Making Xbase Look Like Java

Lorenzo Bettini
Dip. Informatica, Univ. Torino, Italy,
itemis GmbH, Zurich, Switzerland

Joint work with
Pierluigi Crescenzi
Dip. Ing. Inf., Univ. Firenze
Based on the paper

Java--meets Eclipse

An IDE for teaching Java following the object-later approach

Lorenzo Bettini and Pierluigi Crescenzi

ICSOFT-PT 2015

July, Colmar, France

To appear
Object-Later Approach

• At the beginning of a programming course
• Teach programming focusing on
  – Algorithmic aspects
  – Primitive types
  – Basic control structures
• Teach OOP only at a later stage
As observed in (Koulouri et al., 2014),
- even if Java has been widely used as a FPL,
- its complexity “may be overwhelming for learners”.

In particular
- it is “heavily coupled with object-oriented concepts”,
- making more difficult to implement an object-later strategy.

A typical example of such complexity
- the implementation of “hello, world” program
public class HelloWorld {

    public static void main(String[] args) {
        System.out.println("Hello World!");
    }
}

The (not so) simple Hello World
public class HelloWorld {

    public static void main(String[] args) {

        System.out.println("Hello World!");

    }

}

In the end, that's the only important statement
Java--

- Java without OOP constructs
  - No classes
  - Only functions/procedures
  - The rest of the program are the main's expressions

C : C++ = JavaMM: Java. 
In Proceedings of PPPJ, ACM, pages 75–78.
Java-- at the University of Florence

“Drops of Java
an introduction to procedural and object-oriented programming”

It has been used for many years for teaching first programming course at the University of Florence
Java--, old implementation

- Generate Java code
- Call the Java compiler
- Show errors in a tab
- No IDE at all (only syntax highlighting)!

```java
int max(int a, int b) {
    if (a>b)
        return a;
    else
        return b;
}
System.out.println(max(7,11));
```
What we aim at

- Full Eclipse IDE features
  - Automatic building
  - Error reporting while editing
  - Content Assist
  - Debugger

- Technology?
  - Xtext/Xbase
  - What else? ;-)
Why an IDE?

- IDE vs non-IDE war!
- **We support IDEs**
  - Students get familiar with IDE tooling right away
  - Switching to Java tooling will be easier (immediate)
  - Most students not using an IDE fail exams because
    1) They compile the code ONLY after writing the whole program
    2) Their code does not compile
    3) They're not able to fix the problems reported by the command line compiler
A Java-- program

javammProgram:
    javammMethods+=javammMethod*
    main=Main;

javammMethod:
    =>({{javammMethod} type=JvmTypeReference name=ValidID '
    (params+=FullJvmFormalParameter (','
    params+=FullJvmFormalParameter)*)? '})
    body=XBlockExpression;

Main returns XBlockExpression:
    {Main} (expressions+=javammStatementOrBlock ';;'*);
Implement the model inferrer

```python
def dispatch void infer(JavammProgram program,
                         IJvmDeclaredTypeAcceptor acceptor, boolean isPreIndexingPhase) {
    acceptor.accept(program.toClass(program.fullyQualifiedName))
    for (m : program.javammMethods) {
        members += m.toMethod(m.name, m.type)
        static = true
        for (p : m.params)
            parameters += p.toParameter(p.name, p.parameterType)
        body = m.body
    }
}

// the class gets one main method
members += program.main.toMethod('main', typeRef(Void.TYPE))
parameters += program.main.toParameter("args", typeRef(String).addArrayTypeDimension)
static = true
body = main
```
What we get in the end
Wait a minute... This is not Xbase

double avg(int[] a) {
    double r = 0.0;
    for (int i = 0; i < a.length; i++) {
        r = r + a[i];
    }
    return r / a.length;
}

int[] a = {1, 2, 3, 4};
Making Xbase look like Java

- We need to tweak the Xbase grammar so that it handles pure Java expressions
- Xbase is more permissive
  - E.g., does not distinguish between ' ' and “ ” literals
  - Terminating ';' are optional
- Some control structures have to customized
  - switch, for loops, etc.
- Remove Xbase lambdas
  - We need [ ] for array access
- Typing for new expressions
  - Like array access

Be prepared to use **Syntactic Predicates** a lot to deal with ambiguities in the grammar.

Xbase itself uses them.
Reuse as much as we can

- Xbase's other components act on the AST
- As long as your custom grammar rules extend Xbase rules
  - (i.e., your model classes extend Xbase's model classes)
  - All the other parts of Xbase will work out of the box
    - Typing, validation, code generator
- Example: cast expression
Example: cast expression

XCastedExpression returns XExpression:
  XPostfixOperation
    (=>({XCastedExpression.target=current} 'as') type=JvmTypeReference)*

Becomes in Java--

XCastedExpression returns XExpression:
  =>({XCastedExpression} '(' type=JvmTypeReference ')'
     target=XExpression)

... But the Xbase type system, the validator, and the code generator will work as before.

In the AST they have exactly the same structure
Conditional Expression (e ? e1 : e2)

- In Xbase an **if** statement is an expression
  - We just make JavammConditionalExpression extend XIfExpression

\[
\text{XAssignment returns } \text{XExpression} : \\
\ldots \mid \\
\text{XOrExpression (}
\quad \Rightarrow\{{\text{JavammConditionalExpression}.if=\text{current}}\} \ '?')
\quad \text{then= } \text{XExpression} \ '':' \ \text{else= } \text{XExpression}
\mid \\
\Rightarrow\{{\text{XBinaryOperation}.leftOperand=\text{current}}\}
\quad \ldots
\)?;\]
Variable Declaration in Xbase

XVariableDeclaration returns XExpression:
   {XVariableDeclaration}
   (writeable?='var'|'val')
   (=>(type=JvmTypeReference name=ValidID) | name=ValidID)
   ('=' right=XExpression) ;

- The default in Xbase is “not writable” (final)
- The default in Java is “not final”
- First attempt:

XVariableDeclaration returns XVariableDeclaration:
   {XVariableDeclaration}
   type=JvmTypeReference name=ValidID
   ('=' right=XExpression) ;
Drawbacks:

- We don't handle Java final variables
- We need to customize all the parts of the validator that check `XVariableDeclaration.isWritable()`
- We need to customize all the parts of the generator that check `XVariableDeclaration.isWritable()`
- We don't handle Java several variable declarations,
  - `int i = 0, j = k;`
Another attempt

XVariableDeclaration returns XExpression:
{XVariableDeclaration}
(writeable?='var'|'val')
(=>(type=JvmTypeReference name=ValidID) | name=ValidID)
('=' right=XExpression)?;

• We add other fields in our rule:

XVariableDeclaration returns XVariableDeclaration:
{JavammXVariableDeclaration}
final?='final'?
(type=JvmTypeReference name=ValidID) ('=' right=XExpression)?
(=>', additionalVariables+=JavammAdditionalXVariableDeclaration)*;

JavammAdditionalXVariableDeclaration returns XVariableDeclaration:
{JavammAdditionalXVariableDeclaration}
name=ValidID ('=' right=XExpression)?;
But still:

- We need to customize all the parts of the validator that check `XVariableDeclaration.isWritable()`
- We need to customize all the parts of the generator that check `XVariableDeclaration.isWritable()`
Switch to imported Ecore model

- Procedure to follow:

- Use in the mwe2
  - org.eclipse.emf.mwe2.ecore.EcoreGenerator
  - instead of
  - org.eclipse.xtext.generator.ecore.EMFGeneratorFragment

- Then, provide custom implementation of the Java model class
The ImplCustom.java pattern

- If the EcoreGenerator fragment finds a
  - MyClassImplCustom.java
- Then the EMF Factory will instantiate
  - MyClassImplCustom
  - Instead of
  - MyClassImpl
- So you can add/customize Java methods in ImplCustom
- An alternative to the EMF @generated NOT pattern
- This is already used by Xbase itself.
In the custom implementation

```java
public class JavammXVariableDeclarationImplCustom extends XVariableDeclarationImplCustom implements JavammXVariableDeclaration {

    @Override
    public boolean isWriteable() {
        // implement isWritable in terms of isFinal
        return !isFinal();
    }

    // manually copy everything else from JavammXVariableDeclarationImpl

    ● No need to customize the validator nor the generator
    ● Caveat:
      - You need to manually copy the rest from the generated Impl class
```
Implement array access

- We need to get rid of XClosure rule calls in all the grammar rules

XFeatureCall `returns` XExpression:
{XFeatureCall}
('<' typeArguments+=JvmArgumentTypeReference (',' ... featureCallArguments+=XClosure?;)
XConstructorCall `returns` XExpression:
{XConstructorCall}
'new' constructor=[types::JvmConstructor|QualifiedName] ... arguments+=XClosure?;
XLiteral `returns` XExpression:
XCollectionLiteral | XClosure | XBooleanLiteral | XNumberLiteral | XNullLiteral | XStringLiteral | XTypeLiteral
;
ArrayAccess as right-hand side exp

- We need to hook it in the right place in the grammar
- In Xbase we have

```plaintext
XPostfixOperation returns XExpression:
  XMemberFeatureCall =>({XPostfixOperation.operand=current}
  feature=[types::JvmIdentifiableElement|OpPostfix])?;

XMemberFeatureCall returns XExpression:
  XPrimaryExpression
  (=>({XAssignment.assignable=current} ('.'|explicitStatic?="::") ...;

- So we hook it into XPostfixOperation rule...
```
ArrayAccess as right-hand side exp

XPostfixOperation returns XExpression:
XMemberFeatureCall =>>({XPostfixOperation.operand=current} 
   feature=[types::JvmIdentifiableElement|OpPostfix])?

Becomes in Java--

XPostfixOperation returns XExpression:
XMemberFeatureCall
  ( 
    =>>({XPostfixOperation.operand=current} 
         feature=[types::JvmIdentifiableElement|OpPostfix])
  | 
  =>$>({JavammArrayAccessExpression.array=current} 
      '][' indexes+=XExpression ']'(=>'][' indexes+=XExpression '']')* 
  )?
Typing for array access expression

- Compute the type of an array access expression by removing array dimensions from the original array type.
- The Xbase validator will then be able to check for possible type mismatch errors.

```csharp
// each array access removes one array dimension
// from the array type
int[][][] a;

int[] i = a[0]; // a[0] has type int[]
int j = a[0][0]; // a[0][0] has type int
int k = a[0]; // TYPE MISMATCH ERROR
```
def protected _computeTypes(JavammArrayAccessExpression arrayAccess, ITTypeComputationState state) {
    val actualType = // given a[i][j][k], first get the type of a
                    state.withNonVoidExpectation.
                    computeTypes(arrayAccess.array).actualExpressionType
    val type = componentTypeOfArrayAccess(arrayAccess, actualType, state)
    state.acceptActualType(type)

    checkArrayIndexHasTypeInt(arrayAccess, state);
}

private def componentTypeOfArrayAccess(JavammArrayAccess arrayAccess, LightweightTypeReference type, ITTypeComputationState state) {
    var currentType = type
    for (index : arrayAccess.indexes) { // remove array dimension
        if (currentType instanceof ArrayTypeReference)
            currentType = currentType.componentTypeComponentType
        else { // also store error diagnostic in the state (next slide)
            return currentType
        }
    }
    return currentType
}
Custom TypeComputer (cont'd)

• Add possible diagnostic to the type computation state
• The Xbase validator will later generate error markers
  – No need to check that in your validator

```java
if (currentType instanceof ArrayTypeReference) {
    currentType = currentType.componentType
} else {
    val diagnostic = new EObjectDiagnosticImpl(
        Severity.ERROR,
        JavammValidator.NOT_ARRAY_TYPE,
        "The type of the expression must be an array type but it resolved to " + currentType.simpleName,
        arrayAccess,
        featureForError,
        -1,
        null);
    state.addDiagnostic(diagnostic);
    return currentType
}
```
ArrayAccess as left-hand side exp

• Customize all the Xbase rules
  – Where array access can appear as left-hand side expression
    • Assignment
  • Example:

```
XAssignment returns XExpression :
=>({JavammXAssignment}
  feature=[types::JvmIdentifiableElement|FeatureCallID]
  '][' indexes+=XExpression ']['('][' indexes+=XExpression ']]')*
  OpSingleAssign) value=XAssignment
| ...
```
You need to customize ExpressionArgumentFactory
- This tells Xbase validator
  - The type expectations for feature call arguments
  - e.g., in an assignment, the expected type of the right-hand expression
- In our case,
  - when the left-hand side of an assignment accesses an array with indexes

```java
int[][] a;

a[0] = e1;    // e1 must have type type int[]
a[0][0] = e2; // e2 must have type int
a[0] = 0;     // TYPE MISMATCH ERROR
```
public class JavammExpressionArgumentFactory extends ExpressionArgumentFactory {

    @Override
    public IFeatureCallArguments createExpressionArguments(XExpression expression, AbstractLinkingCandidate<?> candidate) {

        if (expression instanceof JavammXAssignment) {
            AssignmentFeatureCallArguments assignmentFeatureCallArguments = (AssignmentFeatureCallArguments) super.createExpressionArguments(expression, candidate);
            JavammXAssignment assignment = (JavammXAssignment) expression;
            LightweightTypeReference featureType = assignmentFeatureCallArguments.getDeclaredType();
            // if it's an array access we must take the array component type
            if (featureType instanceof ArrayTypeReference && !assignment.getIndexes().isEmpty()) {
                return new AssignmentFeatureCallArguments(assignment.getValue(), getComponentType(featureType, assignment));
            } else {
                return assignmentFeatureCallArguments;
            }
        }

        return super.createExpressionArguments(expression, candidate);
    }
}...
Customize XbaseCompiler

- So that it can handle array access expressions
- Not shown here, see the code
Dealing with statements

- In Xbase, EVERYTHING is an expression
  - The terminating ';' is optional
- This is not true in Java
  - Only some statements are also expressions
  - The terminating ';' is mandatory
    - But not on control statements
    - Apart from do...while
  - Additional ';' are considered as empty statements
Dealing with statements

JavammStatementOrBlock returns XExpression:
=>XBlockExpression |
JavammSingleStatement;

JavammSingleStatement returns XExpression:
JavammSemicolonStatement |
XSwitchExpression |
XIfExpression |
XForLoopExpression |
XBasicForLoopExpression |
XWhileExpression;

JavammSemicolonStatement returns XExpression:
( 
  JavammBranchingStatement |
  XExpressionOrVarDeclaration |
  XDoWhileExpression |
  XReturnExpression |
) ->';';"
/**
 * The syntactic predicate on XBlockExpression is required to avoid ambiguity with array literal.
 */

JavammStatementOrBlock returns XExpression:
  =>XBlockExpression |
    JavammSingleStatement
;
Dealing with ';'?

JavammSemicolonStatement returns XExpression:
  (  
    JavammBranchingStatement |  
    XExpressionOrVarDeclaration |  
    XDoWhileExpression |  
    XReturnExpression  
  ) -> ';'?
;

Why not simply impose the mandatory ';' in the grammar?...

JavammSemicolonStatement returns XExpression:
  (  
    JavammBranchingStatement |  
    XExpressionOrVarDeclaration |  
    XDoWhileExpression |  
    XReturnExpression  
  ) -> ';'?
;
JavammSemicolonStatement returns XExpression:

( JavammBranchingStatement |
  XExpressionOrVarDeclaration |
  XDoWhileExpression |
  XReturnExpression |
  ) ';;' ← DON'T DO THAT

• You won't be able to parse a statement until there's a ;
  – Error reported will not be clear
  – Content assist won't work

• You must
  – Be able to parse incomplete programs
  – **Loose grammar, Strict Validation**
    • See Sebastian's “Xtext Best Practices”
def private checkMissingSemicolon(EObject e) {
    if (!e.hasSemicolon) {
        error(
            'Syntax error, insert ";" to complete Statement',
            ...
        }
    }
}

def private hasSemicolon(EObject object) {
    return NodeModelUtils.getTokenText(
        NodeModelUtils.findActualNodeFor(object)).
        endsWith(";");
}

• Content assist will work
• Errors will be crystal clear
• You users will be happier :-)}
Additional ';

- Deal with possible additional ';;' in the block expression rule

```java
/*
 * The ';;' consumes possible additional semicolons which is legal
 * in Java.
 * The required ';;' is taken care of in the
 * JavammSemicolonStatement rule
 */
XBlockExpression returns XExpression:
{XBlockExpression}
'{'
  (expressions+=JavammStatementOrBlock ';;')*
'}';
```
Other customizations to do

• Please, have a look at the code
  – Branching statements: \texttt{break, continue}
  – Distinguish between:
    • Character literals ''
    • String literals “ ”
Other Features

- Automatic imports
- All Java types
  - E.g., collections
  - Generics
- The students can get familiar with them before switching to Java
Debugging Java-- code

- We get it for free from Xbase

```java
/**
 * Computes the average of the numbers specified in the passed array.
 */

double avg(int[] a) {
    double r = 0.0;
    for (int i = 0; i < a.length; i++) {
        r = r + a[i];
    }
    return r / a.length;
}

int[] a = { 1, 2, 3, 4 }; System.out.println(avg(a));
```
Educational value of debugging

- Show the difference between references and values
- The two arrays have the same elements
- But they are different identifiers
- So == fails
Educational value of debugging

- For recursive functions
  - Show activation records
  - Recursion has some overhead
Code

- https://github.com/LorenzoBettini/javamm
- Binaries:
  - Update site
  - Eclipse distributions
Future Work

- Implement Java 8 lambdas
  - Map them to Xbase lambdas
- Extract the expression part from Java--
- So that it can be reused in other DSLs
  - Just like Xbase
  - Maybe `Xjava` or `Xjavaexp`…
  - In DSLs where dealing with (possibly legacy) Java expressions is a strict requirement
Competition: Win the book! :-)  

- All you have to do is...  
- Tweet something about this presentation with the hashtag #XtextCON  
- Mention me: @lorenzo_bettini  
- Starting from now...  
- The first two tweets WIN!  

THANKS!