

---

# Create your own DSL with xtext

**SCG Seminar @ UniBE, 23. November 2010**

---

- 
- MS in Computer Science, Uni Bern 2003
  - Master Thesis: „Detecting Software Patterns using Formal Concept Analysis”
  - 3 years Software Engineer  
Glue Software Engineering AG
  - Since January 2007  
Software Architect  
Zühlke Engineering AG in Bern

# Definition DSL



---

„A computer programming language of limited expressiveness focused on a particular domain.“

[Martin Fowler]

## Key elements

- Computer programming language
  - human → machine
  - Enhance communication human ↔ human
- Language nature / Sense of fluency
- Limited expressiveness
- Domain focus



## ■ Motivation for a DSL

- Improving Development Productivity
- Communication with Domain Experts

# Goal



---

```
datatype String
datatype Bool
```

```
entity Session {
  title: String
  isTutorial : Bool
}
```

```
entity Conference {
  name : String
  attendees : Person*
  speakers : Speaker*
}
```

```
entity Person {
  name : String
}
```

```
entity Speaker extends Person {
  sessions : Session*
}
```

# What is needed?



- 
- Parser
    - yacc
    - antlr
  - Lexer
  - Editor
    - Syntax Highlighting
    - Outline
    - Code Completion

The logo for Xtext features the word "Xtext" in a white, sans-serif font. The letter "x" is replaced by a stylized, golden-yellow symbol consisting of two curved lines that meet at a central point, resembling a double-headed arrow or a stylized 'X'.

Xtext

Language Development Framework

# How to start?

---



**Describe the concrete syntax of your language**

**Will contain information about how the parser shall create a model during parsing.**



- Consists of rules like:

```
DomainModel :  
  Entity* ;
```

- Cardinality

<i>(no operator)</i>	exactly one
?	zero or one
*	zero or more
+	one or more

# Grammar

## Give elements names!



```
DomainModel :  
  (elements += Entity) * ;
```

### ■ Assignment operators

=	single feature
+=	list
?=	condition (boolean)

# Grammar Keywords



- Introduce keywords!

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

- ID is predefined rule

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

- (...) ? means optional

# Grammar Inheritance



```
DomainModel :  
    (elements+=Type) *;  
  
Type :  
    DataType | Entity;  
  
DataType :  
    'datatype' name=ID;
```

---

```
Feature:
```

```
name=ID ':' type=TypeRef;
```

- Sorry, no new concept ...

# Grammar

## Optional keyword

---



```
TypeRef :  
  referenced=[Type] (multi?='*')?;
```

# Prerequisites

---



## ■ Installed Xtext

- Download Link:

<http://xtext.itemis.com/xtext/language=en/23947/downloads>



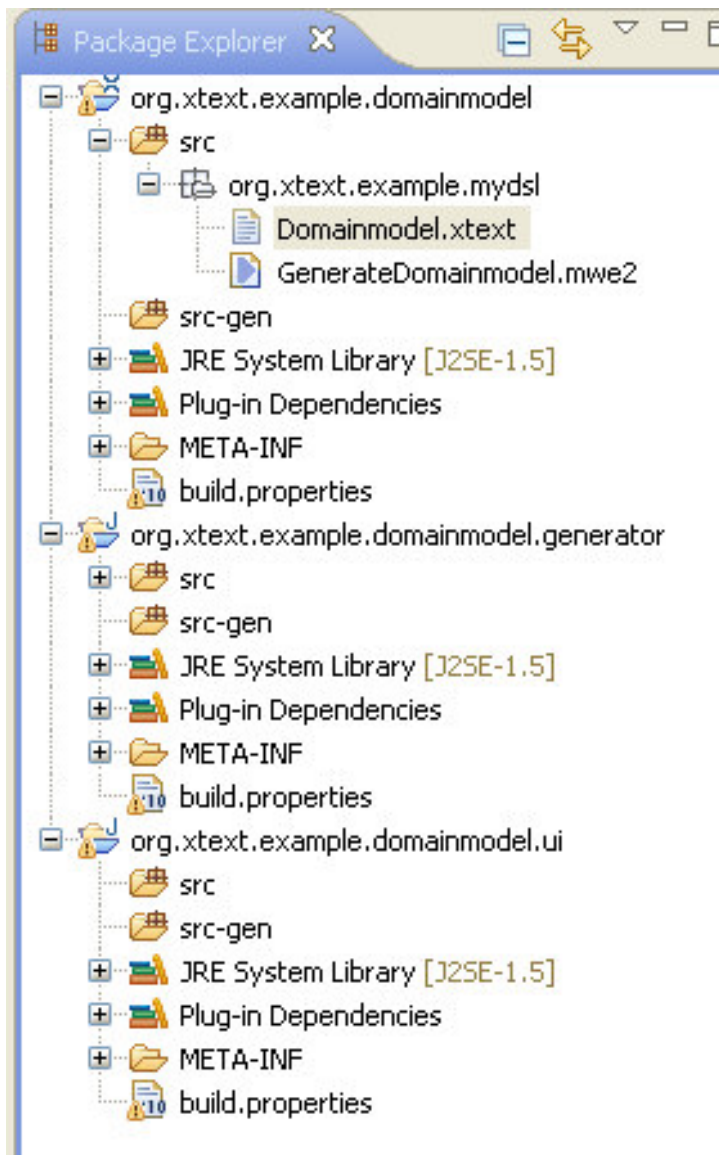
# Create a new project



File → New → Project... → Xtext → Xtext project

A screenshot of the 'New Xtext Project' wizard dialog box. The title bar reads 'New Xtext Project'. The main heading is 'New Xtext Project' with the subtitle 'This wizard creates a couple of projects for Xtext DSL.' and the Xtext logo. The dialog is divided into several sections: 'Project name' with a text field containing 'org.eclipse.xtext.example.domainmodel'; 'Use default location' checked; 'Location' with a text field containing '/workspace/org.eclipse.xtext.example.domainmodel' and a 'Browse...' button; 'Language' section with 'Name' field containing 'org.eclipse.xtext.example.Domainmodel' and 'Extensions' field containing 'dmodel'; 'Layout' section with 'Create a generator project' checked; and 'Working sets' section with 'Add project to working sets' unchecked and a 'Working sets' dropdown menu with a 'Select...' button. At the bottom, there are buttons for '< Back', 'Next >', 'Cancel', and 'Finish'.

# Explain Project Layout



# Build your own grammar

---



## Meta information of the grammar:

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

```
generate domainmodel  
"http://www.xtext.org/example/mydsl/Domainmodel"
```



# Run application

The screenshot shows the Eclipse IDE interface. On the left, the Package Explorer displays a project structure for 'org.xtext.example.domainmodel'. A context menu is open over the 'src' folder, with the 'Run As' option selected. The sub-menu for 'Run As' is visible, showing four options: '1 Eclipse Application' (Alt+Shift+X, E), '2 Java Applet' (Alt+Shift+X, A), '3 Java Application' (Alt+Shift+X, J), and '4 OSGi Framework' (Alt+Shift+X, O). The main editor window shows a grammar file 'Domainmodel.xtext' with the following code:

```
grammar org.xtext.example.mydsl.Dc
    domainmodel "http://www.xtendertechnology.com/DomainModel:"
    elements+=Type) *;
    datatype | Entity;
    type:
    datatype' name=ID;
    entity' name=ID ('extends' st
    (features+=Feature) *
    ';
    name=ID ':' type=TypeRef;
    ref :
    referenced=[Type] (multi?='*') ;
```

# Write a file



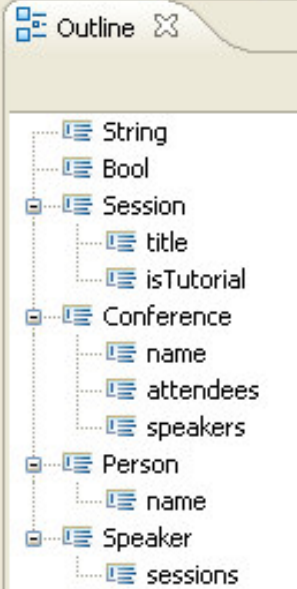
```
demo.dmodel x
datatype String
datatype Bool

entity Session {
  title: String
  isTutorial : Bool
}

entity Conference {
  name : String
  attendees : Person*
  speakers : Speaker*
}

entity Person {
  name : String
}

entity Speaker extends Person {
  sessions : Session*
}
```



# What's next?



- 
- Interpretation of the model
  - Code generator

# Interpretation

```
public static void main(String[] args) {
    init();

    DomainModel dm = readDomainModel("src/model/Example.dmodel");
    System.out.println("Number of elements: " + dm.getElements().size());

    for (Type type : dm.getElements()) {
        System.out.println(" Name: " + type.getName());
    }

    DomainModel aNewModel = DomainmodelFactory.eINSTANCE.createDomainModel();
}

private static DomainModel readDomainModel(String model) {
    ResourceSet rs = new ResourceSetImpl();
    Resource resource = rs.getResource(URI.createURI(model), true);
    EObject eobject = resource.getContents().get(0);
    System.out.println("Object Metaclass: " + eobject.eClass().getName());
    DomainModel dm = (DomainModel) eobject;
    return dm;
}

private static void init() {
    new DomainmodelStandaloneSetup().createInjectorAndDoEMFRegistration();
}
```



# Generator

## Use xpanse template language



```
<<IMPORT org::xtext::example::mysdl::domainmodel>>
```

```
<<EXTENSION templates::Extensions>>
```

```
<<DEFINE main FOR DomainModel>>  
  <<EXPAND typeDef FOREACH elements>>  
<<ENDDEFINE>>
```

```
<<DEFINE typeDef FOR Type>>
```

```
<<ENDDEFINE>>
```

```
<<DEFINE typeDef FOR Entity->>
```

```
<<FILE name+".txt"->>
```

```
This is an example of a generated file.
```

```
The input element was "Hello <<this.javaConformName()>>!"
```

```
All greetings in the same file:
```

```
<<ENDFILE->>
```

```
<<ENDDEFINE>>
```

- 
- <http://www.xtext.org/>
    - Tutorial:  
[http://www.eclipse.org/Xtext/documentation/1\\_0\\_1/xtext.html](http://www.eclipse.org/Xtext/documentation/1_0_1/xtext.html)
    - Interpretation:  
[http://www.eclipse.org/Xtext/documentation/1\\_0\\_1/xtext.html#processing\\_Xtext\\_models](http://www.eclipse.org/Xtext/documentation/1_0_1/xtext.html#processing_Xtext_models)
    - Generator:  
[http://www.eclipse.org/Xtext/documentation/1\\_0\\_1/xtext.html#getting-started-xpand](http://www.eclipse.org/Xtext/documentation/1_0_1/xtext.html#getting-started-xpand)
  
  - DSL Buch von Martin Fowler:  
<http://martinfowler.com/dsl.html>
  
  - Zühlke:  
<http://www.zuehlke.com>