Softlang Working Group
http://softlang.wikidot.com/

MegaL Traceabiltiy Recovery

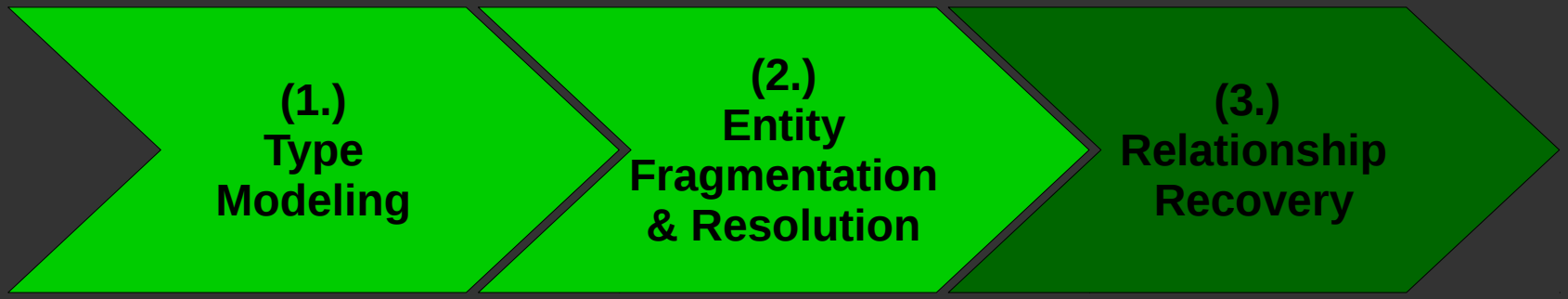
https://github.com/maxmeffert/megal-tr

A Fragment Model & *ANTLR* Backed Plugins

Meeting 2016-06-28

University of Koblenz-Landau
Maximilian Meffert

Quick Recap



Development Process

(1.) Models the fragment ***types*** of the domain

(2.) Extracts the fragments from a domain instance (*Fragmentation*) & resolves further specializations if necessary (*Resolution*)

(3.) Recovers relationships between fragments
[not discussed now]

University of Koblenz-Landau
Maximilian Meffert

Quick Recap

```
JavaFragment < Fragment

// type declarations
JavaClass < JavaFragment
JavaInterface < JavaFragment
JavaEnum < JavaFragment

// member declarations
JavaInnerClass < JavaFragment
JavaMethod < JavaFragment
JavaConstructor < JavaMethod
JavaField < JavaFragment
JavaAnnotation < JavaFragment
```

```
public class Foo {

    static public class Bar {

        private void getBar () {

        }

    }

    private String bar;

    public String getBar() {
        return bar;
    }

    public void setBar(String bar) {
        this.bar = bar;
    }

}
```

Quick Recap

```
aJavaFile: File
aJavaFile elementOf Java
aJavaFile = 'workspace:/org.softlang.megal.plugins/input/Foo.java'

aJavaFile.Foo#0: JavaClass
aJavaFile.Foo#0 partOf aJavaFile
aJavaFile.Foo#0 = 'file:/.../Foo.java#/0/Foo/JavaClass'

aJavaFile.Foo#0.Bar#0: JavaInnerClass
aJavaFile.Foo#0.Bar#0 partOf aJavaFile.Foo#0
aJavaFile.Foo#0.Bar#0 = 'file:/.../Foo.java#/0/Foo/JavaClass/0/Bar/JavaInnerClass'

aJavaFile.Foo#0.Bar#0.getBar#0: JavaMethod
aJavaFile.Foo#0.Bar#0.getBar#0 partOf aJavaFile.Foo#0.Bar#0
aJavaFile.Foo#0.Bar#0.getBar#0 = 'file:/.../Foo.java#/0/Foo/JavaClass/0/Bar/JavaInnerClass/0/getBar/JavaMethod'

aJavaFile.Foo#0.bar#1: JavaField
aJavaFile.Foo#0.bar#1 partOf aJavaFile.Foo#0
aJavaFile.Foo#0.bar#1 = 'file:/.../Foo.java#/0/Foo/JavaClass/1/bar/JavaField'

aJavaFile.Foo#0.getBar#2: JavaMethod
aJavaFile.Foo#0.getBar#2 partOf aJavaFile.Foo#0
aJavaFile.Foo#0.getBar#2 = 'file:/.../Foo.java#/0/Foo/JavaClass/2/getBar/JavaMethod'

aJavaFile.Foo#0.setBar#3: JavaMethod
aJavaFile.Foo#0.setBar#3 partOf aJavaFile.Foo#0
aJavaFile.Foo#0.setBar#3 = 'file:/.../Foo.java#/0/Foo/JavaClass/3/setBar/JavaMethod'
```

Fragmentation Result

Qualified Fragment Names

`aJavaFile.Bar#0.[…].doStuff#666`

Name of the declared entity
Short name of the fragment
Index of the fragment in its compound

Where a Qualified Fragment Name (QFN) conforms to:

```
QFN         : ENAME Fragment+
Fragment    : '.' FNAME '#' INDEX
ENAME       : \S+
FNAME       : \w+
INDEX       : \d+
```


University of Koblenz-Landau
Maximilian Meffert

Qualified Fragment Names

- ***QFNs*** are used as identifies for the derived entities
- ***QFNs*** depict *parthood* relationships
  - So ***partOf***-reasoning is triggered in the next evaluation cycle
- ***Indexes*** depict the position of fragments in their respective compounds

Fragment URIs

Generic URI Form:
`scheme:[//[user:password@]host[:port]][/]path[?query][#fragment]`

Fragment URI Form:
scheme://location#fragment

Where **fragment** conforms to:

```
Fragment  : '/' INDEX '/' NAME '/' TYPE ('/' Fragment )*
INDEX     : \d+
NAME      : \w+
TYPE      : \w+
```

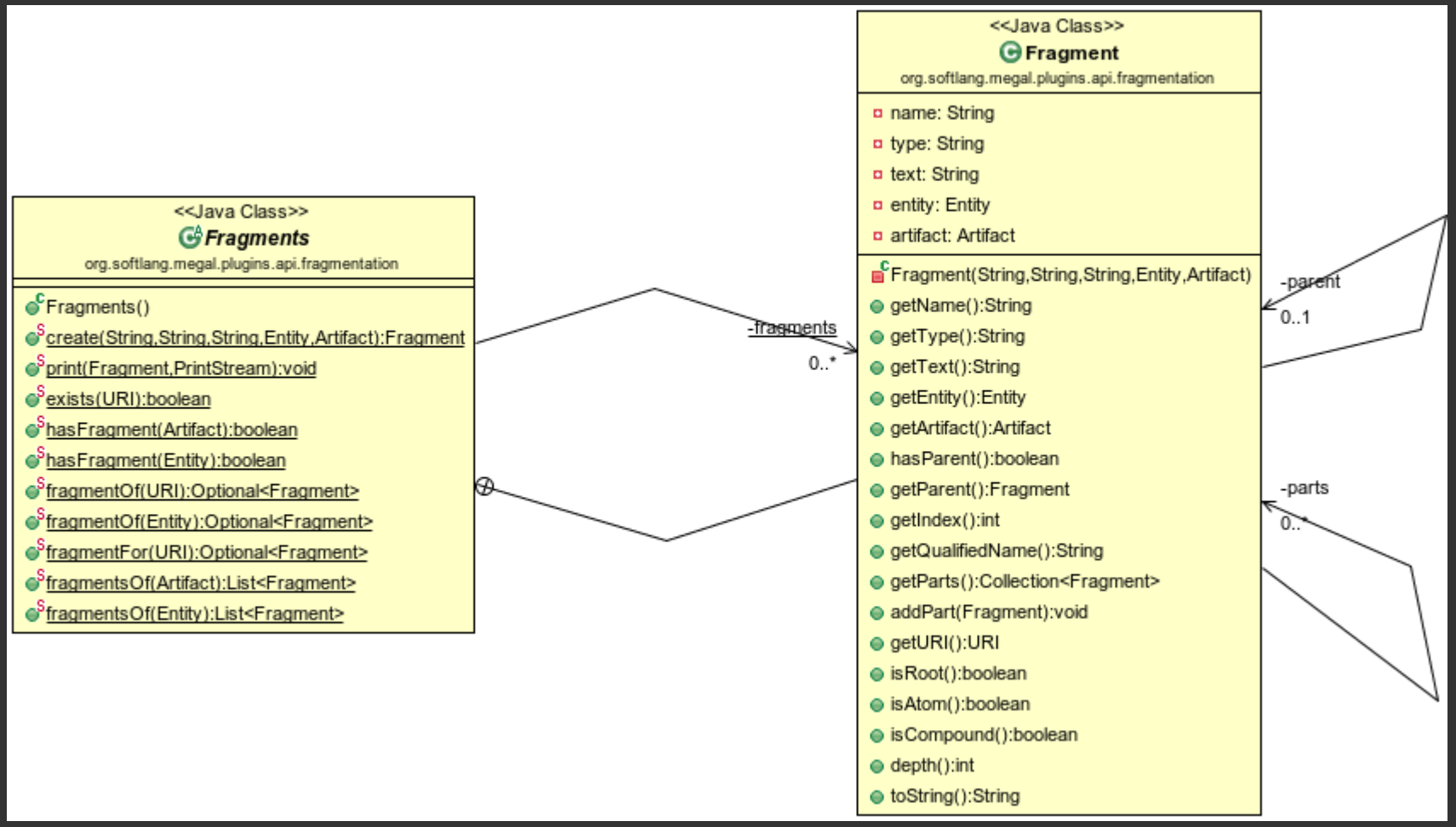
file://path/to/Foo.java#/0/Foo/JavaClass/2/getBar/JavaMethod

Fragment Model & KB

```
public class Foo {

    static public class Bar {

        private void getBar () {

        }

    }

    private String bar;

    public String getBar() {
        return bar;
    }

    public void setBar(String bar) {
        this.bar = bar;
    }

}
```

A computational fragment model should be loosely based on syntax trees.
Scope defines parthood.

Fragment Model & KB



Fragment Model & KB

- Fragments store the manifestation text and additional meta-info
- Fragments build a simple generic tree corresponding to the original AST
  - A leaf node is called ***atom***
  - A non-leaf node is called ***compound***
- A Fragment KB (*Fragments*) exists separately from the Megamodel KB during the evaluation process

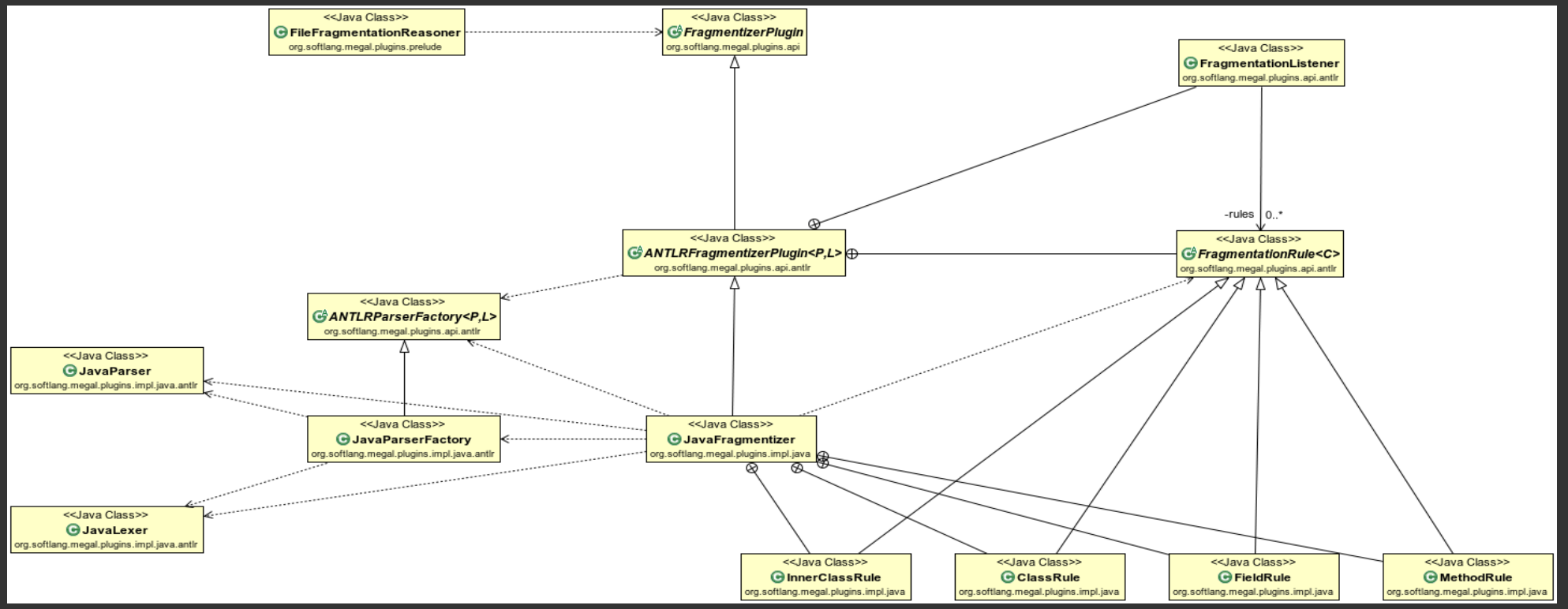
ANTLR Baked Plugins

```
File < Artifact

elementOf < File (+) * Language
realizationOf < Plugin * Entity
partOf < Plugin * Plugin

Java : Language

…

aJavaFile : File
aJavaFile elementOf Java

…

JavaAcceptor: Plugin
JavaAcceptor realizationOf Java
JavaAcceptor partOf FileElementOfLanguage
JavaAcceptor = 'classpath:org.softlang.megal.plugins.impl.java.JavaAcceptor'

…

JavaFragmentizer : Plugin
JavaFragmentizer realizationOf Java
JavaFragmentizer partOf FileFragmentReasoner
JavaFragmentizer = 'classpath:org.softlang.megal.plugins.impl.java.JavaFragmentizer'
```

ANTLR Backed Plugins

- Some entities are ***elementOf*** a language
- Some plugins are ***realizationOf*** a language
- Thus bound manifestations may need to be **parsed** for further analysis
- Also plugins may be ***partOf*** other plugins
- So actual KB derivations and parsing/anlysis can be **decoupled**

ANTLR Backed Plugins



Java Fragmentation

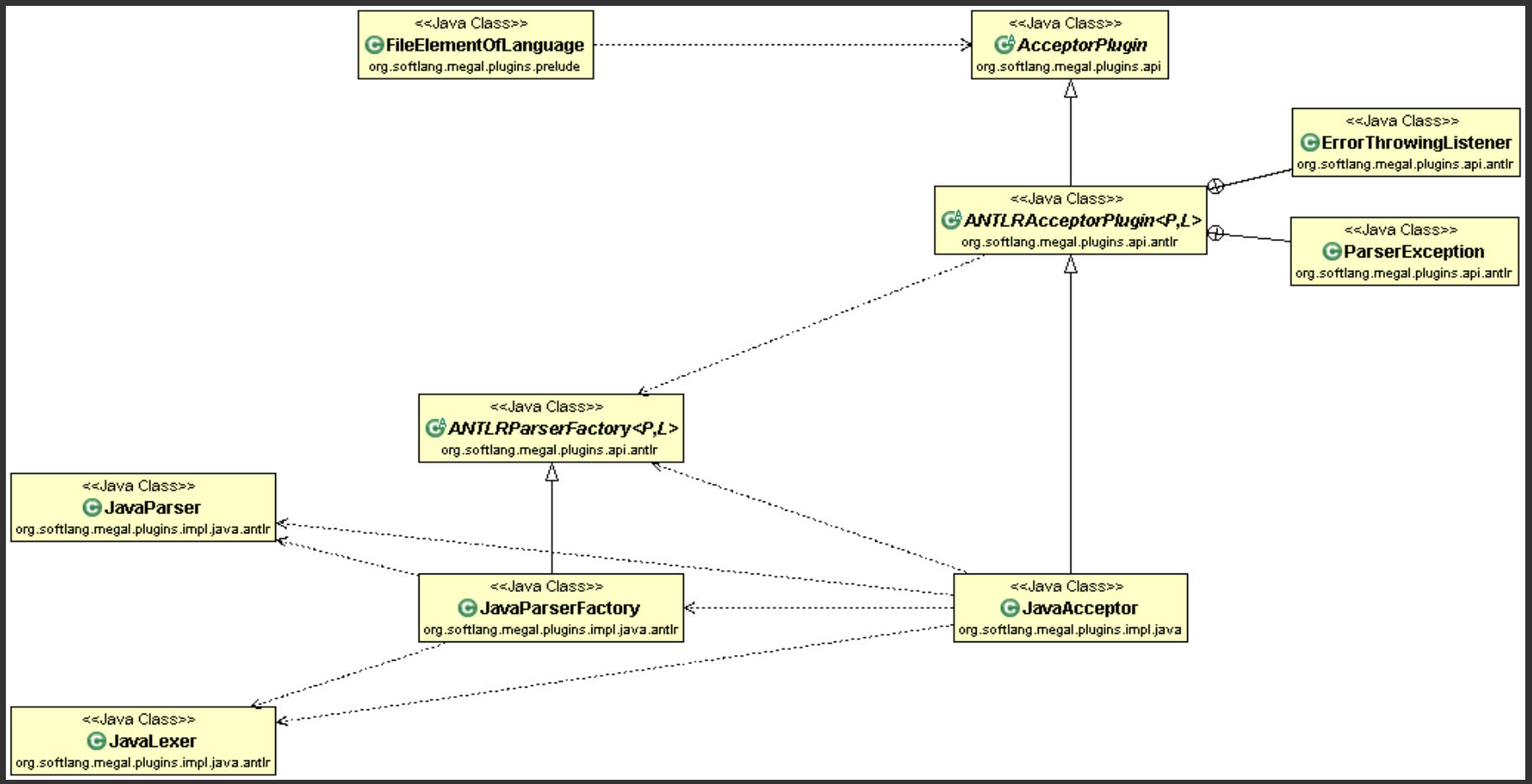
ANTLR Backed Plugins

JavaFragmentizer.java

```
1  package org.softlang.megal.plugins.impl.java;
2
3  import java.util.ArrayList;
24
26 * Disassembles a Java artifact into its fragments.
31 public class JavaFragmentizer extends ANTLRFragmentizerPlugin<JavaParser, JavaLexer> {
32
33
35     * Fragmentation rule for classes
40     static private class ClassRule extends FragmentationRule<TypeDeclarationContext> {
41
42         @Override
43         protected Class<TypeDeclarationContext> contextType() {
44             return TypeDeclarationContext.class;
45         }
46
47         @Override
48         protected boolean isAtom(TypeDeclarationContext context) {
49             return false;
50         }
51
52         @Override
53         protected boolean test(TypeDeclarationContext context) {
54             return context.classDeclaration() instanceof ClassDeclarationContext;
55         }
56
57         @Override
58         protected Fragment createFragment(Entity entity, Artifact artifact, TypeDeclarationContext context) {
59             // Create a new JavaClass fragment
60             return Fragments.create(
61                     context.classDeclaration().Identifier().getText(),
62                     "JavaClass",
63                     ANTLRUtils.originalText(context),
64                     entity,
65                     artifact
66                     );
67         }
68
69     };
70
71
73     * Fragmentation rule for inner classes
78     static private class InnerClassRule extends FragmentationRule<ClassBodyDeclarationContext> {
108
110    * Fragmentation rule for methods
```

Java Fragmentation

ANTLR Backed Plugins



Java Acceptance

ANTLR Backed Plugins

JavaAcceptor.java

```java
package org.softlang.megal.plugins.impl.java;

import org.softlang.megal.plugins.api.antlr.ANTLRAcceptorPlugin;
import org.softlang.megal.plugins.api.antlr.ANTLRParserFactory;
import org.softlang.megal.plugins.impl.java.antlr.JavaLexer;
import org.softlang.megal.plugins.impl.java.antlr.JavaParser;
import org.softlang.megal.plugins.impl.java.antlr.JavaParserFactory;


public class JavaAcceptor extends ANTLRAcceptorPlugin<JavaParser, JavaLexer> {

    @Override
    public ANTLRParserFactory<JavaParser, JavaLexer> getParserFactory() {
        return new JavaParserFactory();
    }


}

```

Java Acceptance

ANTLR Backed Plugins

- ABPs make the MegaL Plugin API **extensible** for various languages
- Acceptance plugins can be created **effortless**, just by providing a Parser-Lexer pair
- Fragmentation plugins can be created by providing by set of **“rules”** for *ANTLR ParserRuleContext* instances

XML Fragmentation

```
XMLFragment < Fragment
XMLElement < XMLFragment
XMLAttribute < XMLFragment
XMLNSAttriubte < XMLAttribute
```

```
<company
    xmlns:"http://www.101companies.org"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.101companies.org path/to/companies.xsd"
>
    <name>…</name>
    <departments>
        <department name="Research">
            …
        </department>
    </departments>
</company>
```

XSD Fragmentation

```
XML: Language
XSD: Language
XSD subsetOf XML


XMLFragment < Fragment
XMLElement < XMLFragment
XMLAttribute < XMLFragment
XMLNSAttribute < XMLAttribute


XSDFragment < XMLFragment
XSDSchema < XMLAttribute       // or XSDFragment ???
XSDElement < XMLElement        // or XSDFragment ???
XSDElementName < XMLAttribute // or XSDFragment ???
XSDComplexType < XMLElement    // or XSDFragment ???
```

Because XSD is a subset of XML,
some uncertainty for XSD fragment type specializations may exist!

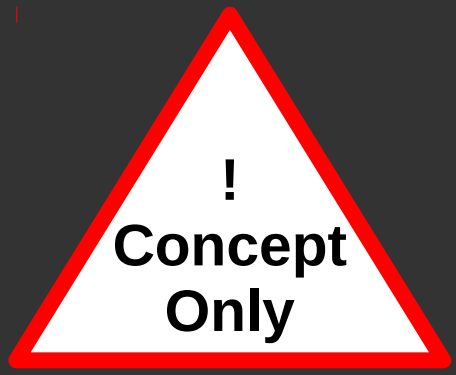
XSD Fragmentation

```
XML: Language
XSD: Language
XSD subsetOf XML


XMLFragmentizer : Plugin
XMLFragmentizer realizationOf XML
XMLFragmentizer partOf FileFragmentReasoner
XMLFragmentizer = 'classpath:org.softlang.megal.plugins.impl.xml.XMLFragmentizer'


XSDFragmentizer : Plugin
XSDFragmentizer realizationOf XSD
XSDFragmentizer partOf FileFragmentReasoner
XSDFragmentizer = 'classpath:org.softlang.megal.plugins.impl.xsd.XSDFragmentizer'
```

**Because XSD is a subset of XML,
XML fragmentation is currently also applied to
entities which are element of XSD!**

XML Namespace Specialization

```
XMLFragment < Fragment
XMLElement < XMLFragment
XMLAttribute < XMLFragment
XMLNSAttriubte < XMLAttribute
```



- Use XML Namespace Attributes as anchors for further fragment type specialization
- Analysis of XSD artifacts should infer new entity types
- **This would also cover XSD Fragmentation**
- (DTD support may also be necessary)

TODO

- Implement further Java fragments:
  - Interface, Enum, Annotation, …
- Implement SQL/DDL Fragmentation
- (?) Implement XML Namespace Specialization
- (?) Merge with main repository
  - Integration of Fragment Model & KB
    - Add manifestation support to Entity class
    - Change binding type from plain Object to java.net.URI