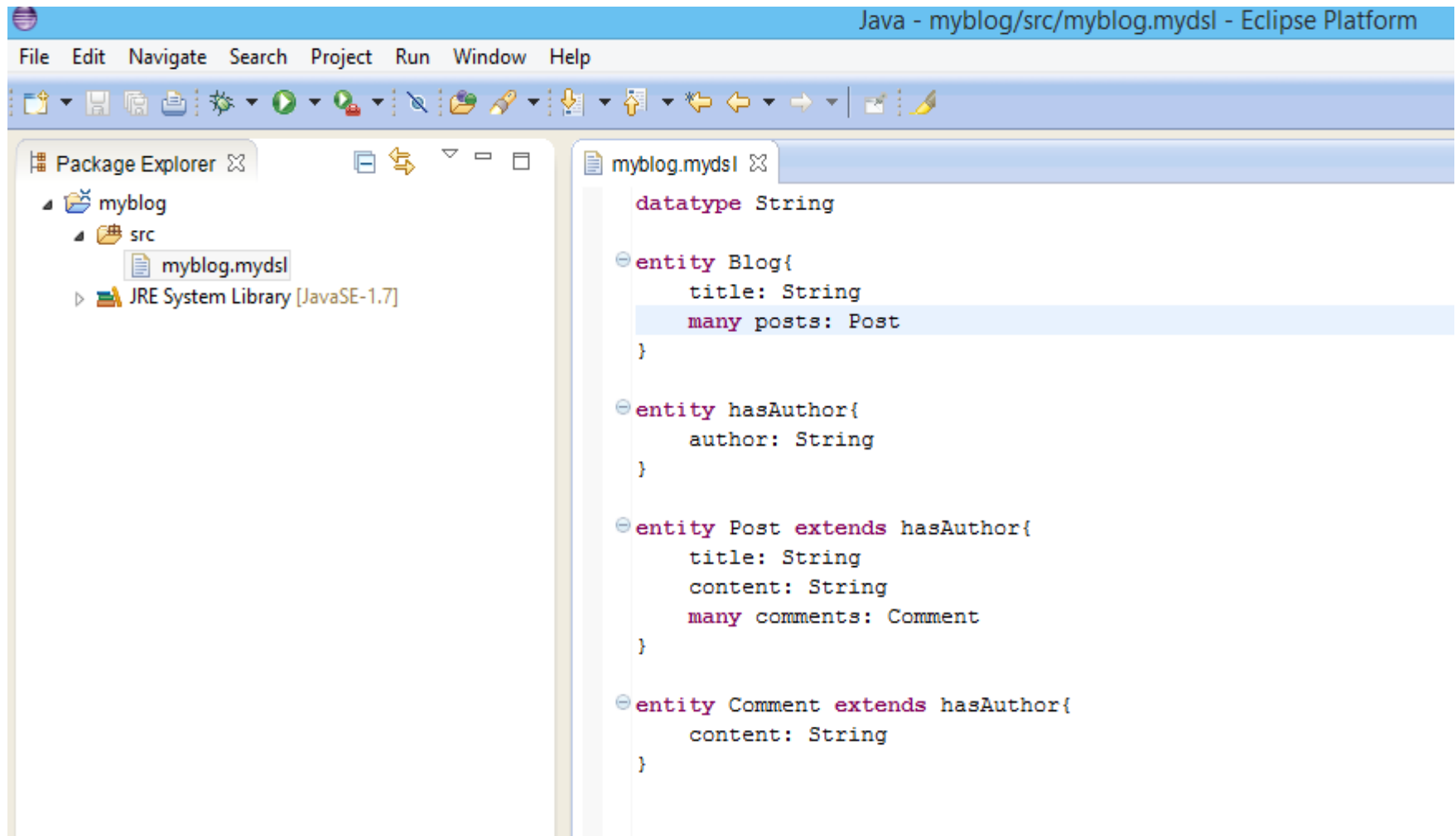
For every EClass, a Java class is generated, installing generalizations and references.

Xtext: Deploy generated artifacts



Launches a new Eclipse instance allowing to edit a DSL project

Xtext: Editing the new DSL-Project



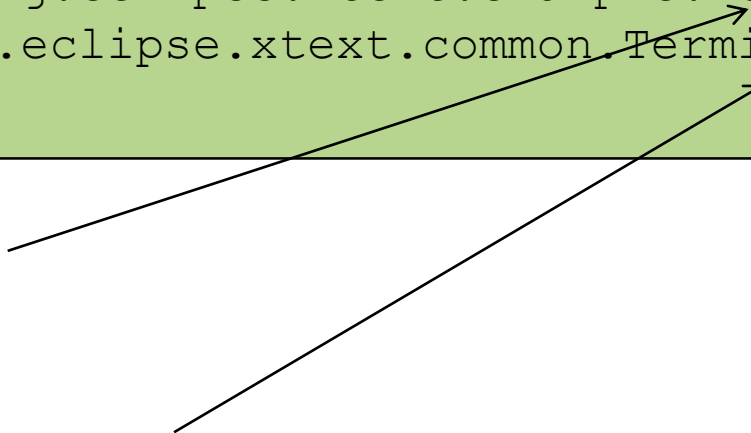
Syntax Definition

Grammars in Xtext

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

Grammar name

Grammar uses



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`.

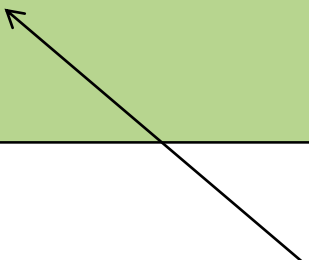
Causes the rule name class to be subclass of its return-type class.

Syntax Definition

Datatype Rules

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

optional



- 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

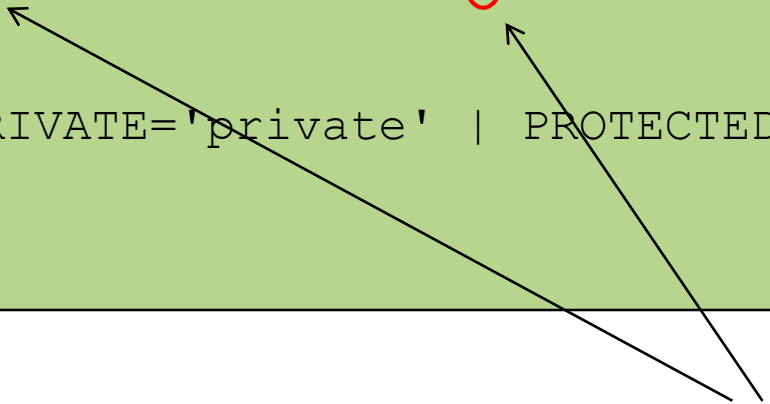
(Unordered Groups / Enums)

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

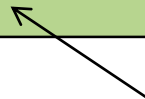
Simple Actions

```
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

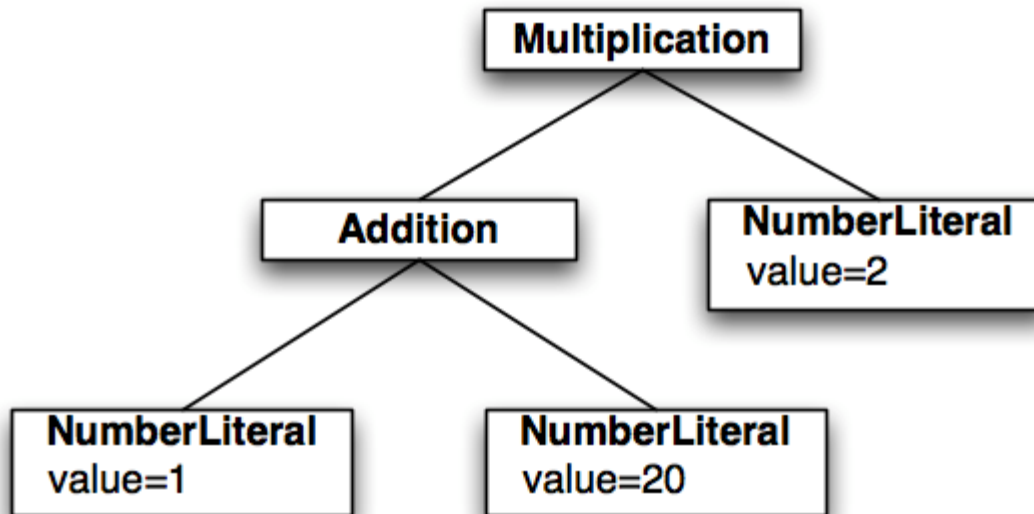
```
Addition returns Expression:  
  Multiplication ({Addition.left=current} '+'  
                  right=Multiplication)*;  
  
Multiplication returns Expression:  
  Primary ({Multiplication.left=current} '*'  
           right=Primary)*;  
  
Primary returns Expression:  
  '(' Addition ')' |  
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

The diagram illustrates assigned actions in grammar rules. Two red ovals highlight the actions `{Addition.left=current}` and `{Multiplication.left=current}`. Arrows point from the text "Assigned actions" to these ovals.

Assigned actions

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 Eclipse plugins
 - parser
 - syntax oriented editor (...)

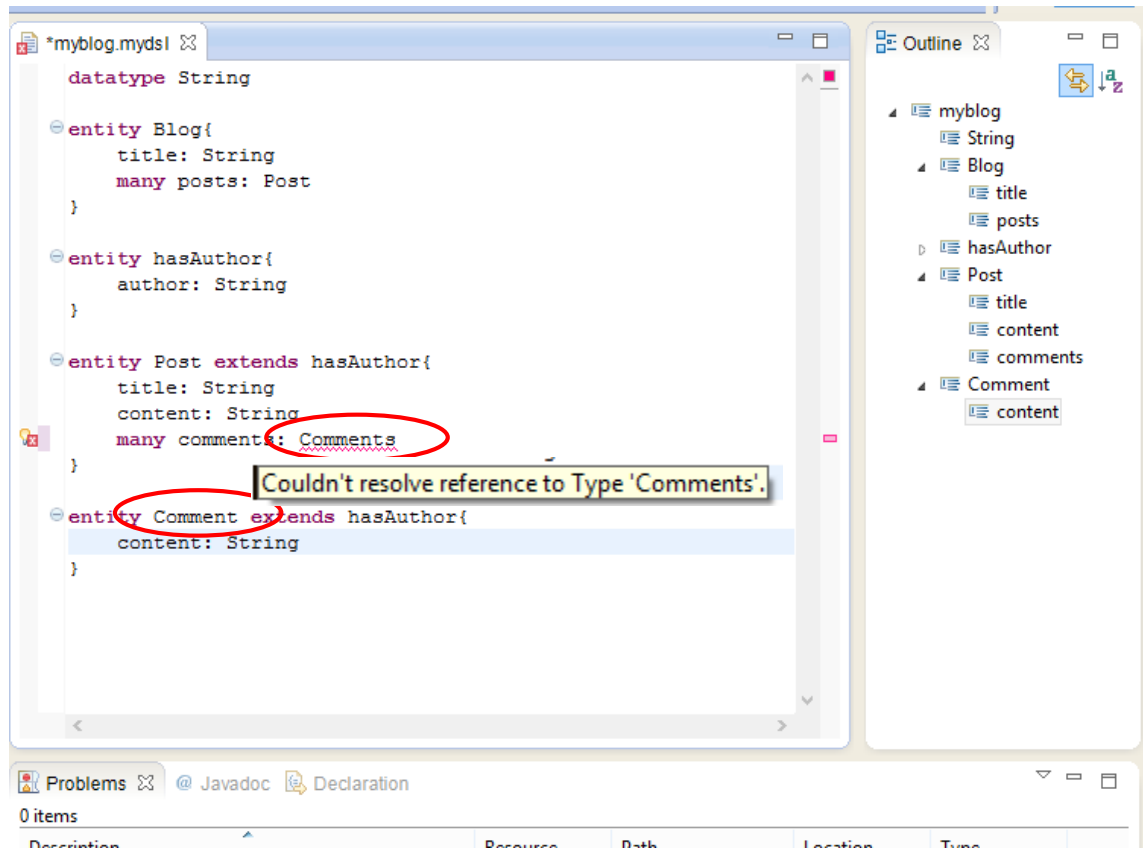
Xtext

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

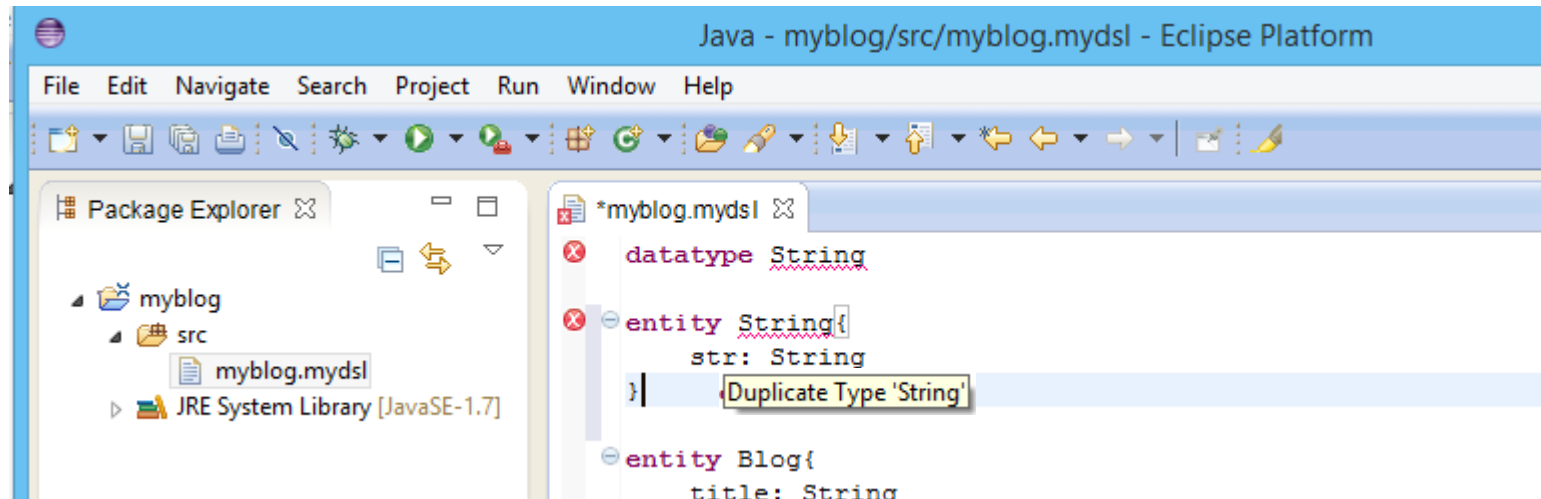
Preconfigured validations

The screenshot shows the Eclipse IDE interface. The Package Explorer on the left displays the project structure, with the file `GenerateTypeDSL.mwe2` circled. The main editor shows the content of `GenerateTypeDSL.mwe2`, which is a language configuration file. The configuration includes settings for the Eclipse plugin, plugin test, meta-data, code generation, language, serializer, and validator. A red box highlights the line `// composedCheck = "org.eclipse.xtext.validation.NamesAreUniqueValidator"` under the `validator` section, and an arrow points to it with the text "Uncomment".

```
17 }
18 eclipsePlugin = {
19     enabled = true
20 }
21 eclipsePluginTest = {
22     enabled = true
23 }
24 createEclipseMetaData = true
25 }
26 code = {
27     encoding = "windows-1252"
28     fileHeader = "/*\n * generated by Xtext \${version}\n */"
29 }
30 }
31 language = StandardLanguage {
32     name = "org.xtext.example.mydsl.TypeDSL"
33     fileExtensions = "type"
34 }
35 serializer = {
36     generateStub = false
37 }
38 validator = {
39     // composedCheck = "org.eclipse.xtext.validation.NamesAreUniqueValidator"
40 }
41 }
42 }
43 }
```

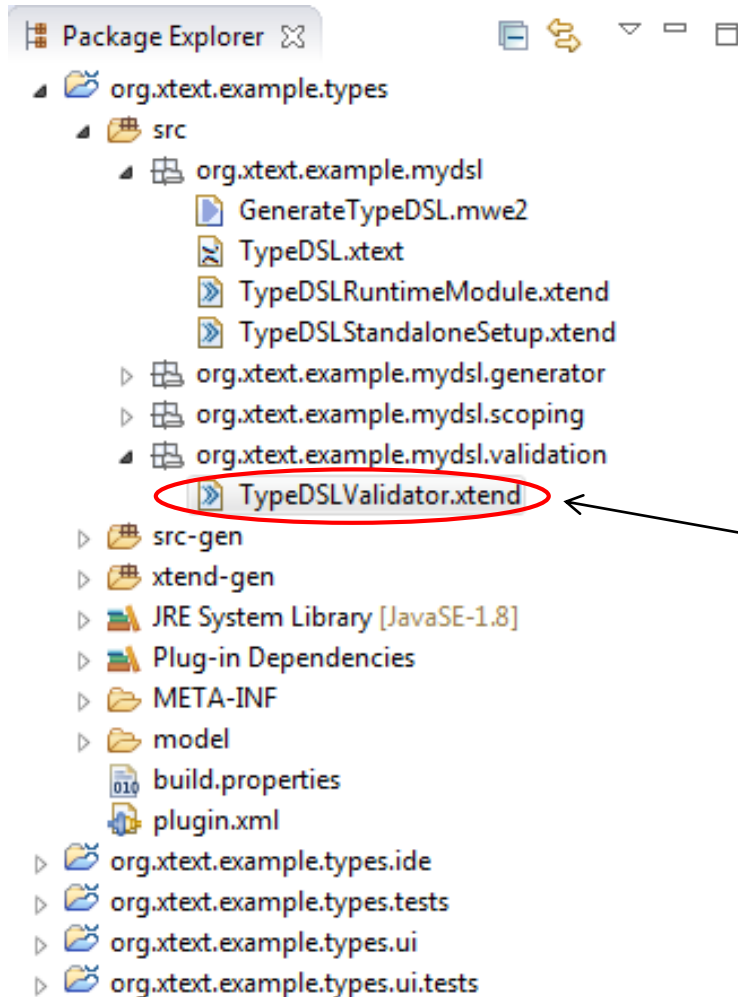
Static Semantics

Preconfigured validations



Static Semantics

Customized validations



Edit source file

Static Semantics

Customized validations

Package Explorer

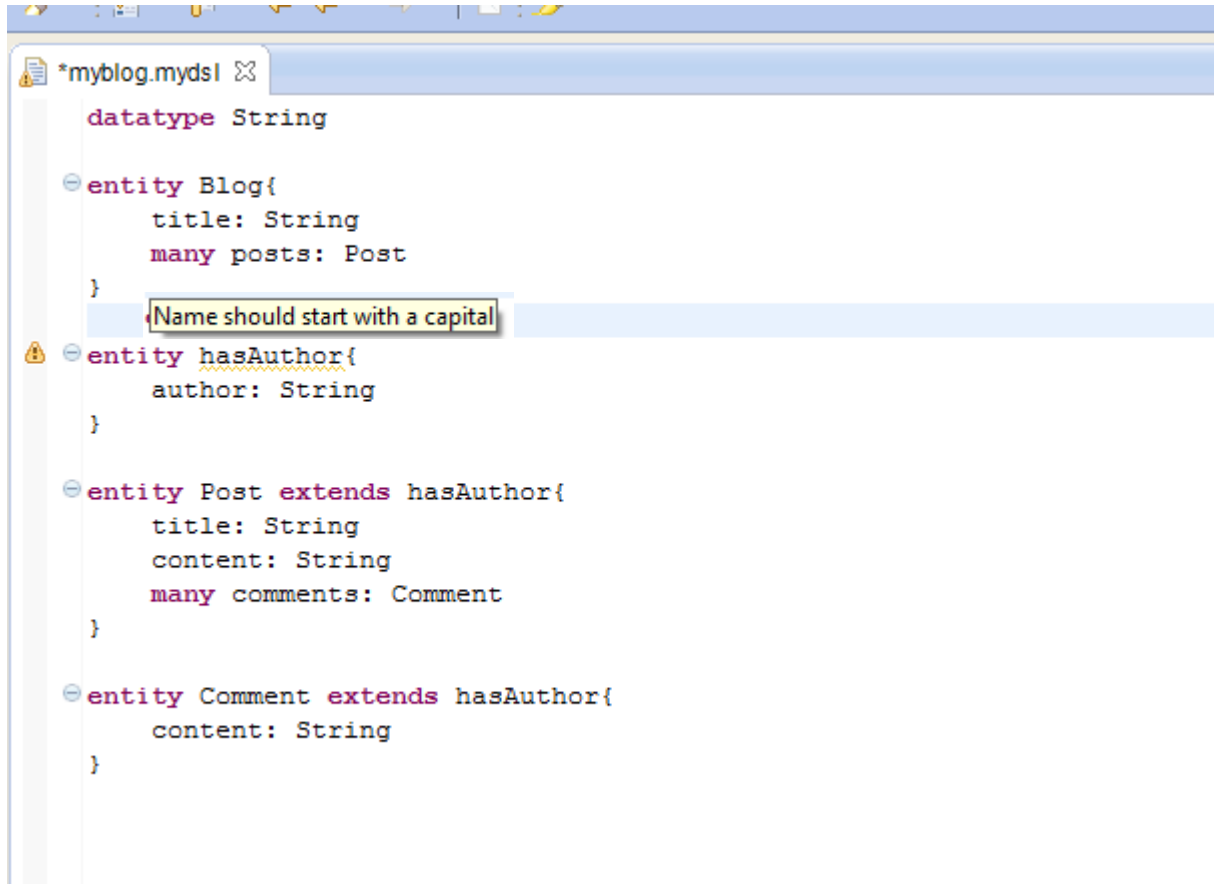
- org.xtext.example.types
 - src
 - org.xtext.example.mydsl
 - GenerateTypeDSL.mwe2
 - TypeDSL.xtext
 - TypeDSLRuntimeModule.xtend
 - TypeDSLStandaloneSetup.xtend
 - org.xtext.example.mydsl.generator
 - org.xtext.example.mydsl.scoping
 - org.xtext.example.mydsl.validation
 - TypeDSLValidator.xtend
 - src-gen
 - xtend-gen
 - JRE System Library [JavaSE-1.8]
 - Plug-in Dependencies
 - META-INF
 - model
 - build.properties
 - plugin.xml
- org.xtext.example.types.ide
- org.xtext.example.types.tests
- org.xtext.example.types.ui
- org.xtext.example.types.ui.tests

```
20 * 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.Type
8 import org.xtext.example.mydsl.typeDSL.TypeDSLPackage
9
10 /**
11  * This class contains custom validation rules.
12  *
13  * See https://www.eclipse.org/Xtext/documentation/303_runtime_concepts.html#validation
14  */
15 class TypeDSLValidator extends AbstractTypeDSLValidator {
16
17     public static val INVALID_NAME = 'invalidName'
18
19     @Check
20     def checkTypeNameStartsWithCapital(Type type) {
21         if (!Character.isUpperCase(type.name.charAt(0))) {
22             warning('Name should start with a capital',
23                 TypeDSLPackage.Literals.TYPE_NAME,
24                 INVALID_NAME)
25         }
26     }
27
28 }
29
```

Check annotation
→ each Type object is checked accordingly

Static Semantics

Customized validations



The screenshot shows a code editor window with a tab labeled `*myblog.mysdl`. The code defines a GraphQL schema with the following structure:

```
datatype String

entity Blog{
  title: String
  many posts: Post
}

entity hasAuthor{
  author: String
}

entity Post extends hasAuthor{
  title: String
  content: String
  many comments: Comment
}

entity Comment extends hasAuthor{
  content: String
}
```

A custom validation message, `Name should start with a capital`, is displayed in a light blue box, highlighting the `title` field of the `Blog` entity. A yellow warning icon is visible in the left margin next to the `entity hasAuthor` definition.

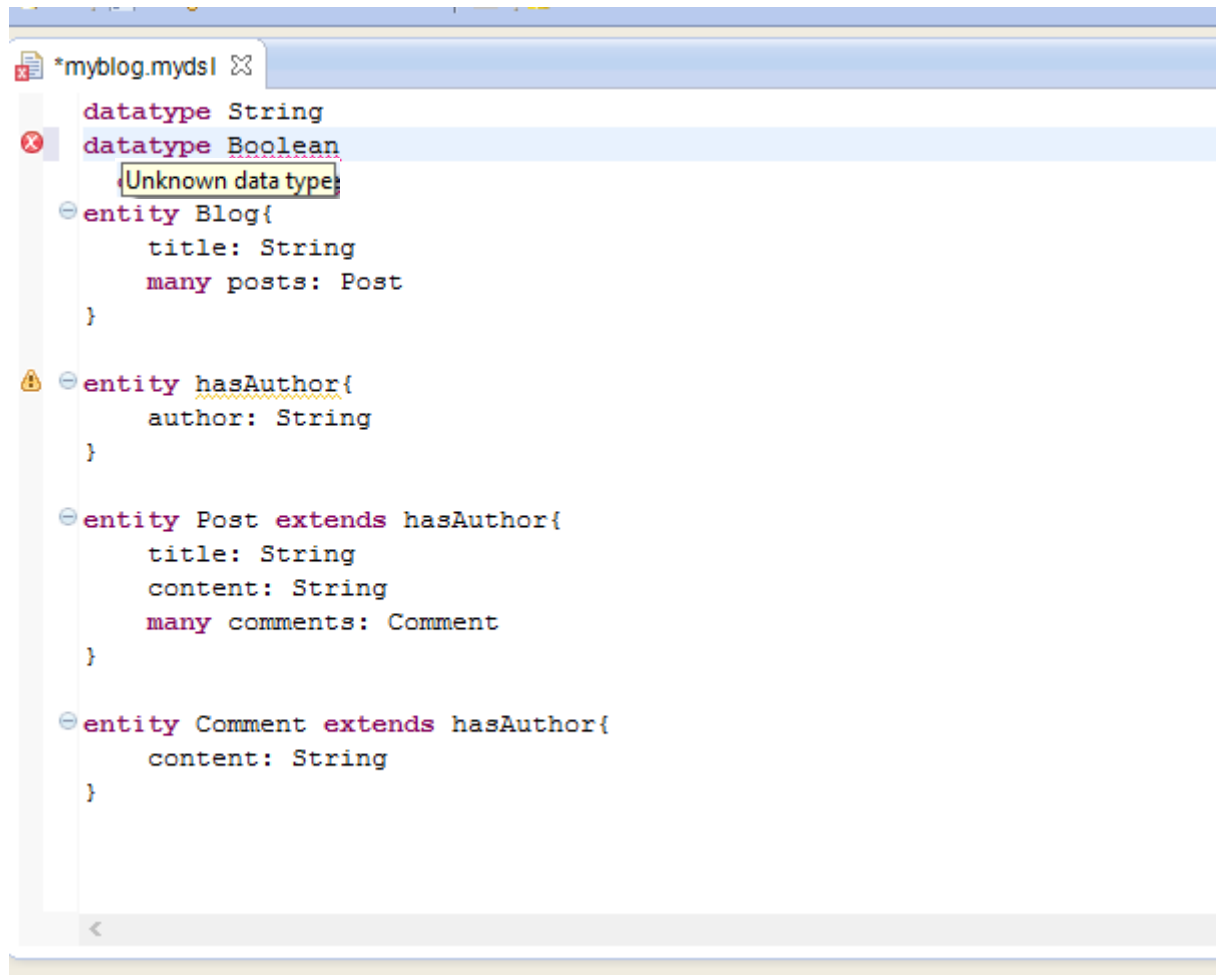
Static Semantics

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



The screenshot shows a code editor window titled `*myblog.mydsl`. The code defines a domain-specific language for a blog system. There are two error markers on the left margin: a red 'X' for an unknown data type and a yellow warning triangle for a missing closing brace.

```
datatype String
datatype Boolean
  Unknown data type.
entity Blog{
  title: String
  many posts: Post
}
entity hasAuthor{
  author: String
}
entity Post extends hasAuthor{
  title: String
  content: String
  many comments: Comment
}
entity Comment extends hasAuthor{
  content: String
}
```

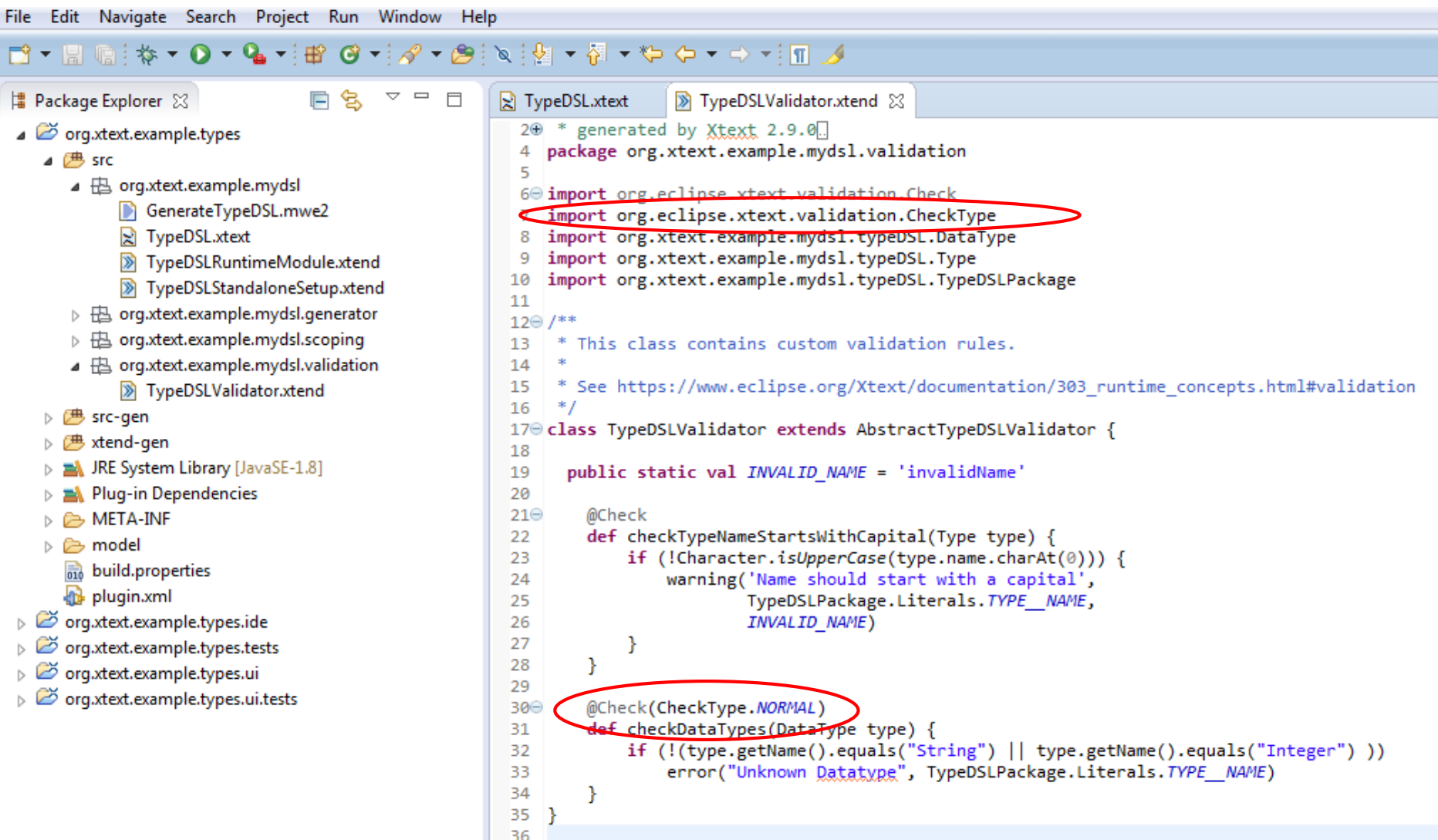
Static Semantics

Customized validations

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

Static Semantics

Customized validations



The screenshot displays the Eclipse IDE interface. On the left, the Package Explorer shows a project named `org.xtext.example.types` with a source folder `src`. Inside `src`, there is a package `org.xtext.example.mydsl` containing `GenerateTypeDSL.mwe2`, `TypeDSL.xtext`, `TypeDSLRuntimeModule.xtend`, and `TypeDSLStandaloneSetup.xtend`. Other packages include `org.xtext.example.mydsl.generator`, `org.xtext.example.mydsl.scoping`, and `org.xtext.example.mydsl.validation` (containing `TypeDSLValidator.xtend`). There are also folders for `src-gen`, `xtend-gen`, and system libraries like `JRE System Library [JavaSE-1.8]`.

The main editor window shows the file `TypeDSLValidator.xtend`. The code is as follows:

```
2+ * generated by Xtend 2.9.0
4 package org.xtext.example.mydsl.validation
5
6- import org.eclipse.xtext.validation.Check
7- import org.eclipse.xtext.validation.CheckType
8 import org.xtext.example.mydsl.typeDSL.DataType
9 import org.xtext.example.mydsl.typeDSL.Type
10 import org.xtext.example.mydsl.typeDSL.TypeDSLPackage
11
12- /**
13  * This class contains custom validation rules.
14  *
15  * See https://www.eclipse.org/Xtext/documentation/303\_runtime\_concepts.html#validation
16  */
17 class TypeDSLValidator extends AbstractTypeDSLValidator {
18
19     public static val INVALID_NAME = 'invalidName'
20
21-     @Check
22     def checkTypeNameStartsWithCapital(Type type) {
23         if (!Character.isUpperCase(type.name.charAt(0))) {
24             warning('Name should start with a capital',
25                 TypeDSLPackage.Literals.TYPE__NAME,
26                 INVALID_NAME)
27         }
28     }
29
30-     @Check(CheckType.NORMAL)
31     def checkDataTypes(DataType type) {
32         if (!(type.getName().equals("String") || type.getName().equals("Integer"))) {
33             error("Unknown Datatype", TypeDSLPackage.Literals.TYPE__NAME)
34         }
35     }
36 }
```

Red circles highlight the `import org.eclipse.xtext.validation.CheckType` statement on line 7 and the `@Check(CheckType.NORMAL)` annotation on line 30.

Static Semantics

Customized validations

Not checked until saved

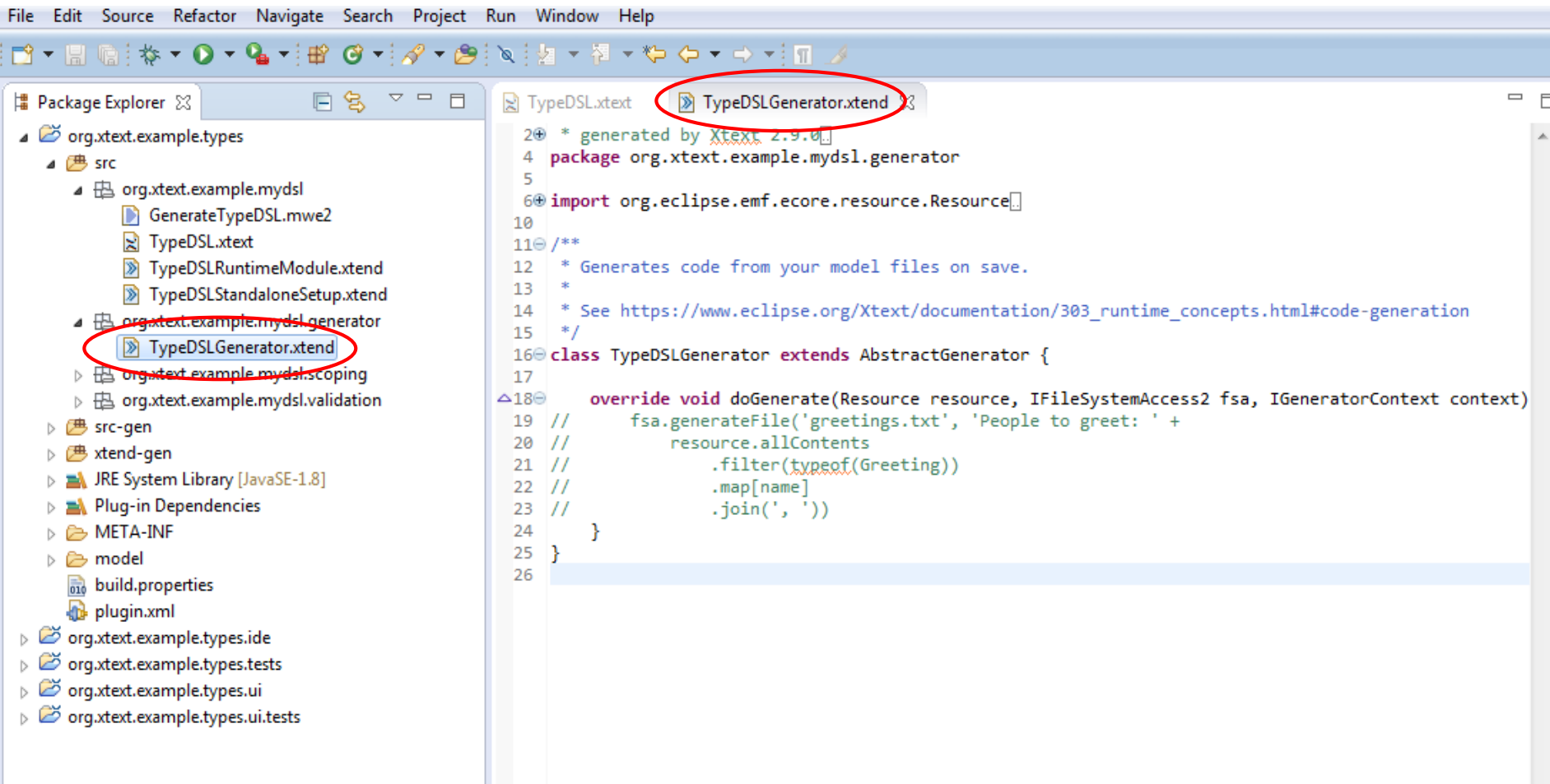
```
*myblog.myds|  
datatype String  
datatype Boolean  
  
entity Blog{  
  title: String  
  many posts: Post  
}  
  
entity hasAuthor{  
  author: String  
}  
  
entity Post extends hasAuthor{  
  title: String  
  content: String  
  many comments: Comment  
}  
  
entity Comment extends hasAuthor{  
  content: String  
}
```

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 → [documentation!!!](#)

Code Generation with Xtend



Code generation stub to be modified

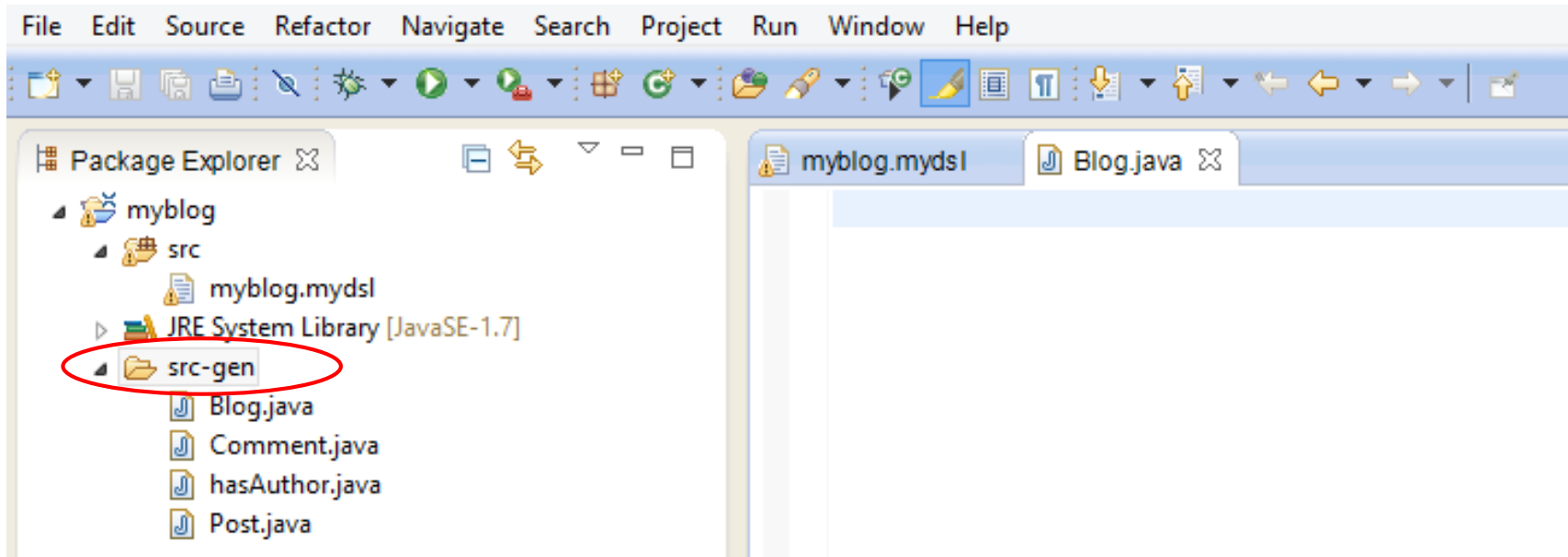
Code Generation with Xtend

```
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.IQualifiedDataProvider
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 IQualifiedDataProvider
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

