Workshop: How to create a DSL with Xtext

DAMIAAN VAN DER KRUK

Objective

Create a simple DSL with Xtext on your own machine

- Grammar
- Code generation
- Validation

What is a Domain Specific Language (DSL)?

A domain specific language (DSL) is a formal, processable language targeting at a specific viewpoint or aspect of a system

Examples

- HTML → Markup for websites/webdocuments
- SQL → Querying databases
- VHDL → Hardware design
- Capella/Arcadia DSL → Model based engineering solution

Xtext

Xtext is a language engineering framework

Grammar driven

Open source & an Eclipse.org project

Multiple platform/IDE support (Eclipse, IntelliJ & web)

Development Environment

JDK >= 1.8

- http://www.oracle.com/technetwork/java/javase/downloads/index.html

Eclipse 4.6.2 Neon.2 (Eclipse IDE for Java and DSL Developers)

- https://eclipse.org/downloads/eclipse-packages/

Xtext 2.10 (included in Eclipse IDE for Java and DSL Developers)

Outline of Xtext Eclipse project

Demo

Xtext Grammar Exercises

Get exercise files from the USB stick or GitHub:

- https://github.com/dvdkruk/xtext-workshop/tree/master/org.example.domainmodel.exercises/exercisefiles

Exercise: Implement exercise files, one by one

- Change `Domainmodel.xtext`
- Run MWE2 workflow
- Run second Eclipse instance
  - Copy exercise file into an Eclipse project
  - Check and test
- Repeat

Exercise 0 - Create A New Xtext Project

Create A New Xtext Project

- File → New → Project… → Xtext → Xtext project

Project name:	org.example.domainmodel
Language name:	org.example.domainmodel.Domainmodel
Language extension:	dmodel

Build the Greetings Hello example grammar

- Right click `Domainmodel.xtext` → Run As → 1 Generate Xtext Artifacts

Start a second Eclipse instance

- Right click `org.example.domainmodel` → Run As → 1 Eclipse Application

Exercise 1 – Basic Grammar Elements

File: exercise1.dmodel

The first rule in a grammar is used a the start rule
- `Domainmodel: … ;`

Keywords are defined between single quotes
- `'{'` `'}'` `'entity‘`

Features are assigned to a rule with = or +=, the later one is used for lists
- `name=ID` `elements+=Elements`

EBNF Expressions for cardinality
- Default = exactly one, ? = optional, + = at least once, * = any number

Exercise 2 – Cross-References, Groups & Terminal Rules

File: exercise2.dmodel

Cross-references

- Reference: `[EClass]`

Reference w/ syntax: `[Eclass|Syntax]`

Groups

- Alternatives: `(Entity | Datatype)`

Unordered: `(Entity & DataType)`

Terminal rules are used for value literals (floats, etc.)

- Built-in: ID, STRING, INT
- `QualifiedName: ID ('.' ID)*;`

Exercise 3 – Optional Elements

File: exercise3.dmodel

Boolean attributes are defined with `?=`

- `manditory?='manditory'`

In most cases used in combination with the `?` (optional) operator

- `(manditory?='manditory')? …`

Exercise 4 – Multiple Files and Imports

Files: exercise4.dmodel, exercise4_common.dmodel & exercise4_datatype.dmodel

Cross-references are resolved over all files on the build path
- Can be limited by scoping

Xtext builds a tree reference based on the name feature
- Also used for outline

`importedNamespace` is a special feature to make references shorter
- `importedNamespace=QualifiedNameWithWildcard`

Code Generation: Xtend

Xtend is a dialect of Java and compiles to Java source code.

Has a lot of modern features like lambdas, operator overloading, method dispatching, etc.

Template Expression
- Templates are surround by triple single quotes (`''' template '''`)
- Terminal for interpolated expression are guillemets (`«expression»`)
  - Ctrl+shift+< = «, ctrl+shift+> = » or ctrl+space inside a template block

Conditions and loops
- `«IF number != null» … «ENDIF»`
- `«FOR element : elements» … «ENDFOR»`

Exercise: Code Generator

Generate

- C struct for an Entity and/or
- POJO for an Entity and/or
- HTML page for an Entity

Use

```
@Inject extension IQualifiedNameProvider

for (e : resource.allContents.toIterable.filter(Entity)) {
    fsa.generateFile(e.fullyQualifiedName.toString("/") + ".c/java/html", e.compile)
}

def compile(Entity e) '''
    Your template here
'''
```

Exercise: Validation Rule

Implement the following validation rules:

- Show a warning when: The name of an entity should start with a capitol

```
entity job { //Warning: Name should start with a capitol
    …
}
```

- Show an error when: A feature name is not unique (exists in one of their super types)

```
entity Person {
    name: String
}
entity Employee extends Person {
    name: String //Error: Feature name is not unique
}
```

Exercise: Unit Testing

Implement an unit test to check if the type of a self reference is the same as the entity in which it is declared

```
entity MyEntity {
      parent: MyEntity
}
```

Implement an unit test to check if the warning is given when the following snippet is used:

```
entity noCapitol {
      parent: noCapitol
}
```

Tip: Use the `ValidationTestHelper`

Extra Exercises

Implement defaults for entity features and datatypes

- `datatype String = "UNDEFINED“`　　　　`age : Integer = -1`

Implement an expression with Xtext

- `Show Blog.posts.comments.content`

Implement a quick fix

Implement a formatter

Implement scoping

Create a standalone runnable jar

Implement your own DSL idea