

# Transforming Java by separated analysis/synthesis phases

An introduction using ANTLR and  
StringTemplate

[oliver@zeigermann.de](mailto:oliver@zeigermann.de)

# What do we want

- ▶ Java to Java transformation
- ▶ Generation of an EJB Session Bean delegate for a POJO implementation of a simple Java Service Interface

# Example: Java Source Interface

```
package test.service;

import test.bo.Person;
import test.core.ServiceException;

public interface PersonService {

    boolean isMale(Person id) throws ServiceException;

    Person populate(Person id) throws ServiceException;
}
```

# Example: Session Bean Target

```
package test.service;
```

```
import test.bo.Person;
```

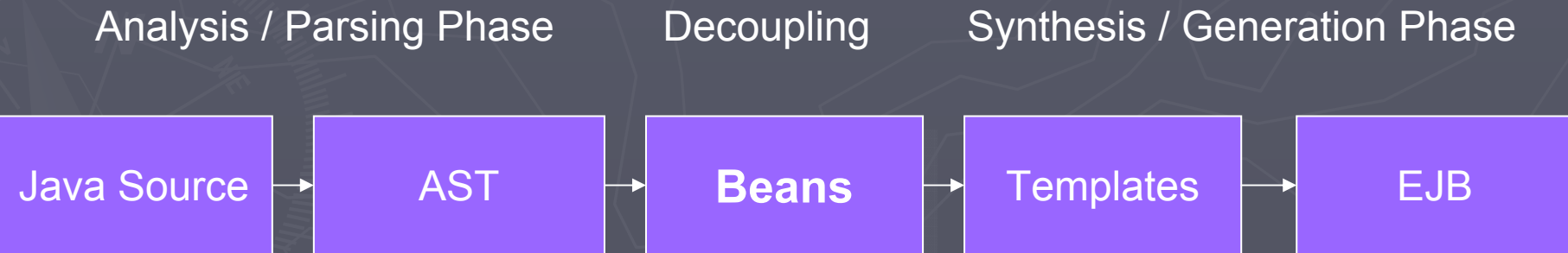
```
import test.core.ServiceException;
```

```
import javax.ejb.*;
```

```
public class PersonServiceSessionBean implements SessionBean, PersonService {  
    private PersonService service;  
    public void setService(PersonService service) {  
        this.service = service;  
    }  
    public boolean isMale(Person id) throws ServiceException {  
        return service.isMale(id);  
    }  
    public Person populate(Person id) throws ServiceException {  
        return service.populate(id);  
    }  
}
```

# The basic idea

- ▶ Decoupling the parsing from the generation phase using an intermediate Java-Bean representation



# Tools

- ▶ ANTLR: <http://www.antlr.org/>
  - Java Grammar:  
<http://www.antlr.org/grammar/1093454600181/java15-grammar.zip>
- ▶ StringTemplate:  
<http://www.stringtemplate.org/>

# Benefits of the basic idea

- ▶ Backend developer does not need rarely found knowledge about compiler design and ANTLR
- ▶ StringTemplate has natural support for beans, properties and collections

# Interface-Bean

Java 5 syntax  
used for clarity,  
but real code uses  
1.2 syntax

```
public class Interface {  
    String packageName;  
    List<String> imports = new ArrayList<String>();  
    List<String> modifiers = new ArrayList<String>();  
    String name;  
    List<Methods> methods = new ArrayList<Methods>();  
}
```

# Method-Bean

```
public class Method {  
    List<String> modifiers = new ArrayList<String>();  
    Type type;  
    String name;  
    List<Parameter> parameters = new ArrayList<Parameter>();  
    List<String> exceptions = new ArrayList<String>();  
}
```

# Parameter-Bean

```
public class Parameter {  
    boolean isFinal;  
    String name;  
    Type type;  
}
```

# Type-Bean

```
public class Type {  
    String name;  
    int arrayCnt ← 0;  
}
```

Dimension of array,  
0 stands for no array

# Tree Parser Action Code Sample

```
methodDecl
{
  Method method = new Method();
  Type type;
  List<String> m;
}
: #(METHOD_DEF m=modifiers { method.setModifiers(m); }
  type=typeSpec { method.setType(type); }
  methodHead[method])
{ getInterface().getMethods().add(method); }
;
```

Bean data to be filled  
by analysis

Parses tree fragment  
coming from  
Java parser

Filling of bean  
data

# Java Glue Code: Analysis

```
Reader reader = new BufferedReader(new FileReader(file));  
JavaLexer lexer = new JavaLexer(reader);  
lexer.setFilename(fileName);
```

```
JavaRecognizer parser = new JavaRecognizer(lexer);  
parser.setFilename(fileName);
```

```
parser.compilationUnit();
```

```
JavaTreeParser tparse = new JavaTreeParser();  
tparse.compilationUnit(parser.getAST());  
Interface interfaz = tparse.getInterface();
```

Start rule (entry point)  
for both Java and  
Java tree grammar

Retrieval of filled  
bean data

# Java Glue Code: Synthesis

Helper method  
that returns a Reader

```
StringTemplateGroup group =  
    new StringTemplateGroup(getGroup("objectGroup"));  
StringTemplate template = group.getInstanceOf("object");
```

```
template.setAttribute("package", interfaz.getPackageName());  
template.setAttribute("imports", interfaz.getImports());  
template.setAttribute("service", interfaz.getName());  
template.setAttribute("methods", interfaz.getMethods());
```

```
String target = template.toString();
```

This attribute remains  
a complex typed list.  
Conversion to string  
takes place at the very  
end.

# Session Bean Template #1

```
group objectGroup;
```

Template uses group format

```
object(package,imports,service,methods) ::= <<
```

Matches attributes set by glue code

```
package $package$;
```

```
$imports:{n| import $n$;}; separator="\n"$
```

Simple string substitution

```
import javax.ejb.EJBObject;  
import java.rmi.RemoteException;
```

Anonymous subtemplate application to String typed List

```
public interface $service$Object extends EJBObject {  
    $methods:method(); separator="\n"$  
}
```

Named subtemplate application to complex Typed List

```
>>
```

# Session Bean Template #2

```
method(method) ::= <<
```

Access to bean fields

```
public $method.type$  
$method.name$($method.parameters:parameterDecl(); separator=", "$)  
    throws $method.exceptions; separator=", "$, RemoteException;  
  
>>
```

```
parameterDecl(parameter) ::= <<
```

```
$if($parameter.final)$final $endif$ $parameter.type$ $parameter.name$  
  
>>
```

Only control structure  
available in ST,  
very useful (but not mandatory)  
for conditional text

# Summary

- ▶ You get a complete Java Source parser for free
- ▶ Traversing an AST requires ANTLR and parsing knowledge
- ▶ Beans as an intermediate format is an ideal input for templates and decouples parsing from generation
- ▶ StringTemplate is a great template engine
- ▶ Full source code including libraries available at <http://www.zeigermann.de/genEJB.zip>