Type System Frameworks for Xtext

Lorenzo Bettini\textsuperscript{1}    Dietmar Stoll\textsuperscript{2}

joint work with

Serano Colameo\textsuperscript{2}    Markus Völter\textsuperscript{3}

\textsuperscript{1}Dipartimento di Informatica, Università di Torino, Italy
\textsuperscript{2}itemis GmbH, Zurich, Switzerland
\textsuperscript{3}itemis AG, Stuttgart, Germany

Eclipse Day Florence, May 4th, 2012
Are Typesystems Useful?

- Advent of DSLs and DSL tools
- Writing languages is easy, even for complex domains
- Complex languages $\Rightarrow$ complex constraint checking
- Most languages have a concept of **types**: reusable building blocks for constraints
1. Introduction
2. Case Study
3. Implementation in Plain Xtext
4. Implementation in Xbase
5. Implementation in Xsemantics
6. Implementation in XTS
7. Comparison
8. Conclusion
9. Appendix
name : `string`;
greeting = "Hello " + name + "!";

Common type system tasks:
- Assign types to language elements
- Infer types of complex expressions
- Is a type conformant to another type?
A language to specify a GUI to edit entities
Case Study

```plaintext
entity Person {
  name : string;
  firstName : string;
  age : int;
  weight : float;
  likesCake : bool;
  isAdult = age > 10 + 8;
  greeting = "Hello " + firstName + " !";
}
```

```plaintext
form PersonFormWithError edits Person {
  text(10) -> name validate widgetcontent;
  text(10) -> isAdult;
  checkbox -> firstName;
  checkbox -> greeting
}
```

Checkbox widget may only refer to boolean attributes.
3 Implementation in Plain Xtext
Implementation in Plain Xtext

- Implementation in Xtend
  - Type provider - recursive computation
  - Type conformance computer
  - Xtext validator hook
The Plus Operator in Plain Xtext

* Type Provider

```python
def Type getType(EOBJECT e) {
  ...
  switch e {
    ...
    IntLiteral: ...createIntType
    Plus: mostGeneral(e.left.type, e.right.type)
    ...
  }

def Type getExpectedType(EOBJECT e) {
  ...
  switch e {
    ...
    Plus: mostSpecific(string, e.type)
    ...
```
The Plus Operator in Plain Xtext

- Type Conformance

...  def dispatch isAssignable(Type a, Type b) { ... }
...
  def dispatch isAssignable(StringType left, NumberType right) { true }
  ...

Implementation in Xbase

1. Introduction
2. Case Study
3. Implementation in Plain Xtext
4. Implementation in Xbase
5. Implementation in Xsemantics
6. Implementation in XTS
7. Comparison
8. Conclusion
9. Appendix
Implementation in Xbase

- Entity attributes \(\Rightarrow\) Java types, Xbase expressions

- Infer Java type for Entities and Forms with `JvmModelInferrer`

```java
def dispatch void infer(Entity element, IJvmDeclaredTypeAcceptor acceptor,
                        boolean preIndexingPhase) {
    acceptor.accept(element.toClass(element.fullyQualifiedName).initializeLater [
        documentation = element.documentation
        if (element.superType != null)
            superTypes += element.superType.cloneWithProxies
        for (attribute : element.attributes) {
            val getter = attribute.toGetter(attribute.name, attribute.getJvmType)
            members += getter
            if (attribute.expr != null) {
                getter.body = attribute.expr
            } else {
                members += attribute.toField(attribute.name, attribute.getJvmType)
                members += attribute.toSetter(attribute.name, attribute.getJvmType)
            }
        }
    ]
}
```
Xsemantics

1. Introduction
2. Case Study
3. Implementation in Plain Xtext
4. Implementation in Xbase
5. Implementation in Xsemantics
6. Implementation in XTS
7. Comparison
8. Conclusion
9. Appendix
Xsemantics

- A DSL for writing
  - type systems (static semantics)
  - reduction rules (dynamic semantics)
  - interpreters
  - general relation rules

for languages implemented in Xtext

- A system definition is a set of judgment rules which have a conclusion and a set of premises (relying on Xbase for the syntax)

- Thought to be used by people who are familiar with formal type systems and operational semantics

- Aims at providing a syntax which is close to the way deduction rules are written in a formal setting
Define your judgments

```java
system org.typesys.xsem.guidsl.xsemantics.TypeSystem

import org.typesys.xsem.guidsl.xsemGuiDsl.*

judgments {
    type |- Typable typable : output Type
    // whether {@code right} is assignable to {@code left}
    isAssignable |- Type left <∼ Type right
    // computes the most general type between {@code first} and {@code second}
    mostGeneral |- Type first ∼∼ Type second |> output Type
}
```
Similarities

\[ \Gamma \vdash \text{true} : \text{boolean} \]

\[ \Gamma \vdash \text{attr} : T \]

\[ \Gamma \vdash \text{ref attr} : T \]

**axiom** BooleanLiteralType

\[ G \vdash \text{BooleanLiteral } \text{lit} : \text{XsemGuiDslFactory::eINSTANCE.createBooleanType} \]

**rule** AttributeRefType

\[ G \vdash \text{AttributeRef } \text{attrRef} : \text{Type } \text{type} \]

**from** { 

\[ G \vdash \text{attrRef.attr} : \text{type} \]

}
Similarities

\[
\begin{align*}
\Gamma \vdash \text{exp} : \text{string} & \quad \Gamma \vdash \Gamma(\text{widgetcontent}) : T \\
\Gamma \vdash \text{lengthOf}(\text{exp}) : \text{int} & \quad \Gamma \vdash \text{widgetcontent} : T
\end{align*}
\]

**rule** LengthOfType

\[
G \mid \mid \text{LengthOf len} : \text{XsemGuiDslFactory::eINSTANCE.createIntType}
\]

**from**

\[
\begin{align*}
G \mid \mid \text{len.expr} : \text{var String Type stringType}
\end{align*}
\]

**rule** FieldContentType

\[
G \mid \mid \text{FieldContent fieldContent} : \text{Type type}
\]

**from**

\[
\begin{align*}
G \mid \mid \text{env}(G, 'widgetcontent', \text{Attribute}) : \text{type}
\end{align*}
\]
Specific rules for validation

checkrule ValidateMustBeBoolean for Widget widget from {
    widget.validate == null
    or
    'widgetcontent' <= widget.attr |- widget.validate : var BooleanType boolType
}

checkrule ValidateTextWidgetAttributeNotBoolean for TextWidget widget from {
    'widgetcontent' <= widget.attr |- widget.attr : var Type attrType
    !(attrType instanceof BooleanType)
}
The generated validator

```java
entity Person {
    name : string;
    firstName : string;
    age : int;
    weight : float;
    likesCake : bool;
    isAdult = age > 18;
    greeting = "Hello " + firstName + " " + name;
    testExpr = -2 - 2;
}

entity Gym {}

entity School {
    janitor: Person = new Gym; // error
    head : Person = new ExtTeacher;
    gym = new Gym;
    gym2 : Gym = new Gym;
}
```

7 errors, 0 warnings, 0 others

- Errors (7 items)
  - cannot assign Gym to Person
Automatic trace generation:
- for rule successful applications
- for rule failures

Useful for testing and debugging

final result provided by rule MyRule
rule 1 used by MyRule to get to the result
  rule 2 used by rule 1
    rule 3 used by rule 2
    ...
rule 1a used by MyRule to get to the result
rule 2a used by rule 1a
...
A tool for the “formal” designer

- Write the theory of a language
  - Write the type system
  - Write the semantics
  - Prove that the latter is consistent with the former

- With Xtext you can develop a quick prototype

- With Xsemantics you can smoothly encode the formal system in the implementation

- Help in filling the gap between the theory and the implementation
Xtext Typesystem (XTS)

1. Introduction
2. Case Study
3. Implementation in Plain Xtext
4. Implementation in Xbase
5. Implementation in Xsemantics
6. Implementation in XTS
7. Comparison
8. Conclusion
9. Appendix
Xtext Typesystem (XTS)

- A Java framework to declaratively specify type system rules
- DSL to generate the Java code
- If no suitable declarative abstraction is available, Java code can be added manually.
- ITypesystem interface with methods for
  - calculating the type of model elements
  - for comparing types for compatibility and subtyping relationships
Setting up

typesystem org.typesys.xts.guidsl.typesys.GuiDIsTypesystem
core file "platform:/.../GuiDsl.ecore"
language package org.typesys.xts.guidsl.guiDsl.GuiDslPackage

The platform URI is for the Ecore file (the metaclasses for which we want to specify the type system rules)
Specifying types

section "Types"
  typeof Type+ -> clone
  subtype IntType base FloatType

section "Literals"
  typeof StringLiteral -> StringType
  typeof BooleanLiteral -> BooleanType
  typeof NumberLiteral -> javacode

Recursive computation

  typeof AttributeRef -> feature attr
Specifying types

characteristic COMPARABLE {
    IntType, FloatType, BooleanType, StringType
}

characteristic NUMERIC {
    IntType, FloatType
}

typeof Comparison -> BooleanType {
    ensureType left :<=: char(COMPARABLE)
    ensureType right :<=: char(COMPARABLE)
    ensureCompatibility left :<=>: right
}

typeof Minus -> common left right {
    ensureType left :<=: char(NUMERIC)
    ensureType right :<=: char(NUMERIC)
    ensureCompatibility left :<=>: right
}
Specifying types

// text widgets may only refer to non-boolean attributes
typeof TextWidget -> feature attr {
    ensureType length :<=: IntType
    ensureType attr :<=: StringType, IntType, FloatType
    ensureType validate :<=: BooleanType
}

// checkbox widgets may only refer to boolean attributes
typeof CheckBoxWidget -> feature attr {
    ensureType attr :<=: BooleanType
    ensureType validate :<=: BooleanType
}
Which approach to use when? I

Criteria to help in deciding which approach might be good for a particular use case:

- Context, e.g. coupling with Java
- Expressivity
- Verbosity/conciseness
- Customizability
- Additional features
- Target audience
- Learning curve
- Documentation
- Support
Which approach to use when? II

### Coupling of the DSL with Java

<table>
<thead>
<tr>
<th>Java Type System</th>
<th>Other Type Systems</th>
</tr>
</thead>
<tbody>
<tr>
<td>Xbase</td>
<td>Plain</td>
</tr>
<tr>
<td></td>
<td>Xsem</td>
</tr>
<tr>
<td></td>
<td>XTS</td>
</tr>
</tbody>
</table>
### Which approach to use when? III

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Plain</td>
<td>Java</td>
</tr>
<tr>
<td>Xbase</td>
<td>Xbase library</td>
</tr>
<tr>
<td>Xsem</td>
<td>Similar to Xbase + specific syntax</td>
</tr>
<tr>
<td>XTS</td>
<td>EMF feature access + specific syntax</td>
</tr>
</tbody>
</table>
Conclusion

- Typesystem frameworks save time for common tasks
- Typesystem framework applications
  - DSL to model and test cooling algorithms for refrigerators
  - Mapping to and from ISO20022 (financial domain)
    - e.g. PostalAddress6 based on PostalAddress, Max140Text based on Name
  - Service description languages to generate artifacts conforming to an existing framework
    - embed IDL, WSDL, and Java types
  - Implementations of formally described languages
    - traits, mixins, deltas
  - C-like language for hearing aid configuration
- Advanced tasks relying on typesystems
  - Interpreters, dynamic type computation, ...
References

- Xsemantics - http://xsemantics.sourceforge.net/
- XTS - http://code.google.com/a/eclipselabs.org/p/xtext-typesystem/