Rapidly implementing Java-like languages with Xtext and Xbase

Lorenzo Bettini

Dipartimento di Informatica, University of Turin, Italy
itemis Schweiz consultant
Motivations for Xbase

• Developing a textual DSL is easy with Xtext
  – Especially for “structures”
  – Highly customizable (by Injection)

• What about Expressions and Behavior?
  – More complex
  – Recurrent task
  – Harder to validate (e.g., type checking)
Xbase
a reusable expression language

• Expression language embeddable in your DSL
  – Grammar
  – Linking and Scoping
  – Java Type System (including generics)
    • Access to Java types
    • Automatic import handling
    • Seamlessly access any existing Java library
  – Type Inference
  – Code generation
  – Eclipse tooling
  – Debugger
Xbase expressions

- Standard expressions
  - Arithmetic
  - Logical
  - Comparisons, etc.
- Conditionals, Loops, Enhanced switch
- OO expressions
  - Object creation
  - Field access and Method invocation
  - Casts, etc.
- Lambda expressions
- Extension methods
Use Xbase grammar in your DSL

```plaintext
grammar org.eclipse.xtext.example.domainmodel.Domainmodel
with org.eclipse.xtext.xbase.Xbase

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

DomainModel:
importSection=XImportSection?

elements+=Entity*;

Entity:
'entity' name=ValidID ('extends' superType=JvmParameterizedTypeReference)? '{'
  features+=Feature*
'}';

Feature: Property | Operation;

Property: name=ValidID ':' type=JvmTypeReference;

Operation:
'op' name=ValidID '
  (' (params+=FullJvmFormalParameter (',') params+=FullJvmFormalParameter)*))? '
  (':' type=JvmTypeReference)?
  body=XBlockExpression;
```

Inherit from Xbase
Use imports
Refer to Java types
Syntax for parameters
Syntax for Expressions
Example

```java
import java.util.List

definition Person {
    firstName : String
    lastName : String
    friends : List<Person>

    op getFullName() {
        firstName + " " + lastName
    }

    op sortedFriends() : List<Person> {
        friends.sortBy[fullName]
    }
}
```
Use Xbase infrastructure

Grammar Inheritance

My DSL

JvmModelInferrer

Just map your DSL model into the Java model

Xbase grammar

Xbase Runtime

Linker
Type System
Validation
Code Generation
Eclipse Integration
Debugger

JvmModel

JvmModelInferrer
Mapping to Java

```java
import java.util.List

@entity Person {
  firstName : String
  lastName : String
  friends : List<Person>

  @op getFullName() {
    firstName + " " + lastName
  }

  @op getSortedFriends() : List<Person> {
    friends.sortBy[fullName]
  }
}
```

This gives the Xbase expression a context
Automatic Linking

```java
entity Person {
    firstName : String
    lastName : String

    op getFullName() {
        firstName + " " + this.lastName
    }
}

entity Student extends Person {
    op getFullName() {
        "Student: " + super.getFullName()
    }
}
```
Automatic Validation
(Type Checking)

The method's return type is the expected type of the body.
Automatic Code Generation
Implementing the mapping

class DomainmodelJvmModelInferrer extends AbstractModelInferrer {

@Inject extension JvmTypesBuilder
@Inject extension IQualifiedNameProvider

def dispatch infer(Entity entity, IJvmDeclaredTypeAcceptor acceptor, boolean prelinkingPhase) {
    acceptor.accept(
        entity.toClass( entity.fullyQualifiedName )
    ).initializeLater [ 
        if (entity.superType != null) 
            superTypes += entity.superType.cloneWithProxies
        members += entity.toConstructor() []
        for ( f : entity.features ) {
            switch f {
                Property : { 
                    members += f.toField(f.name, f.type)
                    members += f.toGetter(f.name, f.type)
                    members += f.toSetter(f.name, f.type)
                }

                Operation : { 
                    members += f.toMethod(f.name, f.type ?: inferredType) [ 
                        for (p : f.params) { 
                            parameters += p.toParameter(p.name, p.parameterType)
                        }
                        body = f.body
                    ]
                }
            }
        }
    ]
}
Implementing a DSL for the Java Platform

- Give the semantics through translation in Java
  - Perform mapping in the JvmModelInferrer
  - The typing is performed on the mapped model
  - The code generation relies on the mapped model
Additional Validation

- Xbase only validates its expressions
- The “structural” part is up to you
  - Check no duplicate entities
  - Check no duplicate properties
  - ...
Extending Xbase

- Extend the grammar of expressions
  - Provide the typing for your expressions
  - Scoping for your expressions
  - Validation for your expressions
  - Compilation for your expressions
  - ...
Xtraitj

Pure Traits for the Java Platform

http://xtraitj.sourceforge.net
Traits

• Introduced in Smalltalk/Squeak
• A mechanism for fine-grained code reuse
• Overcome the limitations of class-based inheritance
• A trait is a set of methods:
  – completely independent from any class hierarchy and
  – can be flexibly used to build other traits or classes by means of a suite of composition operations.
Distinct Roles

- Subtyping on interfaces only
- Classes only play the role of object generators
  - class-based inheritance is not present
- Traits only play the role of units of code reuse
  - they are not types
Some examples

```java
trait T1 {
    // required field
    String s;

    // required method
    void print(Object o);

    // provided method
    void m() {
        print(this.s)
    }
}

trait T2 {
    void print(Object o) {
        System.out.println(o)
    }
}

class C uses T1, T2 {
    String s = "aString";
}

Use it in your Java programs

import my.traits.C;

public class Main {
    public static void main(String[] args) {
        new C().m();
    }
}
```
Demo!