Component and Services Generation

Didier Donsez

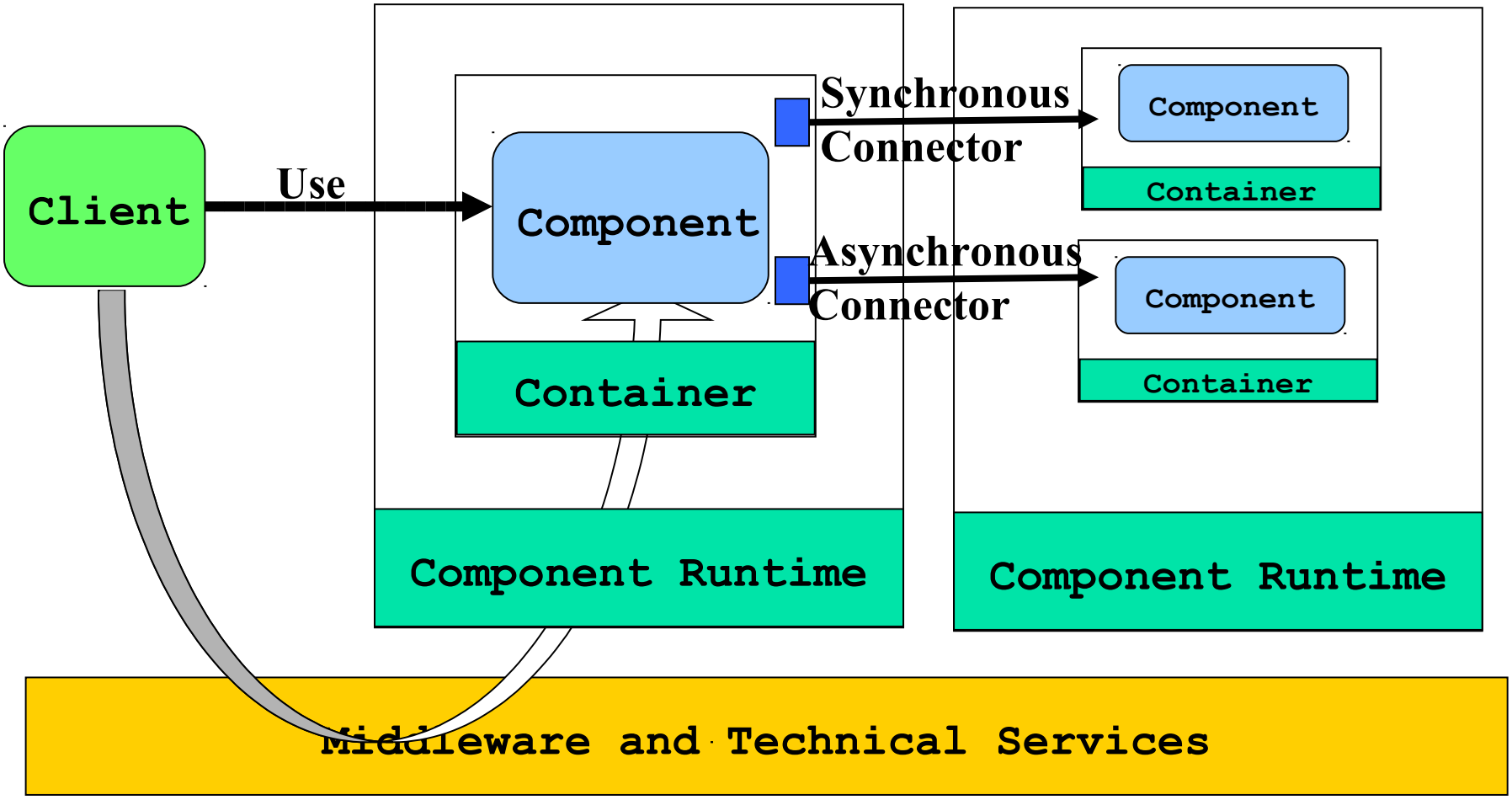
Université Grenoble Alpes

PolyTech' Grenoble - LIG / ERODS

Prenom.Nom@imag.fr

Firstname.Lastname@ieee.org

Component and Service Containers



From [Rveill & Merle]

Generative Programming

- Some definitions (taken from Generative Programming Wiki)
  - The goal of generative programming is to **replace manual** search, adaptation, and assembly of components with the **automatic generation** of needed components on demand [from the call for papers of GP2002 at ICSR7].
  - The goal of generative and component-based software engineering is to **increase the productivity, quality, and time-to-market** in software development thanks to the deployment of both standard componentry and production automation. One important paradigm shift implied here is to build software systems from standard componentry rather than "**reinventing the wheel**" each time. …. . Generative and component-based software engineering seeks to integrate domain engineering approaches, component-based approaches, and generative approaches. [from GCSE working group page]
  - Generative programming is a software engineering paradigm based on **modeling software families** such that, given a particular requirements specification, a highly customized and optimized intermediate or end-product can be automatically manufactured on demand from elementary, **reusable implementation** components by means of configuration knowledge. [from the GenerativeProgrammingBook]

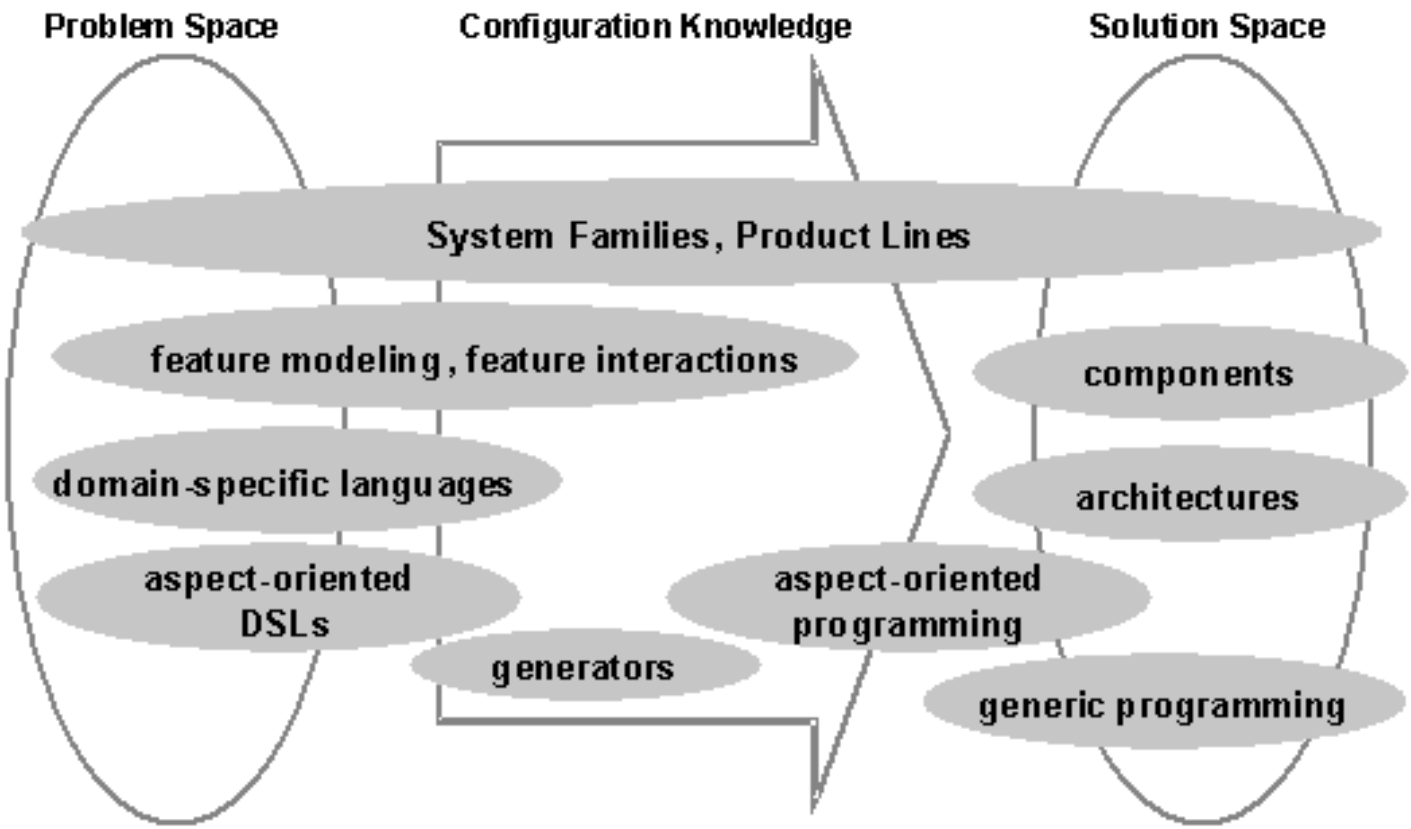
Lectures

- Krzysztof Czarnecki, Ulrich W. Eisenecker: Generative Programming - Methods, Tools, and Applications. Pub. Addison Wesley, 2000, ISBN 0201309777, http://www.generative-programming.org/
- Don Batory: The Road to Utopia: A Future for Generative Programming. International Seminar on Domain-Specific Program Generation, Dagstuhl Castle, Germany, March 23-28, 2003, LNCS 3016, pp 1-17
- Code Generation Network http://www.codegeneration.net/
- Generative Programming Wiki
  - http://www.program-transformation.org/Transform/GenerativeProgrammingWiki

Program transformation tools

- Lex / Yacc (and their derivatives for each language) Macros and preprocessors
- CPP for C camlp4
- Stratego/XT
- Xpath/XLST
- Xtext/XTend
- DMS
- JastAdd (extensible Java compiler)
  - http://jastadd.org/web/index.php
- Cocinelle

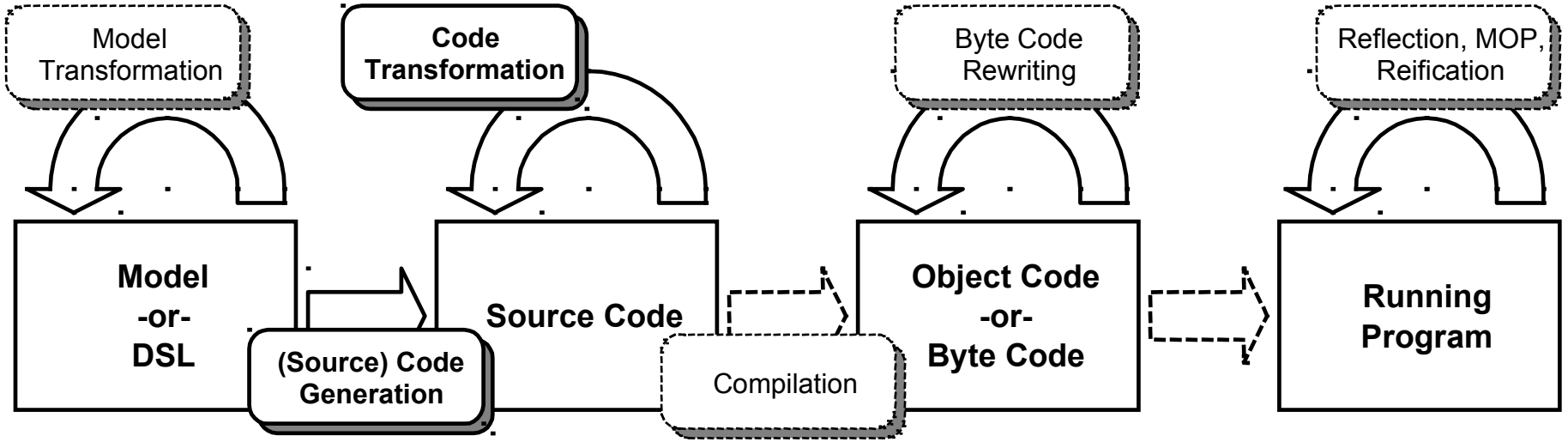
Generative programming



From [Czarnecki & Eisenecker]

- DSL : Domain Specific Language
- SPL : Software Product Line

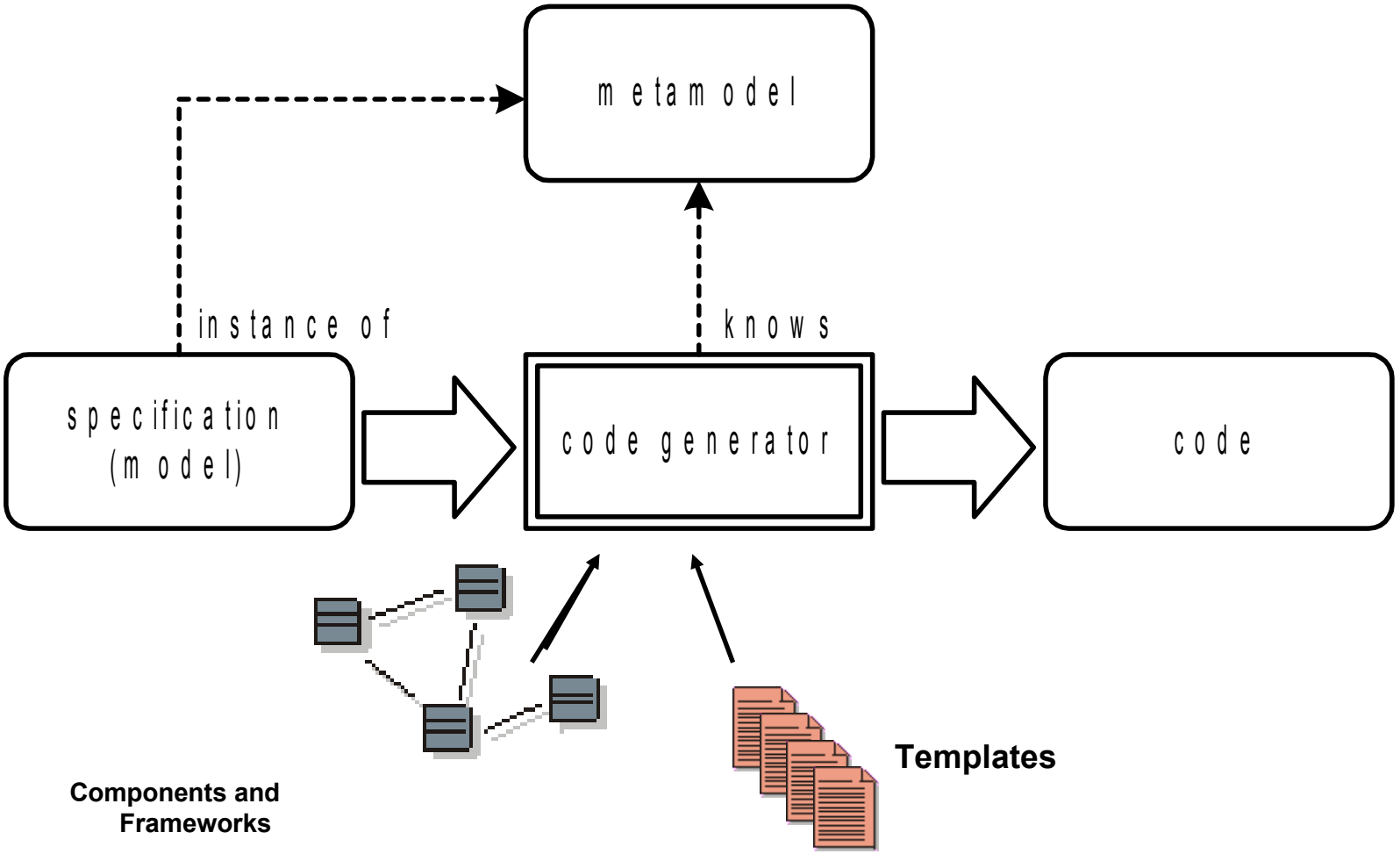
Code generation



- M. Voelter, A Catalog of Patterns for Program Generation, EuroPlop'2003

Sources: www.voelter.de

Code generation



- M. Voelter, A Catalog of Patterns for Program Generation, EuroPlop'2003

Sources: www.voelter.de

Metadata

- Describe and configure a component, a service, a architecure
  - Interfaces, Properties, …
  - Components and services binding
- In-source code / Separated
  - XML/JSON/YAML Format
    -  externe, independance with the programming language, structural validation (*XML schema*) , modular (*namespace*), regular expressions (***/Test**)
    -  Consistence between description ←→ implementation (requière des plugins)
  - .NET Attributes, XDocLet, Java Annotations, CPP Macro, …
    - validation by the compiler of the programming language
    -  Structural validation (Spoon/Aval), scattered in the source code, annotation class by class (no regex)
    - 
  - Major trends
-  Both are required (source code is not alway available or obfuscated)
  - POJO, close-source, COTS, *legacy software*, …

Generation Time

- Build
- Assembling
- Deployment
- Loading
- Execution

Tradeoff in generation

- Optimisation & Performance
  - Operation invocation & memory footprint & GC
- Flexibility
  - Hot Reconfiguration (dynamic)
  - vs static (ie stop, rebuild the container, restart)

Generation Techniques

- Reflexion
  - Proxy dynamiques
- Source generation
  - Velocity/NVelocity, Jelly, XSLT, Moustache
- Bytecode/IL generation
  - ASM, BCEL, Jabyce, CGLib (http://cglib.sourceforge.net/) .NET System.Reflect
- Mixin
  - Julia, Scala
- AST Transformation
  - APT, Spoon, CodeDom
- Aspect weavers
  - AspectJ, AspectWerz, Guice

Réflexion

- Langages reflexifs (Smalltalk, Java, C#, C++, JavaScript …)
  - Représentation du programme
    - Méthodes, champs, annotations, instructions …
  - Capacité du programme à s'introspecter (w/o le modifier)

- Compile time
  - Sert à l'exploration du code en vue de générer des sources
    - comme rmic, java2wsdl …
- Execution time

  - Proxy dynamiques
    - - coût du boxing/unboxing lors des invocations

- coût de la réflexion
  - Lors qu'elle est utilisée au runtime
  - Impossible de l'utiliser dans certains environnements (JavaCard, J2ME, …) car la représentation du programme est coûteuse en mémoire

  - Exemple : JMX MBeans (Java)

Génération de sources (texte)

- Langages de patron (*template*)
  - XSLT, Velocity/NVelocity, Eclipse CodeGen JET, Jelly …

```
package ${pkgName};
public interface ${itfName}MBean {
#foreach ($attribute in ${attributesList})
public void set${attribute}(String new${attribute});
public String get${attribute}();
#end
}
```

```
pkgName=foo
itfName=Config
attributesList=Port,Host,Credential
```

```
package foo;
        public interface ConfigMBean {
        public void setPort(int newPort);
        public int getPort();
        public void setHost(String newHost);
        public String getHost();
        public void setCredential(byte[] newCredential);
        public byte[] getCredential ();
}
```

- ease to learn
- hard to debug and maintain
  - Non modular
  - templates can not be validated by an off-the-shelf compiler/IDE
  - templates can be validated only when all generations cases are tested !!

Velocity (Apache)

- Langage de templates (VTL)
  - Syntaxe proche des macros CPP
    - Macros #set, #foreach() ... #end, #if () …#elseif () …#else …#end, #include(…), #parse(…)
    - Variables $var ou ${var}
- Usage
  - Initialement défini par la génération de pages Web
  - Utilisé aussi pour la génération de conteneurs, *build* (Makefile, Ant, Maven) des projets, …
- Exemple VTL

```
// generated at $date
package ${pkgName};
public interface ${itfName}MBean {
#foreach ($attribute in ${attributesList})
    /** setter for the attribute ${attribute} */
    public void set${attribute}(String new${attribute});
    /** getter for the attribute ${attribute} */
    public String get${attribute}();
#end
    /** reset all the attributes */
    public void reset();
}
```

```
public static void main(String [] args) {
  Velocity.init();
  VelocityContext vc = new VelocityContext();
  vc.put("date", new Date());
  vc.put("itfName", "Config"); …
  Template template
     = Velocity.getTemplate(args[0]);
  OutputStreamWriter osw =
                  new StringWriter(System.out);
  template.merge(vc, osw);
}
```

Ant task, maven plugin, …

Jelly

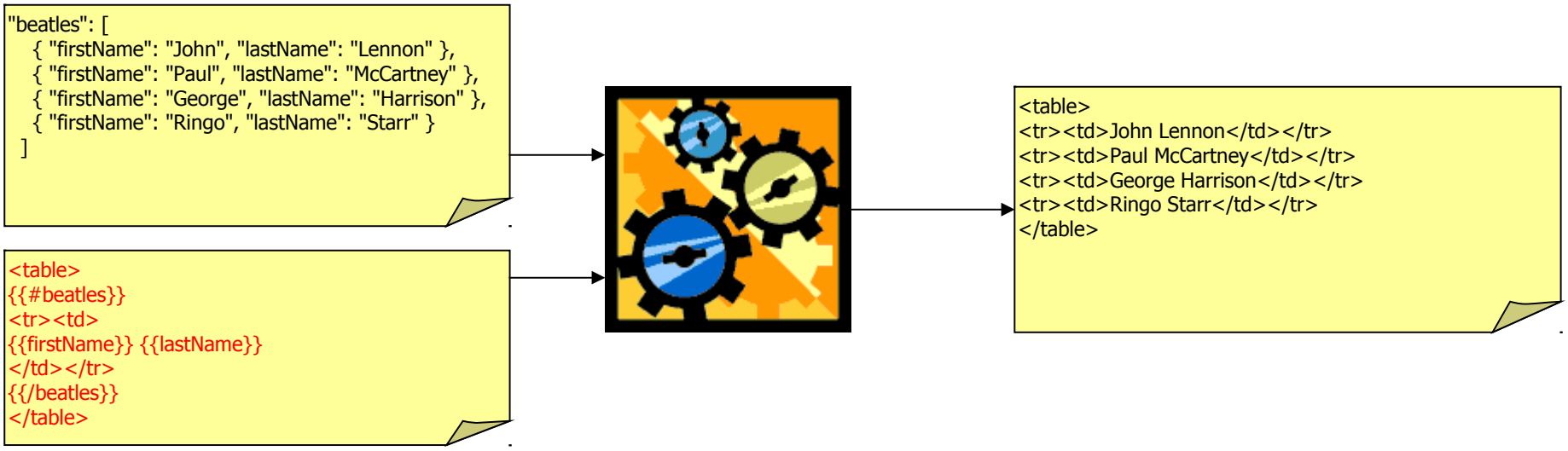
- Langage de templates basé sur XML
  - Similaire aux JSPs
  - modulaire (namespace)
  - extensible (taglib)
  - langage d'expression (jxel)
- Exemple Jelly

```
<j:jelly ...>
// generated at ${date}
package ${pkgName};
public interface ${itfName}MBean {
<j:forEach items="${attributesList}" var="attribue">
   /** setter for the attribute ${attribute} */
   public void set${attribute}(String new${attribute});
   /** getter for the attribute ${attribute} */
   public String get${attribute}();
</j:forEach>
   /** reset all the attributes */
   public void reset();
}
</j:jelly>
```

```
OutputStream output = new FileOutputStream
                ("ConfigMBean.java");
JellyContext context = new JellyContext();
context.setVariable("pkgName","foo");
context.setVariable("itfName","Config");
Vector v = new Vector();
v.add("Port");
v.add("Host");
context.setVariable("attributesList", v);
XMLOutput xmlOutput =
                XMLOutput.createXMLOutput(output);
context.runScript("src/container/"
                +template), xmlOutput);
xmlOutput.flush();
```

Moustache (https://mustache.github.io/)

- Polular in the Javascript world (but available in other languages)
-



XSLT

- Transformation de XML vers XML ou texte

Adapté quand génération de métadonnées XML

Très verbeux

- Maintenance non aisée, …

XSLT Code Generation Example

```
<?xml version="1.0" … ?>
<component name="Config">
 <attributes>
  <attribute name="host" type="String">
  <attribute name="port" type="int">
  <attribute name="guid" type="String">
 </attributes>
 <operations> ... </operations>
</component>
```

```
<xsl:stylesheet ...>
<xsl:output method = "text"/>
<xsl:param name="package"/>

<xsl:template match="/">
package <xsl:value-of select="$package"/>;
<xsl:apply-templates select="*"/>
</xsl:template>

<xsl:template match="component">
public interface <xsl:value-of select="@name"/>MBean {
        <xsl:apply-templates select="attributes/attribute" mode="getterdeclaration"/>
        <xsl:apply-templates select="attributes/attribute" mode="setterdeclaration"/>
        MBeanInfo getMBeanInfo();
}
</xsl:template>

<xsl:template match="attributes/attribute" mode="getterdeclaration">
        <xsl:variable name="capName" select="java:GenerationUtility.capitalize(@name)"/>
        public <xsl:value-of select="@type"/> get<xsl:value-of select="$capName"/>();
</xsl:template>

<xsl:template match="attributes/attribute" mode="setterdeclaration">
        <xsl:variable name="capName" select="java:GenerationUtility.capitalize(@name)"/>
                public void set<xsl:value-of select="$capName"/>(
                        <xsl:value-of select="@type"/> new<xsl:value-of select="$capName"/>Value);
</xsl:template>
<xsl:template match="*"></xsl:template>
</xsl:stylesheet>
```

DVSL (Declarative Velocity Style Language)

http://velocity.apache.org/dvsl/devel/

- Velocity stylesheets similar to XLST templates
  - Document control and selection is based on XPath.
  - and with conventional Velocity syntax

Advantages versus XSLT:

- Less verbose

Limitations versus XSLT:

- namespaces, conditionnal template, result validation, …

- Example

```
#match("component")
// generated at ${context.date}
package ${context.pkgName};
public interface ${attrib.name}MBean {
  $context.applyTemplates()
  /** reset all the attributes */
  public void reset();
}
#end
#match("attributes/attribute")
  /** setter for the attribute ${attrib.name} */
  public void set${attrib.name}(String new${attrib.name});
  /** getter for the attribute ${attrib.name} */
  public ${attrib.type} get${attrib.name}();
#end
```

```
<?xml version="1.0" … ?>
<component name="Config">
  <attributes>
    <attribute name="host" type="String">
    <attribute name="port" type="int">
    <attribute name="guid" type="String">
  </attributes>
  <operations> ... </operations>
</component>
```

Eclipse JET (Java Emitter Templates)

- Generic Template Engine
  - JSP-like syntax (EL, …)
  - EMF eCore model

```
<%@ jet package="translated" imports="java.util.*,my.comp.*" class="ComponentClass" %>
<% Hashtable model = (Hashtable) argument;%>
<% String pkgName = (String) model.get("pkgName");%>
<% Component component = (Component) model.get("Component");%>
package <%=pkgName%>;
public class <%=component.getName()%>MBean {

 <%  Attribute attributes[] = component.getAttributes();
    for (int i=0;i < attributes.length; ++i){
       Attribute attribute = attributes[i];
 %>
 <%=attribute.getType()%> get<%=attribute.getName().toLowerCase()%>();
 void set<%=attribute.getName().toLowerCase()%>(<%=attribute.getType()%> newValue);
 <%
    }
 %>
}
```

Bytecode/IL Generation

- Bytecode or LI manipulations

- Pros and Cons
  -  fits well « on-the-fly » class manipulation
    - Load time generation
  -  light weight frameworks
  -  error-prone, hard maintenance, hard to optimize, …
    - Bytecode Modification Problem
      - Lot of serialization/deserialization detail, Remove/Add in constant pool, Jump offset, Stack Size, …
- Frameworks
  - ASM, BCEL, SERP, JOIE, JMangler, Jabyce
  - **java.lang.instrument package since J2SE1.5**
  - .NET System.Reflect, …

ASM *http://asm.objectweb.org*

- Java Bytecode Manipulation
  - Visitor design pattern
    - ClassReader → *Visitor → *Writer
  - Support Java 5 annotations, generics, …
  - Common transformations
    - Class Transformations
      - Introduce Interface, Add a New Field, Add a New Method, Replace Method Body, Merge Two Classes into One (~*Mixin*)
    - Method Transformations
      - Insert Code before Method, Constructor or Static Initializer Execution, Insert Code before Method Exit, Replace Field Access, Replace Method Call, Inline Method
- Example

```
public class FieldAdder extends ClassAdapter {
  private final FieldNode fn;
  public FieldAdder(ClassVisitor cv, FieldNode fn) {
    super(cv);
    this.fn = fn;
  }
  public void visitEnd() {
    fn.accept(cv);
    super.visitEnd();
} }
```

http://asm.objectweb.org/current/asm-transformations.pdf

Aspect Weaver

- AOP (Aspect Oriented Programming)
  - Séparation des préoccupations
  - Langages d'aspects
    - Aspect, Join point, point cut, advice, …

Voir

- http://en.wikipedia.org/wiki/Aspect-oriented_programming
- http://fr.wikipedia.org/wiki/Programmation_orient%C3%A9e_aspect

  - Canevas
    - AspectJ, AspectWerz, JAC, Spring AOP, AspectDNG, LoomNET …
    - AOP Alliance
- Compenent container with AOP
  - Example : AOKell (AspectJ)
    - http://www2.lifl.fr/~seinturi/papers/fractal-ecoop06-fractnet.pdf

AOKell Container with AspectJ

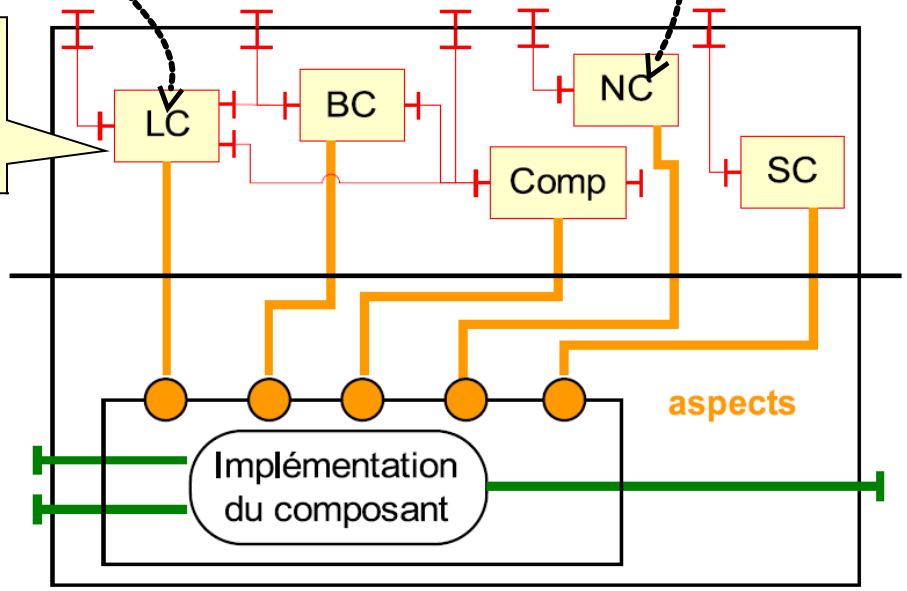
```
public aspect ALifeCycleController {

    private LifeCycleController FlatType._lc;

    public String FlatType.getFcState() { return _lc.getFcState(); }
    public void FlatType.startFc() throws IllegalLifeCycleException { _lc.startFc(); }
    public void FlatType.stopFc() throws IllegalLifeCycleException { _lc.stopFc(); }

    pointcut methodsUnderLifecycleControl( FlatType adviced ):
        execution( * FlatType+.*(..) ) && target(adviced) &&
        ! controllerMethodsExecution() && ! jlObjectMethodsExecution();

    before(FlatType adviced) : methodsUnderLifecycleControl(adviced) {

      if( adviced.getFcState().equals(LifeCycleController.STOPPED) ) {
        throw new RuntimeException("Components must be started before
                                    accepting method calls");
```

```
public aspect ANameController {

    private NameController FlatType._nc;

    public String FlatType.getFcName() {
        return _nc.getFcName();
    }

    public void FlatType.setFcName(String arg0) {
        _nc.setFcName(arg0);
    }

    public NameController FlatType.getFcNameController() { return _nc; }
    public void FlatType.setFcNameController(NameController nc) { _nc=nc; }
}
```

AspectJ Inter-type declarations

Object implementation of the name controller

Controllers are also components



Contrôleurs
BC : Binding
LC : Lifecycle
NC : Name
SC : Super
Comp : Component

Niveau applicatif

d'après http://www.emn.fr/x-info/jte-aets/seinturier.pdf

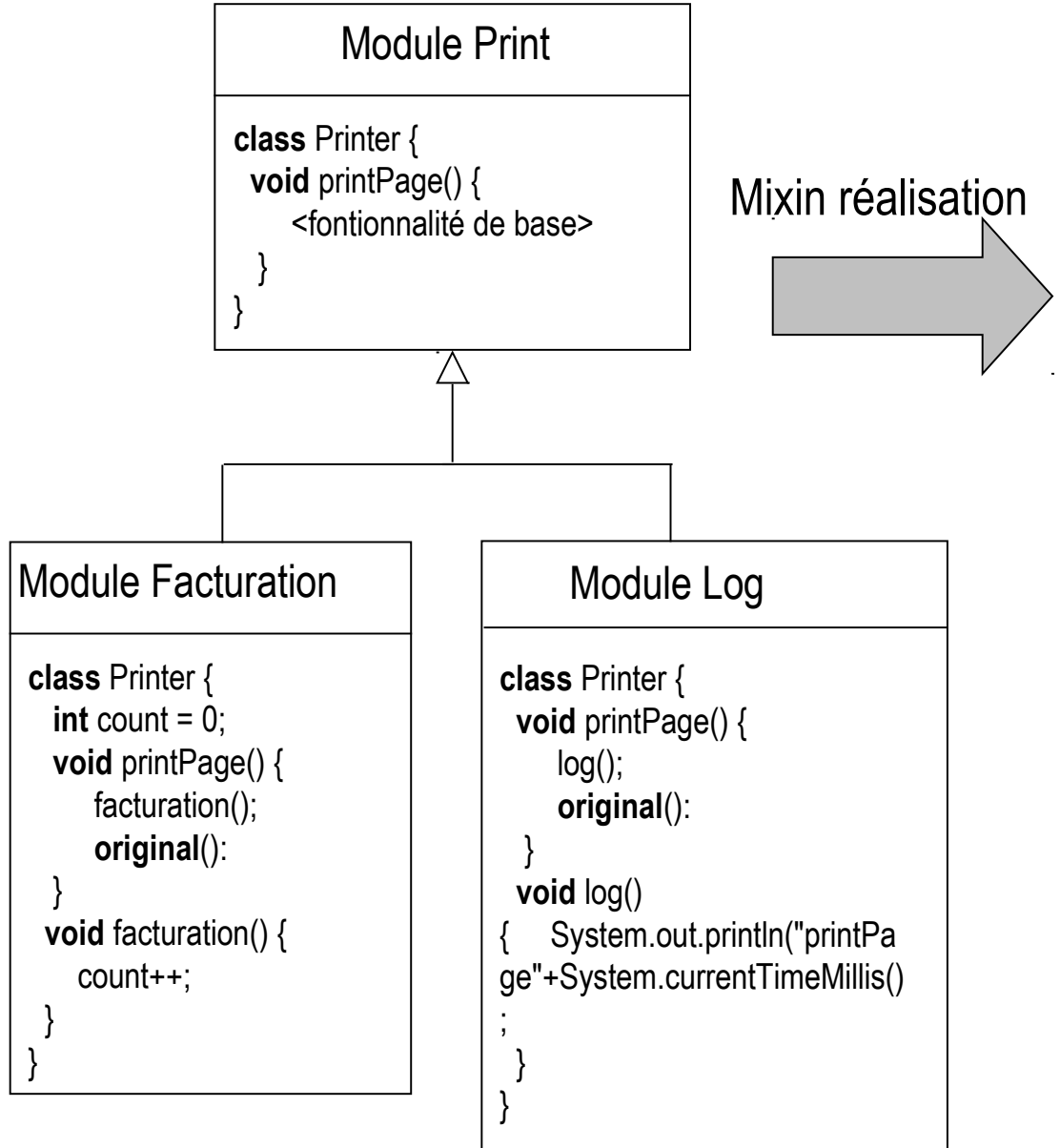
Mixin

- Pour adapter une classe « source »
- Dépendance avec cette classe source.
  - Connaissance des méthodes et des attributs
- Possible en java avec un tisseur
  - Les méthodes et les attributs référencés par la classe mixin sont réécrit dans la classe mixin avec le prefix _super_.
  - Convention du tisseur.
- Pas une vision global du tissage.
- Mixeurs
  - MixJuice, Julia Mixin (basé sur ASM), *Google Guice* … JavaFX 1.2
- Remarque
  - Aussi une manière de combler l'absence d'héritage multiple (Java)

References.

- Gilad Bracha. The Programming Language Jigsaw: Mixins, Modularity and Multiple Inheritance . PhD thesis, University of Utah, 1992.
- Gilad Bracha and William Cook. Mixin-based inheritance. In Proc. of the Joint ACM Conf. on Object-Oriented Programming, Systems, Languages and Applications and the European Conference on Object-Oriented Programming, October 1990.

Mixin -Langage



Mixim - Java

```
class Printer {
  void printPage() {
      <fontionnalité de base>
   }
}
```

```
abstract class Printer_Facturation {
   int count = 0;
  abstract __super__printPage();

  void printPage() {
       facturation();
       __super__printPage();
  }
  void facturation() { count++; }
}
```

```
abstract class Printer_Log {
  abstract __super__printPage();
  void printPage() {
      log();
      __super__printPage();
  }
  void log() {
      System.out.println("printPage"+System.currentTimeMillis();
  } }
```

Mixin transformation

```
class Printer {
     int count = 0;
     void printPage() {
          facturation();
             log();
             <fontionnalité de base>
     }
     void facturation() {
        count++;
     }
     void log() {
        System.out.println("printPage"+
        System.currentTimeMillis();
     }
}
```

Manipulation d'AST

- Principe
  - Parcours/Modification de l'AST d'un programme (ie ensemble de sources annotées ou non) récupéré après l'analyse du compilateur
- Plus et moins
  -  + les sources (ie les templates) doivent être validés
    - Mise au point avec des IDE standards (pas de plugins non maintenus)
  -  - les sources doivent être validés
  -  - plutôt « compile-time »
    - Car coût en temps, en mémoire et en mémoire secondaire
    - Cependant les JRE embarquent parfois un compilateur
      - JSR-199 Java™ Compiler API
- Exemple de canevas
  - APT, Eclipse JDT compiler, Spoon/SpooNet, CodeDom

APT (Annotation Processing Tool)

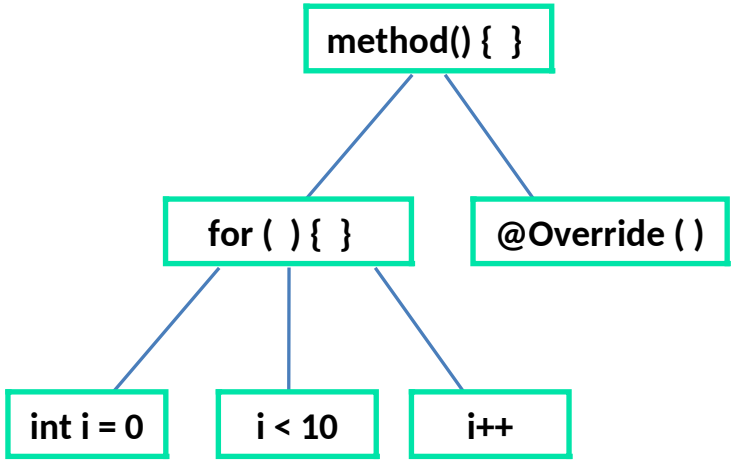
http://java.sun.com/javase/6/docs/technotes/guides/apt/index.html

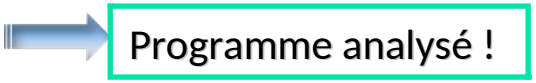
- JDK annotations processing tool
  - Read-only and only data-level (no code level)
- JSR 269 Pluggable Annotation Processing API

Spoon : Généralités

- http://spoon.gforge.inria.fr
- Outil de Méta-Programmation pour Java 5.0
- Agit au niveau source du programme
  - Arbres Syntaxiques Abstraits (= AST : Abstract Syntax Tree)
- Processeur Spoon :
  - Parcours d'AST : selon la patron de conception Visiteur
  - Peut être spécialisé par type de nœud (utilise les Generic)
  - Peut modifier les noeuds

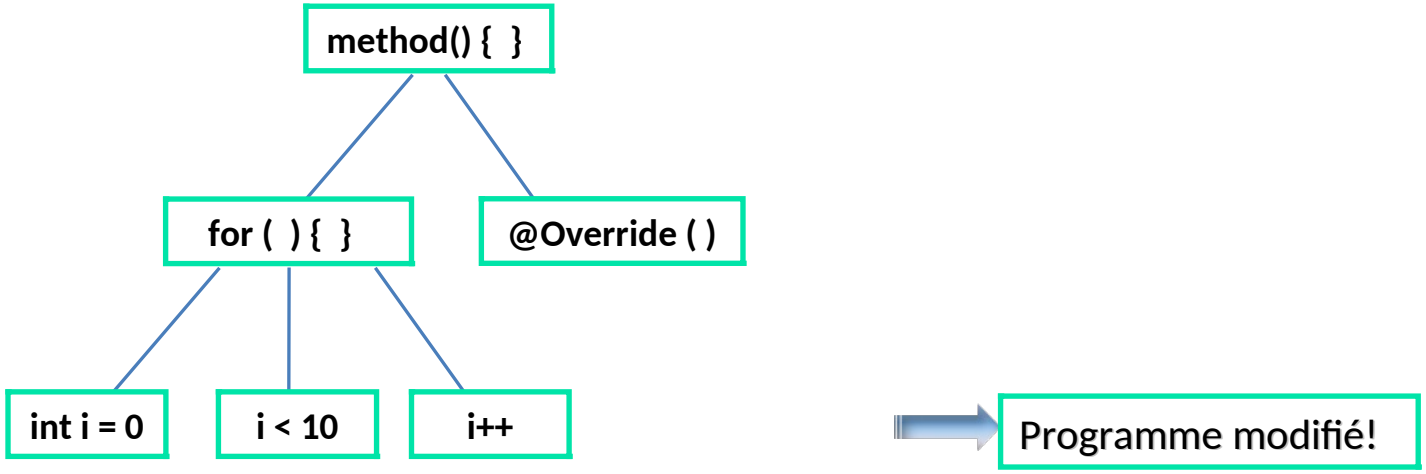
- Analyse de code :





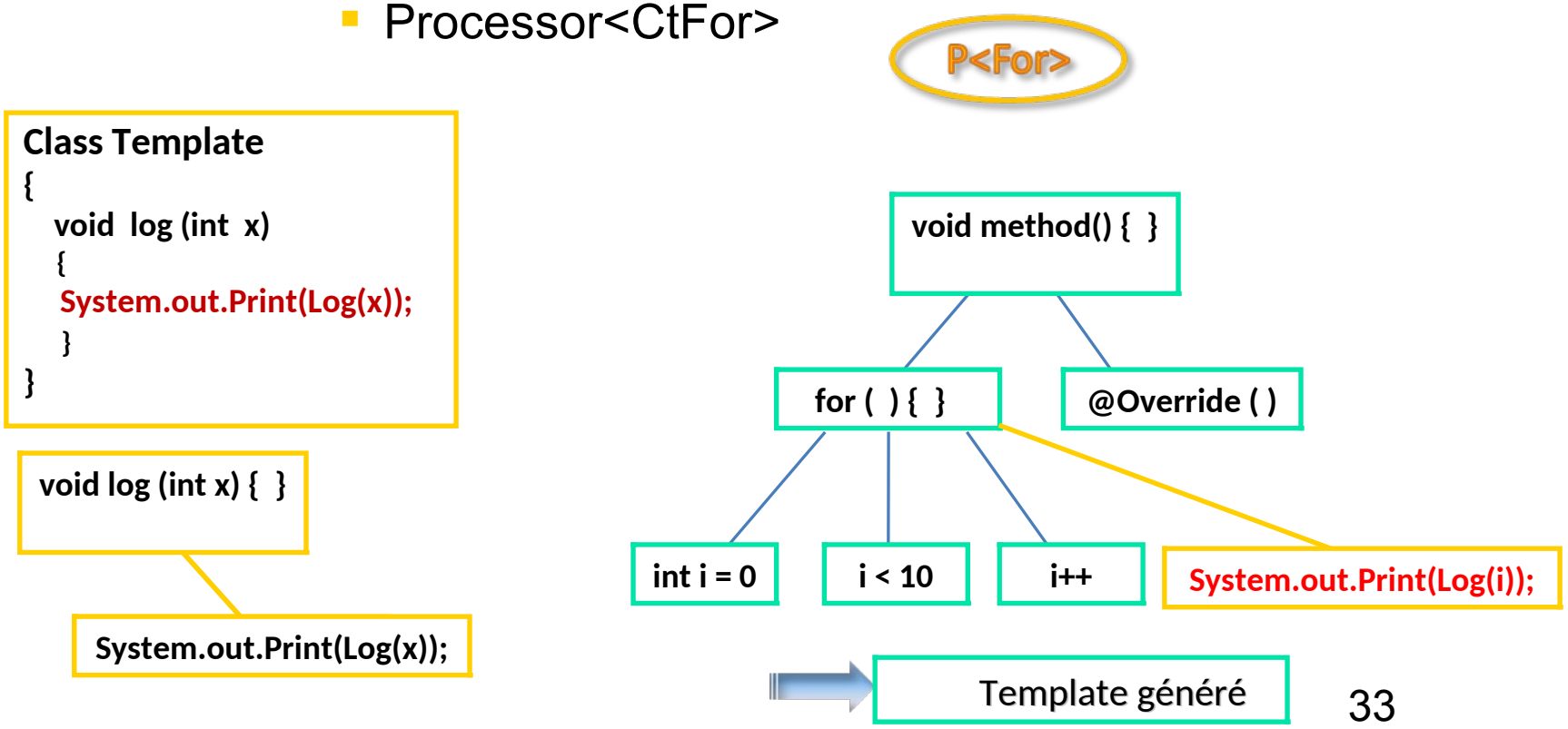
Spoon : Généralités

- Modification de code, exemple :
  - Processor<CtAnnotation>
  - Process() : Supprimer l'annotation

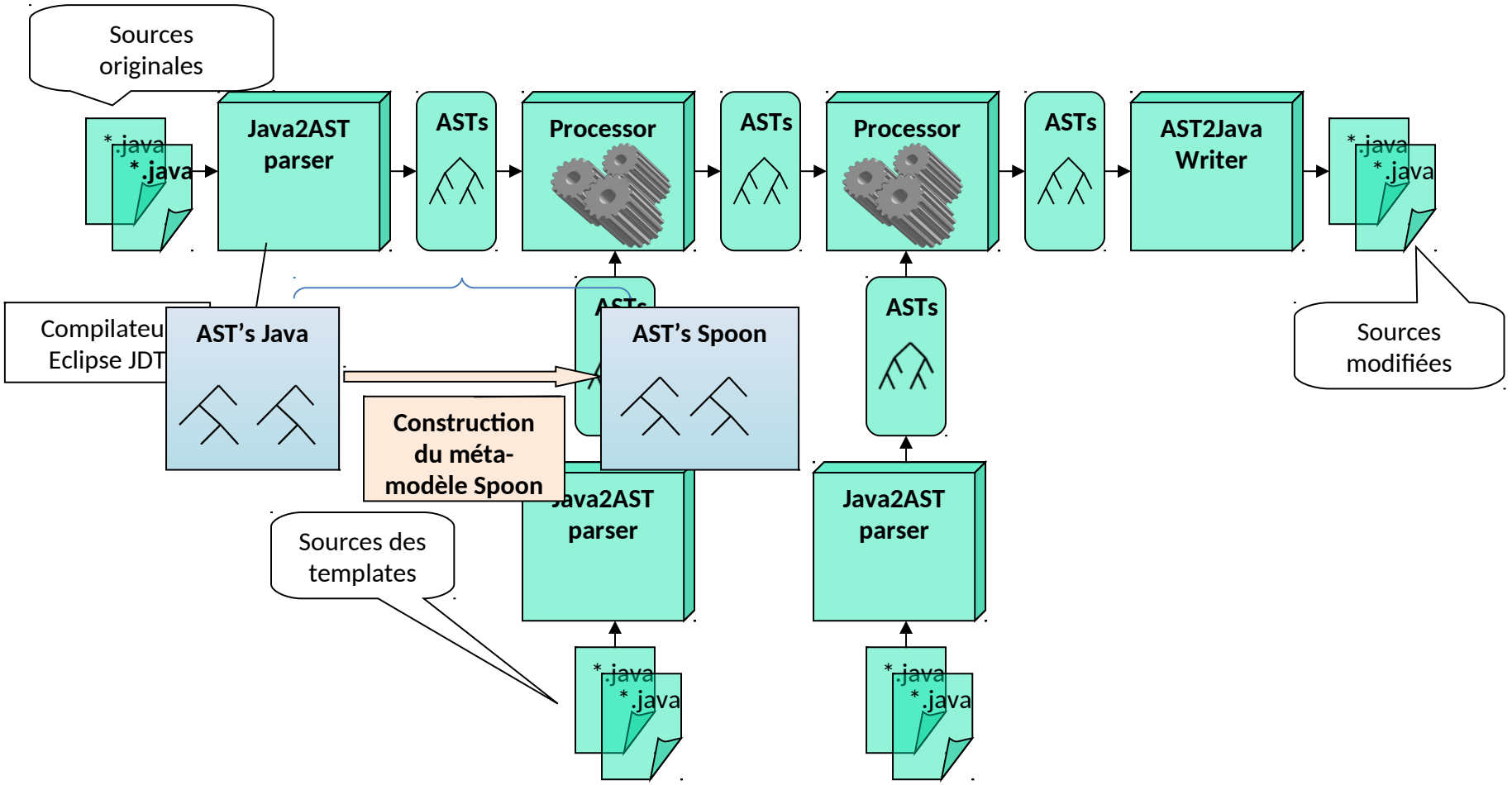


Spoon : Généralités

- Aspect Génération de code :
  - Selon des templates (patrons, gabarits) écrits en pur Java
  - Exemple : insérer une instruction dans les boucles « for »
    - Classe Template qui contient le code à insérer
    - Processor<CtFor>



Chaîne de fonctionnement de Spoon



- Chaîne intégrée dans Eclipse (plugin SpoonJDT)
- Processeurs disponibles : Jdiet, Aval, Vsuite, SpoonAOP…
- SpooNet: version pour C# (.NET)

Summary

- Bytecode injection
  - bytecode level (ASM, BCEL, …)
    - error-prone, hard maintenance, hard to optimize, …
  - fits well on-the-fly class manipulation
- Text-based Templates
  - ASCII level (XSLT, Velocity, Eclipse CodGen JET, Jelly s…)
  - hard to debug and maintain
    - Non modular
    - templates can not be validated by an off-the-shelf compiler/IDE
      - But easy to understand
- AST Manipulation
  - APT & JSR 269 Pluggable Annotation Processing API
    - JDK annotations processing tool
    - Read-only and only data-level (no code level)
  - Spoon & SpooNet
    - source level (source validation)
    - AOT but a on-the-fly spoon exists
    - better performance ?
      - VM JITs optimize bytecodes produced by off-the-shelf compilers (`javac,csc,…`)
        - inlining, …

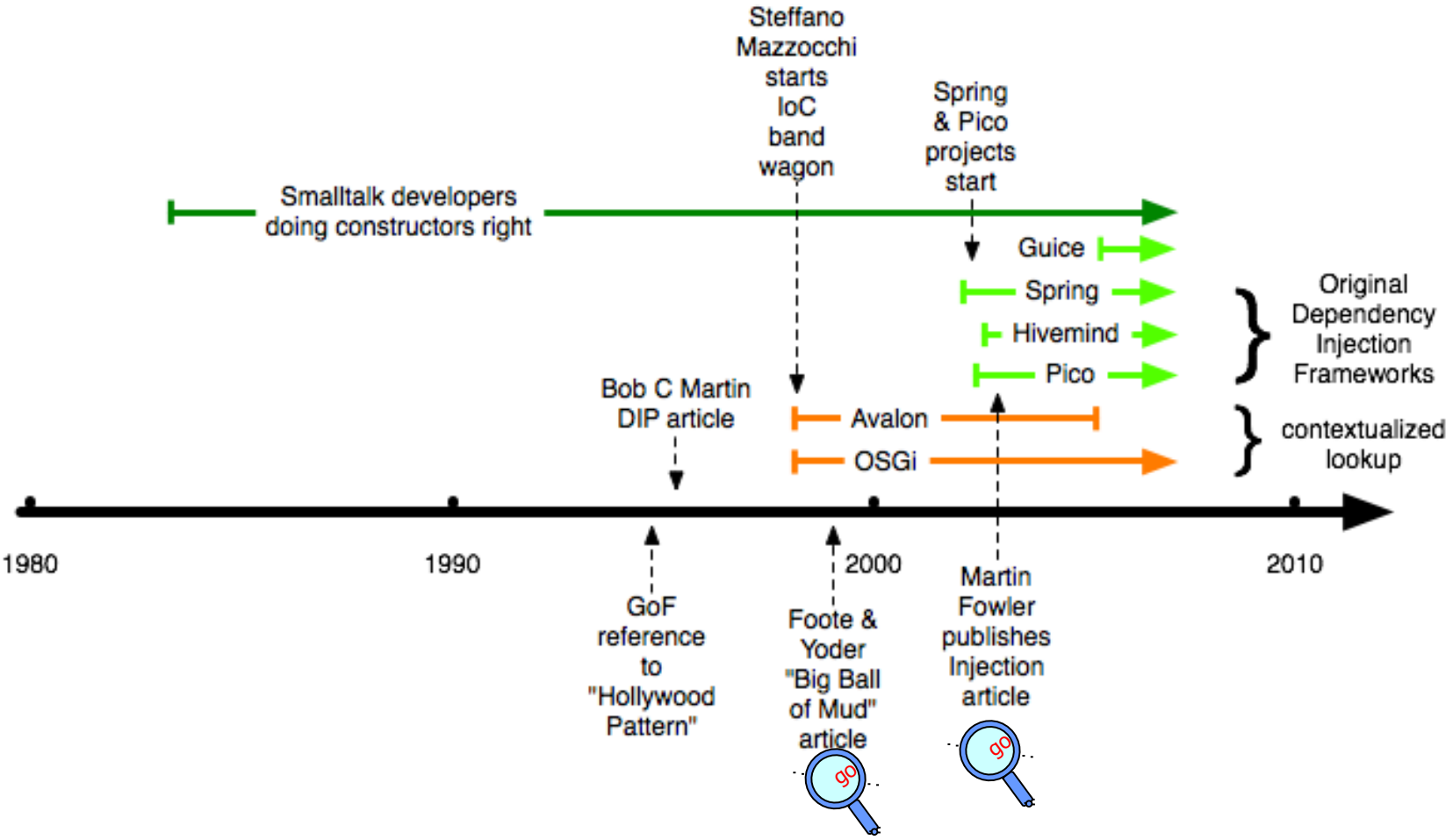
VM Instrumentation

- Motivation
  - Modifier le comportement *« standard »* de la VM
  -  performance
  -  performance (car appliqué à toutes les classes), portabilité (écrit en langage natif), déploiement statique des agents (.dll, .so) …
- JVMTI JVM Tool Interface
  - http://java.sun.com/javase/6/docs/technotes/guides/jvmti
  - Ajout d'agents natifs pour observer/superviser les événements de la VM
- VVM Virtual Virtual Machine
  - Notion de VMLet
    - http://pagesperso-systeme.lip6.fr/Gael.Thomas/papers/these-gael.thomas.pdf

Rappel sur l'injection de dépendances

- Technique to reduce coupling by moving configuration and dependency wiring outside a component
  - Helps design loosely coupled components
    - Improves testability
    - Simplifies unit testing
    - Increases flexibility and maintainability
    - Minimizes repetition
    - Supplies a plug architecture
    - Relies on interfaces
- Other terms
  - Hollywood principle
    - *Don't call us – we'll call you!*
  - Inversion of control

Rappel sur l'injection de dépendances



Martin Fowler , Inversion of Control Containers and the Dependency Injection pattern, 2003, http://www.martinfowler.com/articles/injection.html

Rappel sur l'injection de dépendances Types d'injections

- Method Dependency Injection (called Type 1)
  - Avalon
- Setter Dependency Injection (SDI) (called Type 2)
  - Spring
- Constructor Dependency Injection (CDI) (called Type 3)
  - PicoContainer

- Annotated Field Dependency Injection
  - H2K, Guice, Spring Tiger
- Annotated Method Dependency Injection

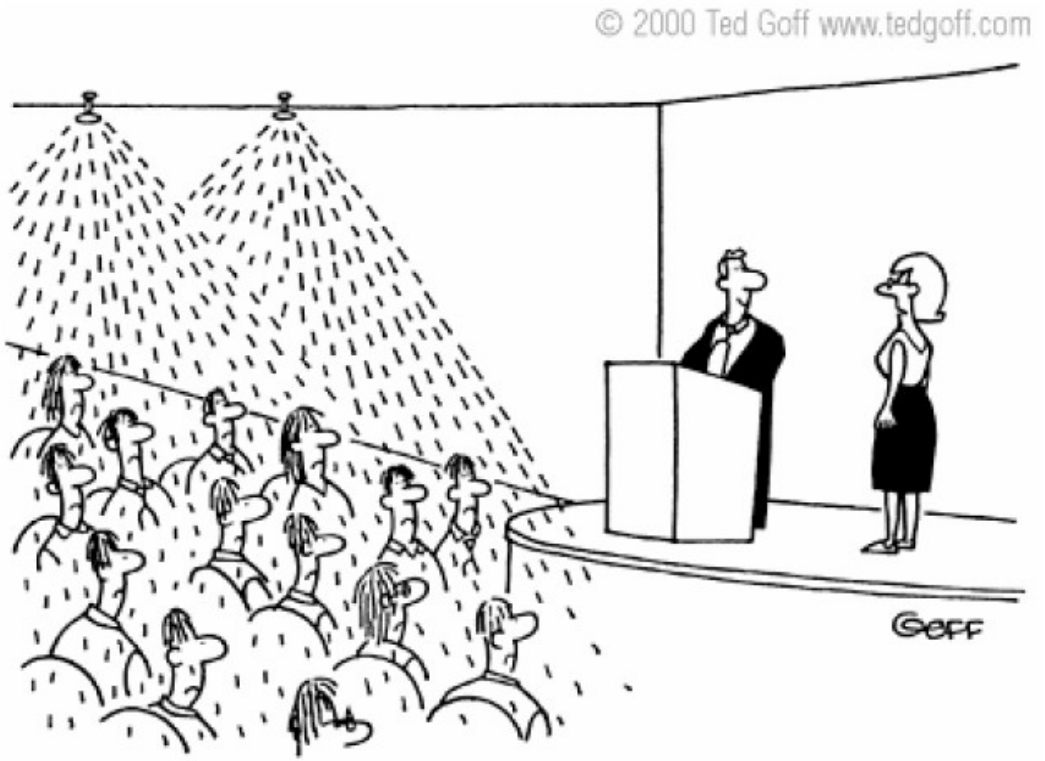
- JSR 330: Dependency Injection for Java
  - maximize reusability, testability and maintainability of Java code by standardizing an extensible dependency injection API.

http://www.picocontainer.org/injection.html

Homework

- JHipster generator
  - https://github.com/jhipster/generator-jhipster
  - https://github.com/jhipster/generator-jhipster/tree/master/generators
- Yeoman
  - http://yeoman.io/generators

Questions & Answers



"You're not allowed to use the sprinkler system to keep your audience awake."