

MegaL Traceability Recovery

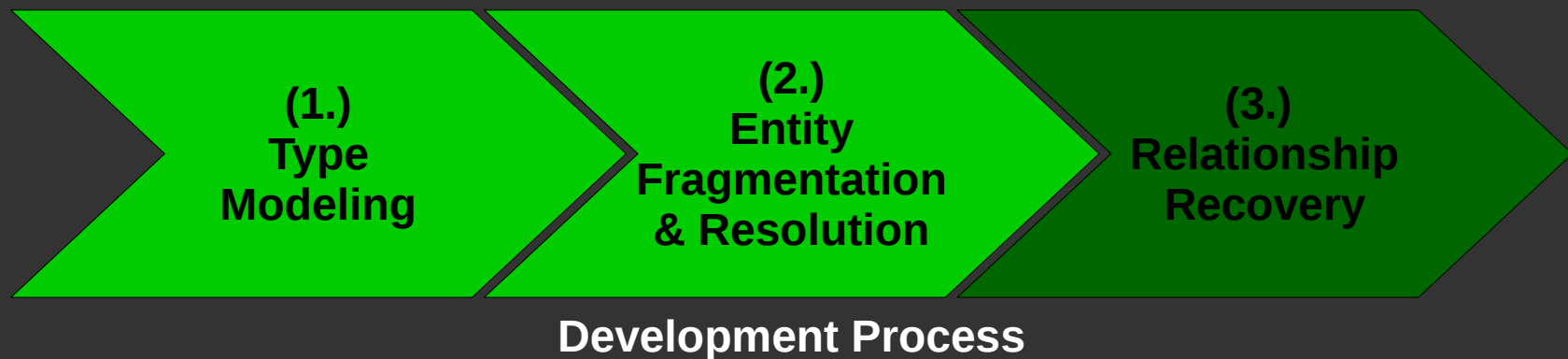
<https://github.com/maxmeffert/megal-tr>

A Fragment Model & *ANTLR* Backed Plugins

Meeting 2016-06-28

*University of Koblenz-Landau
Maximilian Meffert*

Quick Recap



(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]

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+
```

Qualified Fragment Names

- **QFNs** are used as identifiers for the derived entities
- **QFNs** depict *partOf* relationships
- **Indexes** depict the position of fragments in their respective composite

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+										

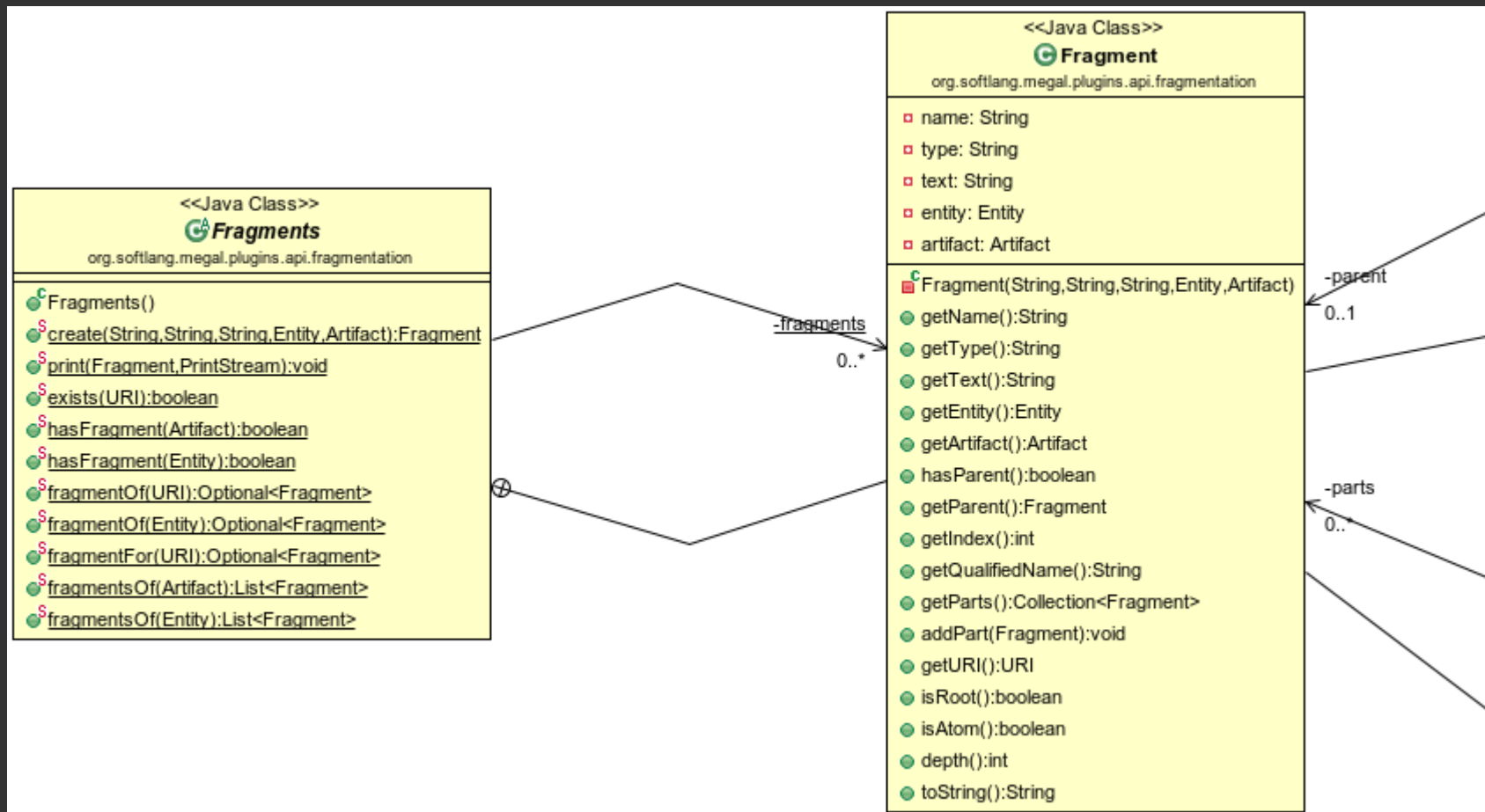
file://**path/to/Foo.java**#**/0/Foo/JavaClass/2/getBar/JavaMethod**

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 alongside 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 an 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'

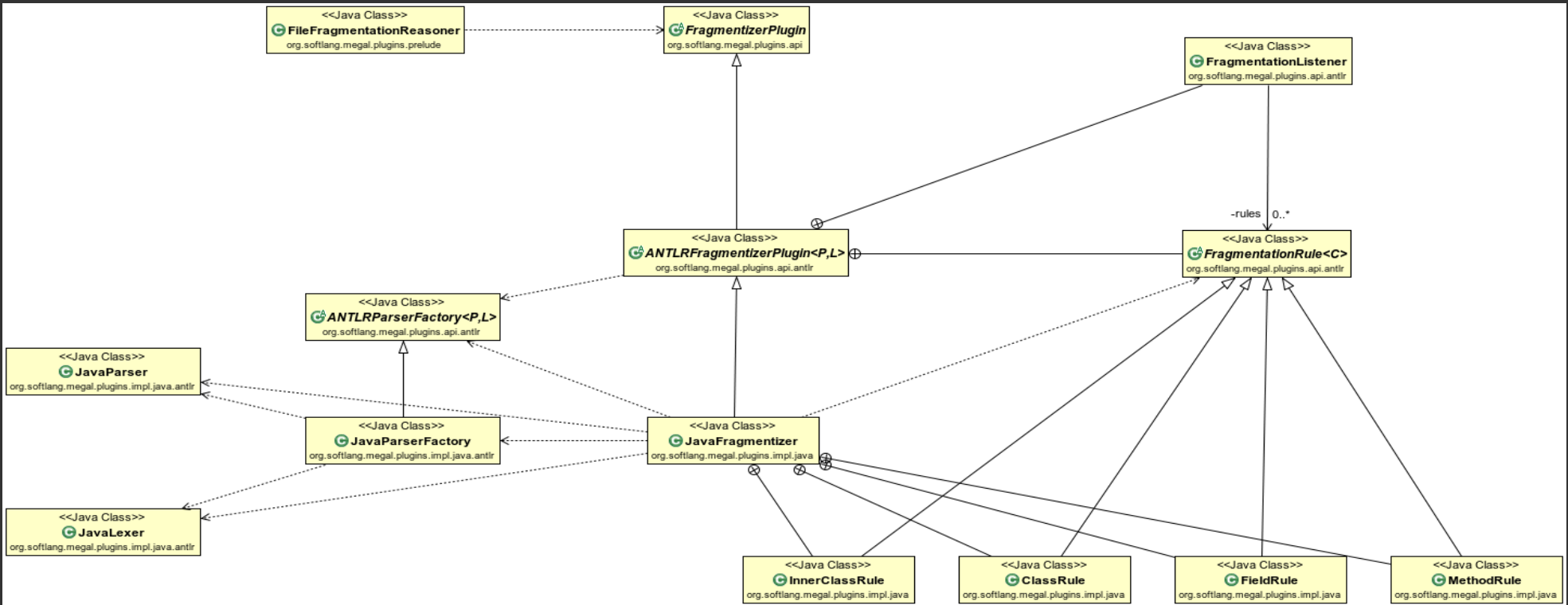
...

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

ANTLR Backed Plugins

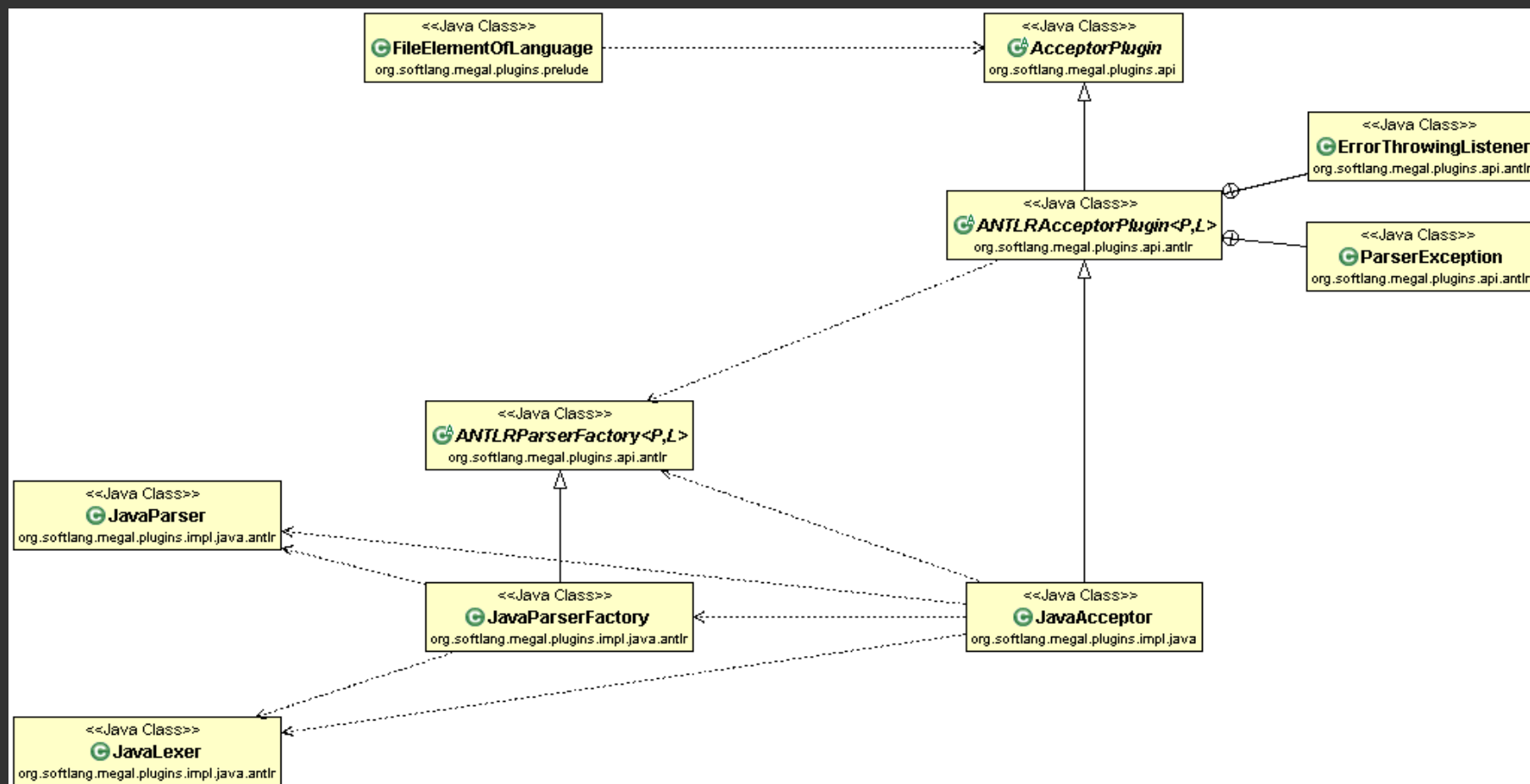
- Some entities are *elementOf* a language
- Some plugins are *realizationOf* a languages
- Thus bound manifestations may need to be parsed for further analysis
- Also plugins may be *partOf* other plugins
- So actual KB derivations and parsing/anlysis can be decoupled

ANTLR Backed Plugins



Java Fragmentation

ANTLR Backed Plugins



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 can be created

XML Fragmentation

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

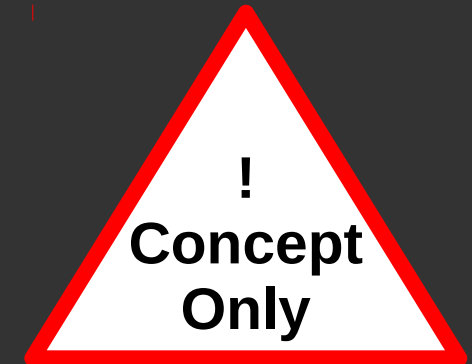
```
XML: Language  
XSD: Language  
XSD subsetOf XML
```

```
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  
XMLNSAttribute < 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/DDDL Fragmentation
- (?) Implement XML Namespace Specialization
- (?) Merge with main repository