http://membres-liglab.imag.fr/donsez/cours

Ingénierie des Conteneurs (de Composants)

Didier Donsez

Université Joseph Fourier – Grenoble 1

PolyTech' Grenoble - LIG / Adèle

Prenom.Nom@imag.fr

Firstname.Lastname@ieee.org

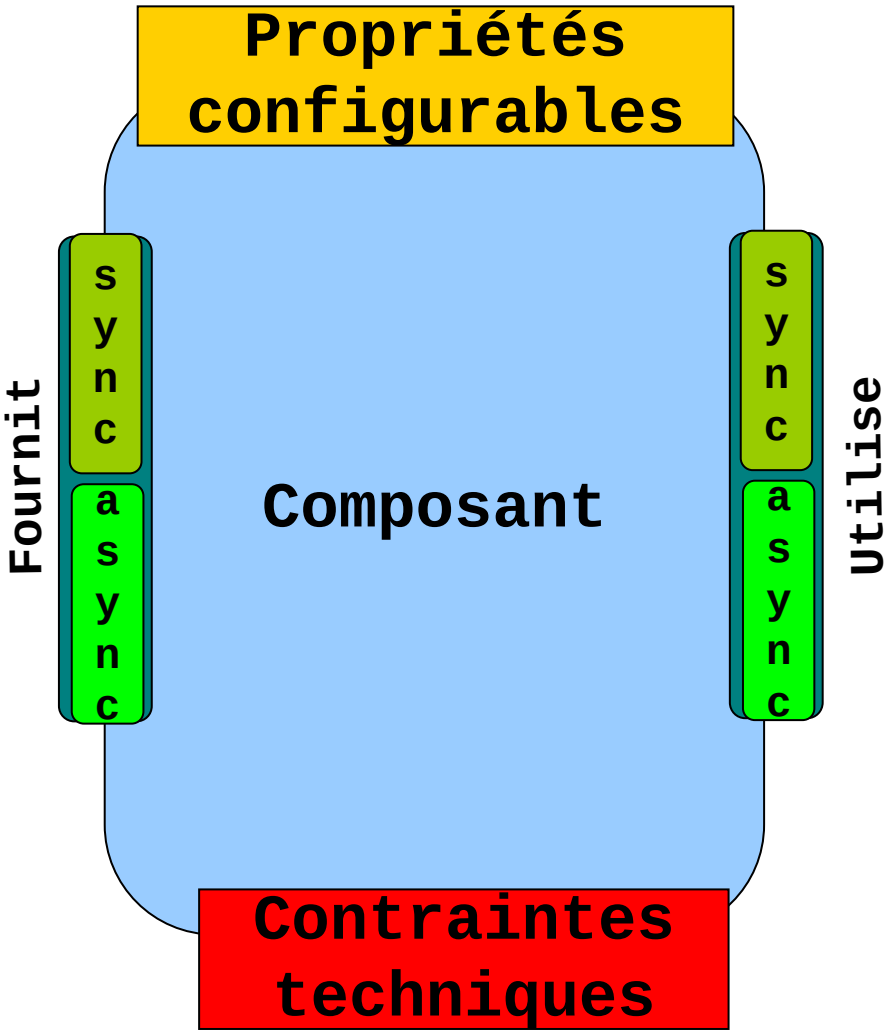
Ce cours ne s'intéresse pas à ce type de conteneurs

Sommaire

- Rappel sur les composants
- Metadonnées
- Temps de génération
  - Compilation, Déploiement, Chargement, Exécution
- Compromis de conception
  - Performance, flexibilité, optimisation
- Techniques
  - Réflexion
    - Proxy dynamiques
  - Génération de bytecode/IL
    - ASM, BCEL, Jabyce, .NET System.Reflect
  - Génération de sources
    - Velocity/NVelocity, Jelly, XSLT
  - Mixin
    - Julia, Julias
  - Manipulation d'AST
    - APT, Spoon, CodeDom
  - Injecteur d'aspects
    - AspectJ, AspectWerz, Guice
- Projets de conteneurs
  - PicoConteneur,HK2, …

Rappel sur les modèles de composants

- Comment coopère un composant
  - Ce que fournit le composant (entrées)
    - composantes, interfaces, opérations, propriétés
  - Ce qu'utilise le composant (dépendances)
    - composition et références aux autres composants
  - modes de communication des connecteurs (synchrone, asynchrone, flots, …)
- Propriétés configurables du composant
- Contraintes techniques (QoS)
  - middleware : placement, sécurité, transaction
  - internes : cycle de vie, persistance
  - implantations : OS, bibliothèques, version



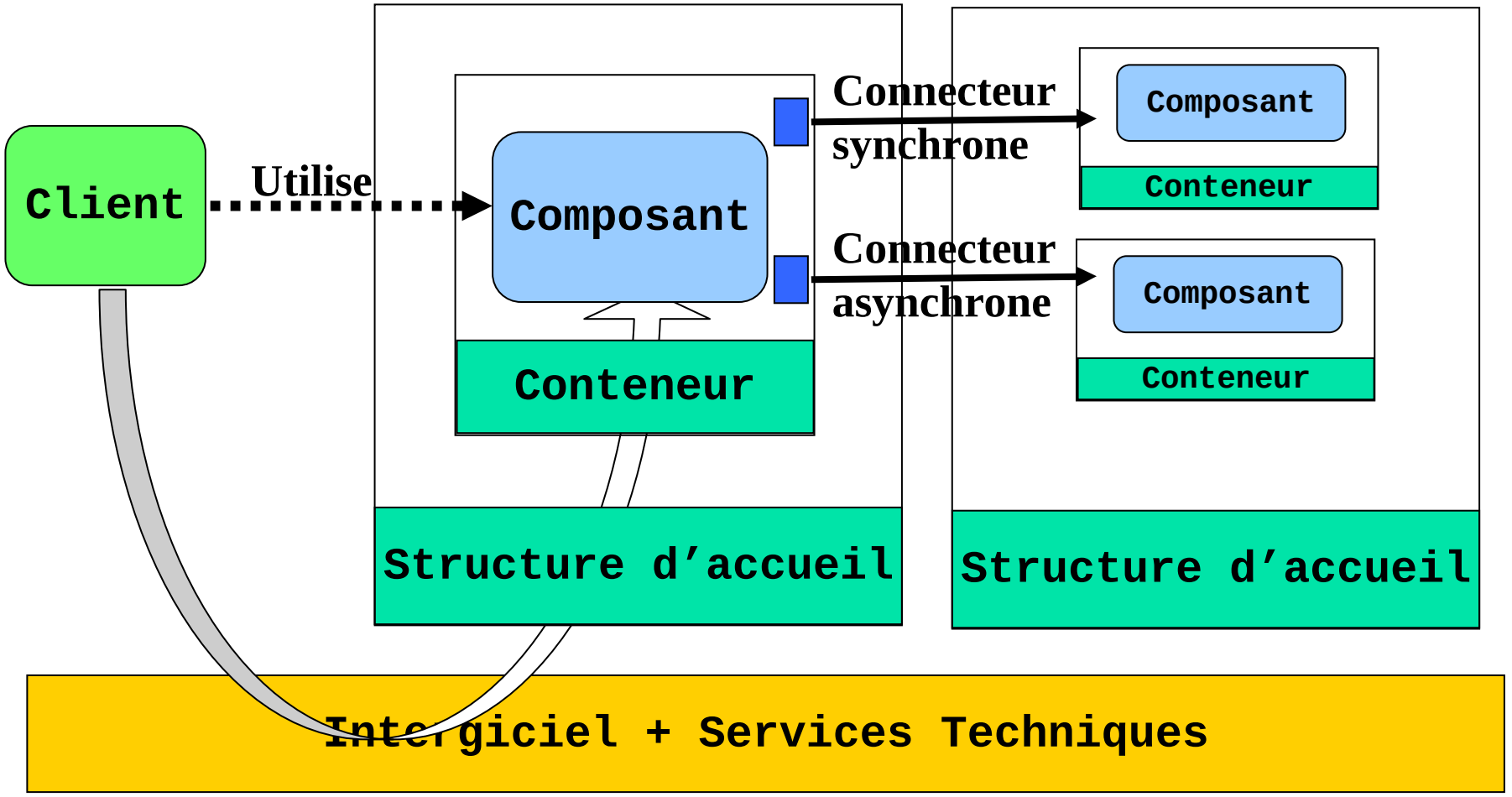
D'après [Riveill & Merle]

Rappel sur les modèles de composants

- Conteneur
  - encapsulation d'un composant (et ses composantes)
  - prise en charge (masque) les services systèmes
    - nommage, sécurité, transaction, persistance ...
  - prise en charge partielle des connecteurs
    - invocations et événements
  - techniquement par interposition (ou délégation)
- Structures d'accueil
  - espace d'exécution des conteneurs et des composants
  - médiateur entre les conteneurs et les services systèmes
  - des + comme le téléchargement de code (navigateur)

D'après [Riveill & Merle]

Rappel sur les modèles de composants



D'après [Riveill & Merle]

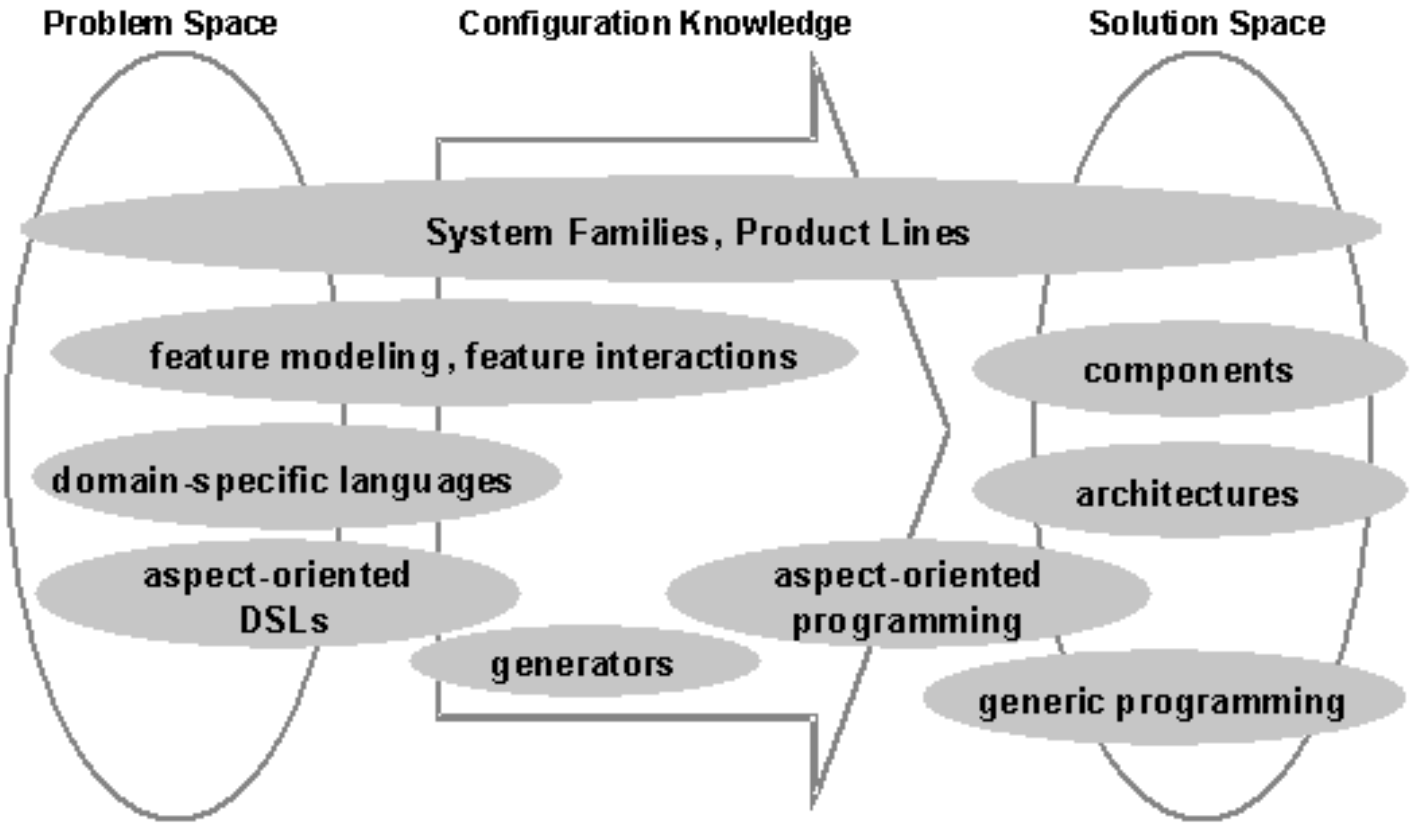
Rappel sur la programmation générative

- 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

Rappel sur la programmation générative



From [Czarnecki & Eisenecker]

Méta-données

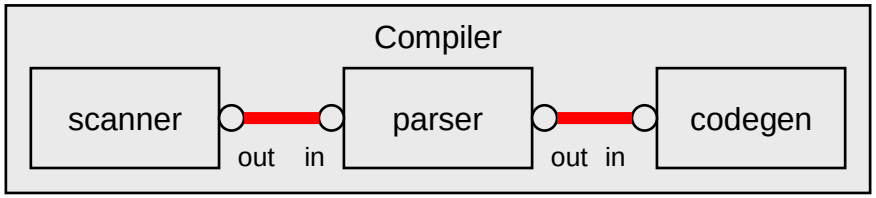
- Nécessité de décrire et configurer le composant
  - Interfaces requises et fournis, propriétés, …
  - Liaisons entre sous-composants
- Réuni vs Séparé des sources
  - Format XML
    - externe, indépendance au langage, validation structurelle/valeur (*XML schema*) , modulaire (*namespace*), expressions régulières pour un sous-ensemble de classes (***/Test**)
    - cohérence description ←→ implémentation (requière des plugins)
  - Attributs .NET, XDocLet, Annotations Java 5, Macro CPP, …
    - validation pour le compilateur du langage de l'implémentation du composant (moins de risque d'erreur)
    - validation structurelle (Spoon/Aval), annotation classe par classe
- Remarque: *c'est une tendance lourde !*
  - Obligation de mélanger les 2
    - POJO, close-source, COTS, *legacy software*, …

Méta-données

- Modèles ayant migrer vers les annotations
  - EJB 2 → EJB3
  - Spring → Spring/Tiger
  - Fractal ADL → Fraclet
    - http://fractal.objectweb.org/tutorials/fraclet/fracletannotation.html
    - http://fractal.objectweb.org/tutorials/fraclet/index.html
- Modèles faisant coexister les annotations et les descriptions XML
  - EJB3, Web Beans (JSR 299), …
- Autres modèles utilisant les annotations
  - MBean, JAXB, JAXR, SIPServlet …
- Généralement conversion XML ←→ Annotations

Méta-données
Exercice avec ArchJava

- Donnez un ensemble des annotations Java 5 qui permettraient de décrire un composant ArchJava (Java5)
- Rappel sur ArchJava




```
public component class Parser {
  public port in {
    requires Token nextToken();
  }
  public port out {
    provides AST parse();
  }
  AST parse() {
    Token tok=in.nextToken();
    return parseExpr(tok);
  }
  AST parseExpr(Token tok) { ... }
  ...
}
```

```
public component class Compiler {
  private final Scanner scanner = new Scanner();
  private final Parser parser = new Parser();
  private final CodeGen codegen = new CodeGen();
  connect scanner.out, parser.in;
  connect parser.out, codegen.in;
}
```

Méta-données
Réponse à l'exercice avec ArchJava

- Annotations
  - @Component (@Port[] ports, @Connect[] connections)
  - @Instance(String id)
  - @Port(String id)
  - @Requires(String portId)
  - @Provides(String portId)
  - @Connect(String instance1Id, String port1Id, String instance2Id, String port2Id)
- Composants

```
@Component(
    ports={@Port("in"), @Port("out")})
public class Parser {
  @Requires("in")
  Requires_In in;

  @Provides("out")
  AST parse() {
    Token tok=in.nextToken();
    return parseExpr(tok);
  }

  AST parseExpr(Token tok) { ... }
  ...
}
```

```
@Component(connections={
  @Connect("scanner","out", "parser","in"),
  @Connect("parser", "out", "codegen", "in")
})
public class Compiler {
  @Instance("scanner")
  private final Scanner s = new Scanner();
  @Instance
  private final Parser parser = new Parser();
  @Instance
  private final CodeGen codegen = new CodeGen();
}
```

Meme exercice avec des Doclets …

Aval (Spoon project)
http://spoon.gforge.inria.fr/AVal/Main

- extendable *meta-annotation* based Annotation Validator for java 5.
- provides a set of basic validation rules, as well as means to define more domain-specific ones.

- Structural validation
  - @AValTarget : the annotation concerns a element : CtClass, CtMethod, CtField, CtAnnotation, …
  - @Inside
  - @Prohibits
  - @Requires
  - @Type
- Value validation
  - @Matches : value must match a regular expression
  - @RefersTo : references a
  - @Unique : value must be unique in all the analysed sources
  - @URLValue : value must match a URL format
- Run as
  - Ant task, Eclipse plugin

Aval
Simple Example

```
@Web
public class MyWebPage{

    @Page
    public void home(){
       //Some code
    }

    @Page(dateCreated="2007-12-29")
    public void about(){
    }

    @Link("http://www.google.com")
    public void anExternalLink(){
    }
}
```

```
@AValTarget(CtClass.class)
public @interface Web{}
```

```
@AValTarget(CtMethod.class)
@Inside(Web.class)
@Prohibits(Link.class)
@Type(void.class)
public @interface Page {
   @Matches("[\\d]{4}-[\\d]{2}-[\\d]{2}")
   String dateCreated() default "";
}
```

```
@AValTarget(CtMethod.class)
@Inside(Web.class)
@Type(void.class)
public @interface Link {
   @URLValue
   String value();
}
```

Aval
Simple Example

```
@Web
public class MyWebPage{

    @Page
    public String home(){
       //Some code
    }

    @Page(dateCreated="Dec. 2007")
    public void about(){
    }

    @Link("www.google.com")
    public void anExternalLink(){
    }
}
```

```
@AValTarget(CtClass.class)
public @interface Web{}
```

```
public @interface Link {
   @URLValue
   String value();
}
```

Error: The use of @dsl.Page must be defined on an element inside @dsl.Web annotation at src.MyWebPage.about(MyWebPage.java:13)
Error: Value "Dec. 2007" does not match [\d]{4}-[\d]{2}-[\d]{2} at src.MyWebPage.about(MyWebPage.java:13)
Error: The use of @dsl.Link must be defined on an element inside @dsl.Web annotation at src.MyWebPage.anExternalLink(MyWebPage.java:17)
Error: Value "www.google.com" is supposed to be an URL at src.MyWebPage.anExternalLink(MyWebPage.java:17)
Error: The use of @dsl.Page must be defined on an element inside @dsl.Web annotation at src.MyWebPage.home(MyWebPage.java:8)
Warning: Annotation @Page is only allowed on elements of type void at src.MyWebPage.(MyWebPage.java:9)
1 warning 5 errors

Exercice Aval

- Annoter les annotations ArchJava 5
  avec les annotations Aval

Temps de génération

- Compilation
  - La génération est réalisée
- Assemblage
- Déploiement
- Chargement
- Exécution

Compromis de conception

- Optimisation & Performance
  - Operation invocation & memory footprint & GC
- Flexibilité
  - Reconfiguration à chaud (dynamique)
  - vs static (ie stop, rebuild the container, restart)

Techniques de construction

- Réflexion
  - Proxy dynamiques
- Génération de sources
  - Velocity/NVelocity, Jelly, XSLT
- Génération de bytecode/IL
  - ASM, BCEL, Jabyce, CGLib (http://cglib.sourceforge.net/) .NET System.Reflect
- Mixin
  - Julia, Julias
- Manipulation d'AST
  - APT, Spoon, CodeDom
- Tisseurs (weavers) d'aspects
  - AspectJ, AspectWerz, Guice

Réflexion

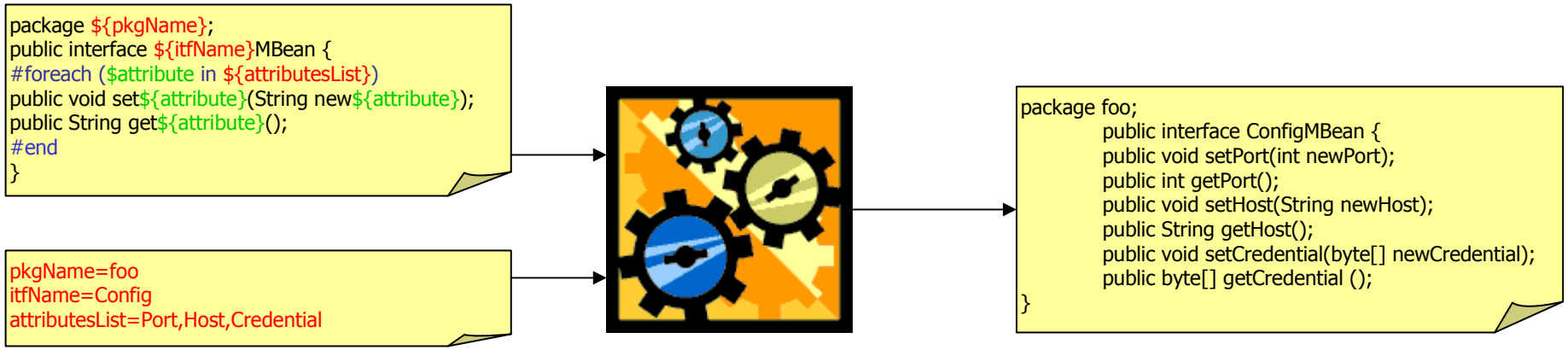
- Langages reflexifs (Smalltalk, Java, C#, C++, JavaScript …)
  - Représentation du programme
    - Méthodes, champs, annotations, instructions …
  - Capacité du programme à s'introspecter (w/o le modifier)

- 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

Génération de sources (texte)

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



- 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();
```

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);
 <%
    }
 %>
}
```

Génération de bytecode/IL

- 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, …
- Canevas
  - 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

Tisseurs d'aspects

- 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
- Utilisable pour construire un conteneur
  - Cas d'AOKell (v1 avec AspectJ)
    - http://www2.lifl.fr/~seinturi/papers/fractal-ecoop06-fractnet.pdf

Conteneur d'AOKell avec AspectJ

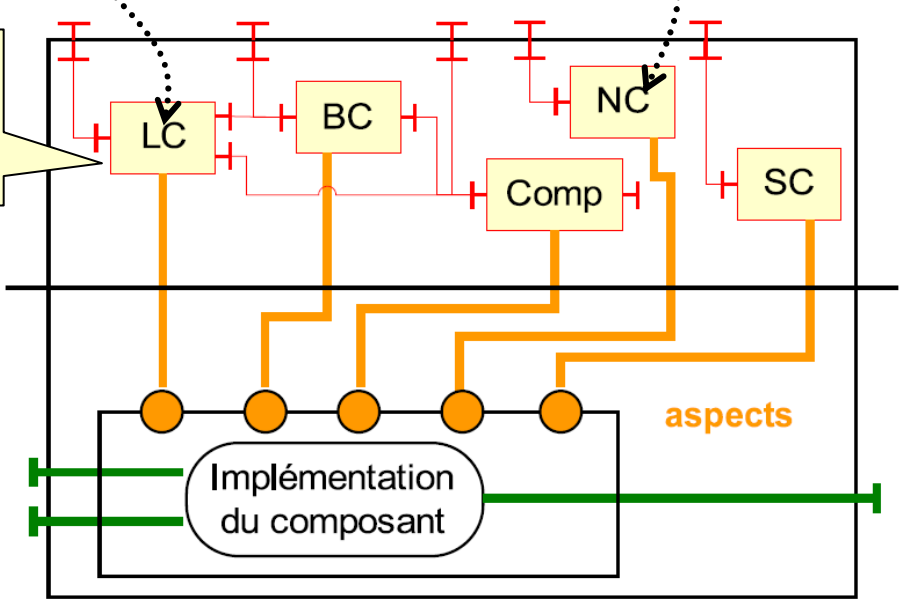
```
public aspect ALifeCycleController {

  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

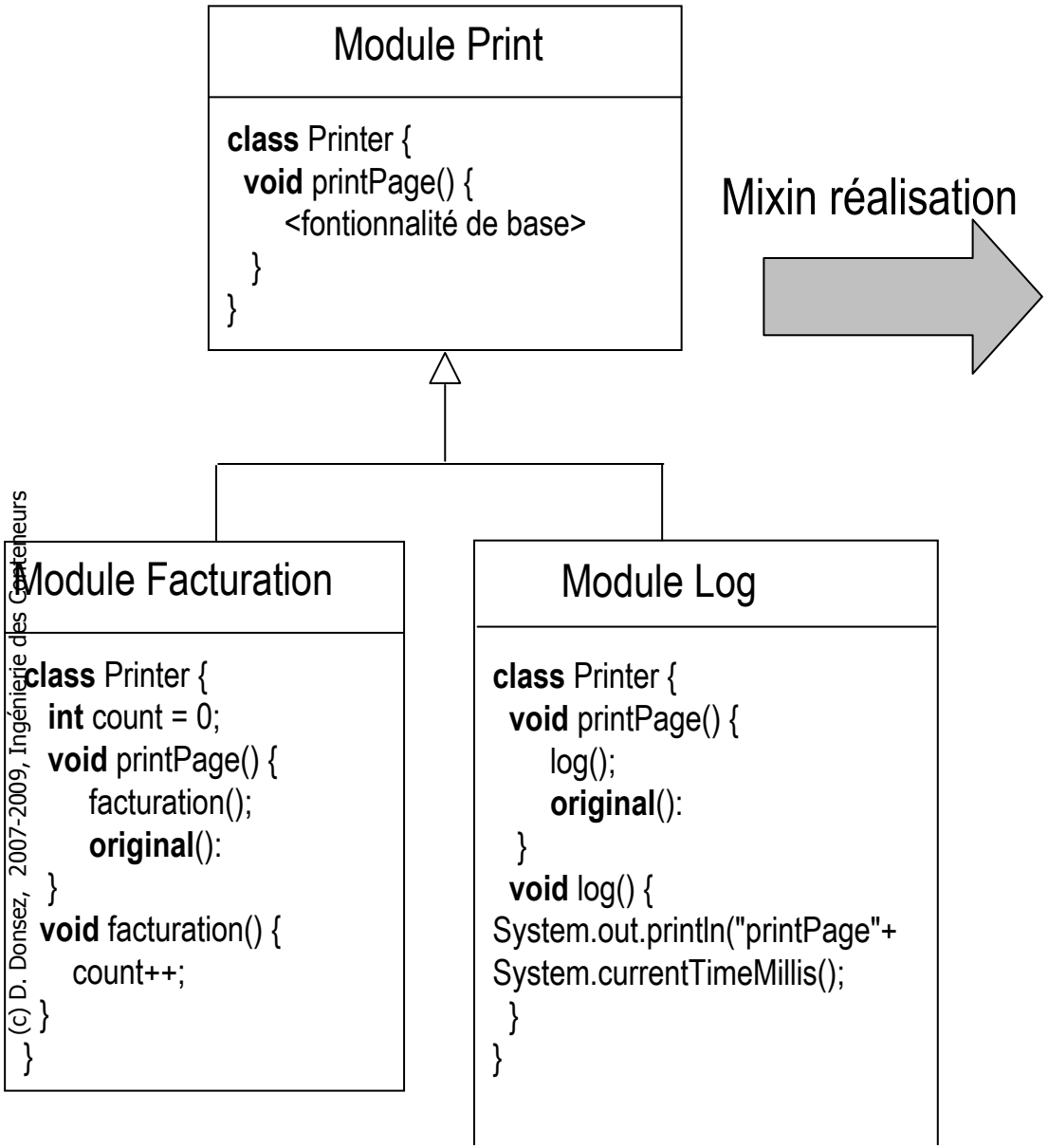
d'après http://www.emn.fr/x-info/jte-aets/seinturier.pdf

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* …
- 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)

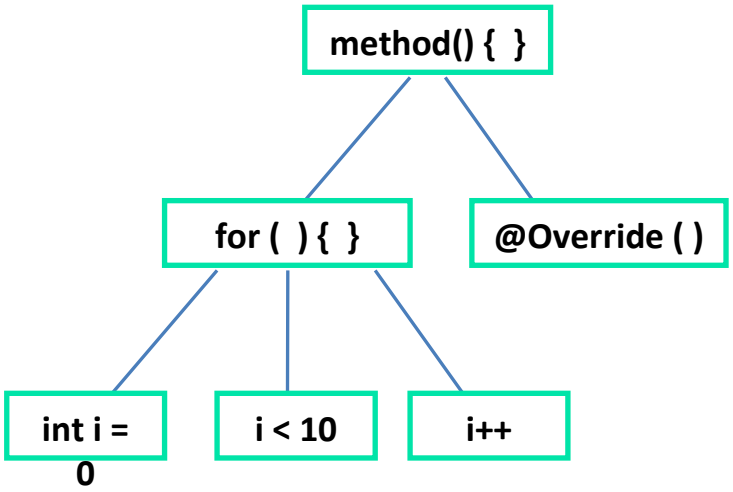
http://java.sun.com/javase/6/docs/technotes/guides/apt/index.html

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

Spoon : Généralités

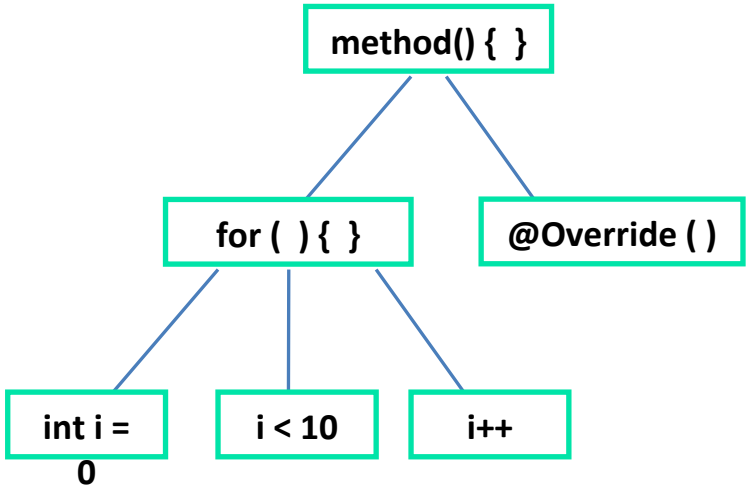
- http://spoon.gforge.inria.fr
- Outil de Méta-Programmation pour Java 5.0
- Agit au niveau source du programme
  - Arbres Syntaxiques Abstraits (= AST : Abstract Syntax Tree)
- Processeur Spoon :
  - Parcours d'AST : selon la patron de conception Visiteur
  - Peut être spécialisé par type de nœud (utilise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

- Analyse de code :



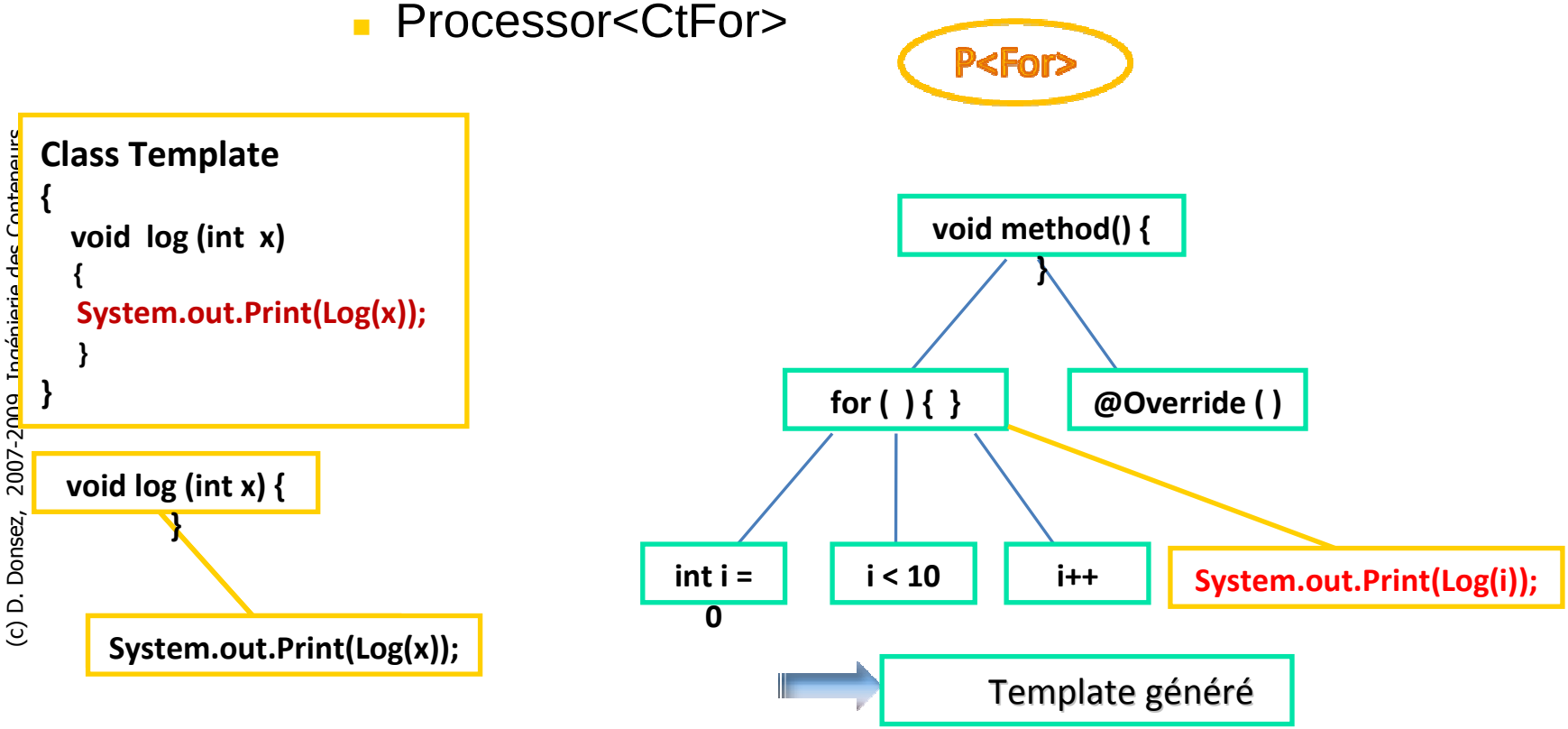
Spoon : Généralités

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

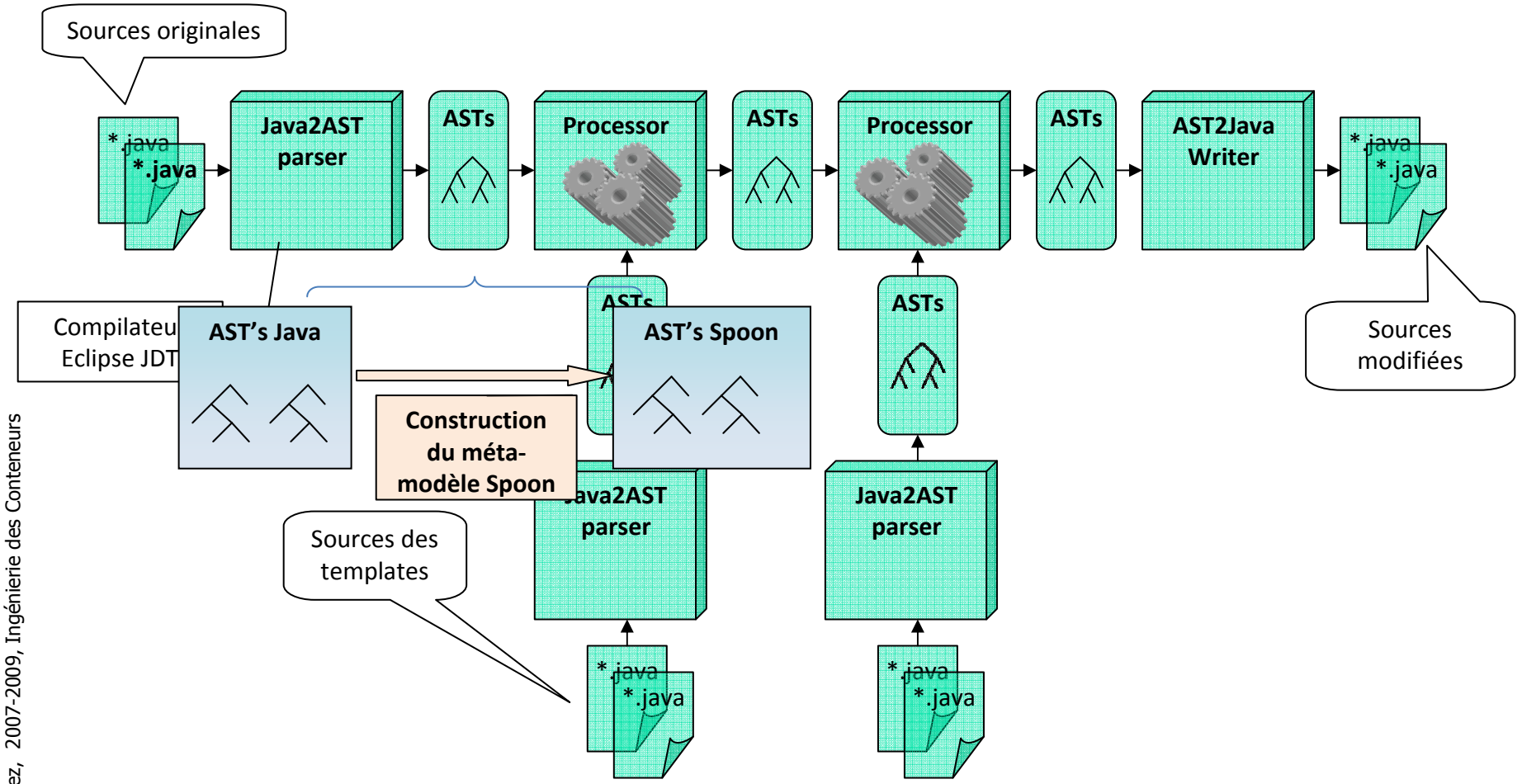


Spoon : Généralités

- 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, …

Summary

- TODO
- Matrix technique / generation time, … more

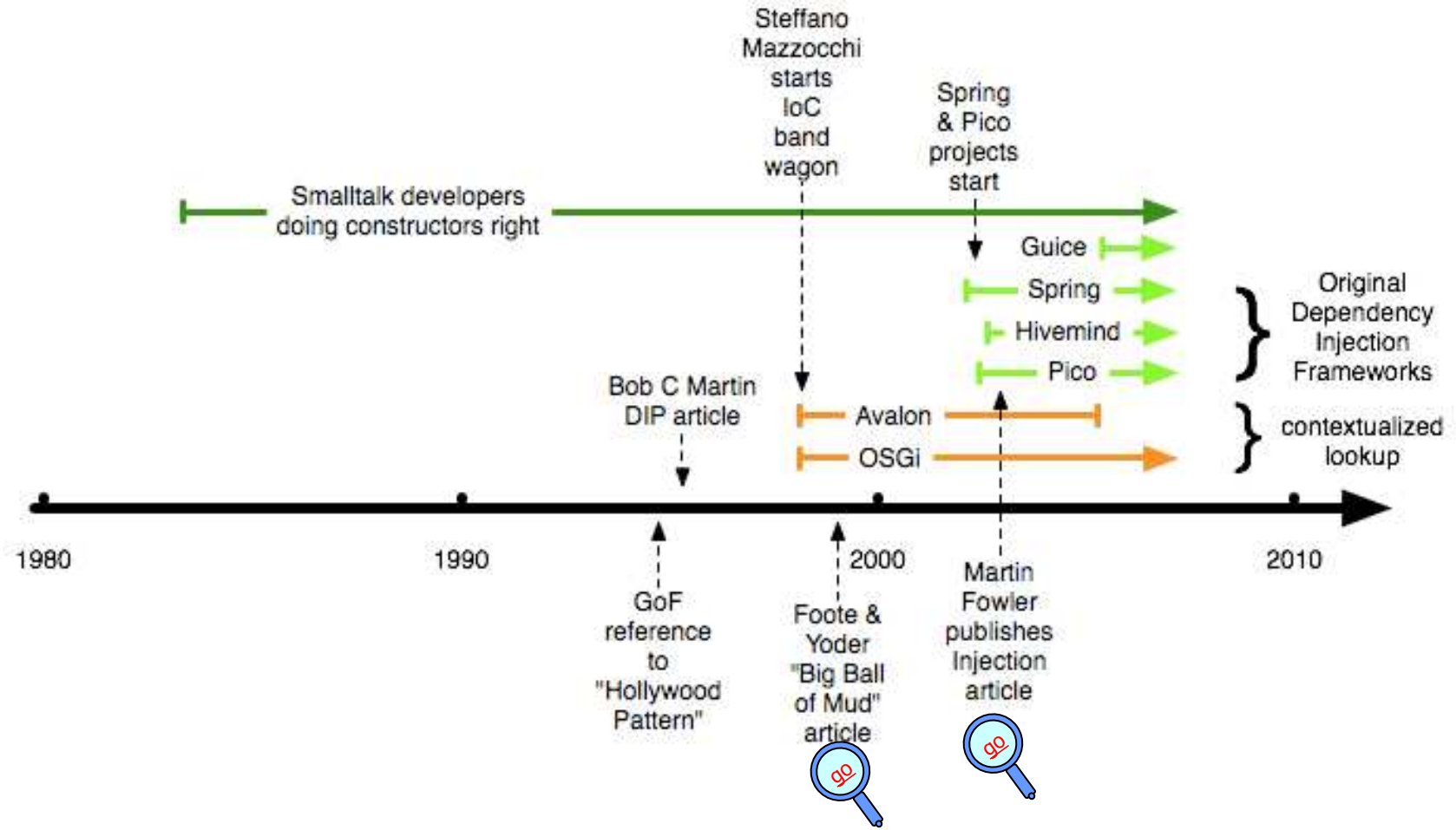
Instrumentation de VMs (To expand)

- 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

Containers

- Mainly based on IoC & IoD
- Containers
  - For Java
    - PicoContainer (used also Java 5 annotations)
    - Hivemind
    - Avalon
      - → Excalibur Fortress,, ECM,, Phoenix,, Merlin
      - →YAAFI (Yet Anotther Avallon Framework Implementation)
    - Spring
    - JBoss MicroContainer (based on JMX)
    - Guice (used only Java 5 annotations)
    - H2K (used only Java 5 annotations)
    - Plexus (Maven core)
    - JSR 330
    - …
  - For .NET
    - Castle MicroKernel/Windsor
    - ObjectBuilder
    - PicoContainer.NET
    - Puzzle.NFactory
    - Spring.NET
    - StructureMap
    - Ninject
- See
  - http://en.wikipedia.org/wiki/Dependency_injection
  - http://henning.schmiedehausen.org/container/
  - http://henning.schmiedehausen.org/container/Apache_Container_20.pdf

PicoContainer http://www.picocontainer.org/

- TODO
- *« PicoContainer is a highly embeddable full service Inversion of Control (IoC) container for components honour the Dependency Injection pattern. »*
  - *supports different dependency injection types (Constructor, Setter, Annotated Field and Method)*
  - *Supports scoped and hierarchical containers*
  - *offers interceptions*
  - *offers multiple lifecycle and monitoring strategies*
  - *very compact in size (the core is ~128K and it has no mandatory dependencies outside the JDK)*

JBoss Microcontainer
http://www.jboss.com/products/jbossmc

- *"… refactoring of JBoss's JMX Microkernel to support direct POJO deployment and standalone use outside the JBoss application server.*
- *Features*
  - *All the features of the JMX Microkernel*
  - *Direct POJO deployment (no need for Standard/XMBean or MBeanProxy)*
  - *Direct IOC style dependency injection*
  - *Improved lifecycle management*
  - *Additional control over dependencies*
- *…"*

HK2 « Hundred Kilobytes Kernel »

https://hk2.dev.java.net

- Small kernel (than OSGi™) to build modular softwares
  - consist of two technologies :
- Modules subsystem
  - offer a better level of isolation between parts of the application.
  - path to the implementation of modules (JSR 277 & 294) in Java SE 7.
- Component Model
  - define components which can also be seen as Services.
    - These components can be automatically and dynamically discovered by the runtime and can use innovative technologies such as Inversion of Control or injection of dependencies as well as automatic resolution of dependencies.
    - Annotations (org.jvnet.hk2.annotations)
      - @Contract, @Service, @Inject, @Extract, @Factory, @Scoped, @ContractProvided, …
- Remark: foundation for the GlassFish V3 application server

HK2 Example

```
@Contract
public interface Responder {
   String echo(String str);
}
```

```
<toy-http-server>
  <http-listener port="1234">
     <greeter greeting="Salut, "/>
  </http-listener>
</toy-http-server>
```

```
@Configure public class ToyHttpServer {
   @Element("*") public List<Object> allBeans;
}
```

```
@Configured public class HttpListener implements ModuleStartup {
   @Attribute public int port;
   @Element("*") public Responder responder;
   public void setStartupContext(StartupContext context) {}
   public void run() { try {
         ServerSocket ss = new ServerSocket(port);
         while(true) {
            Socket s = ss.accept();
            String text = IOUtils.toString(s.getInputStream());
            text = responder.echo(text);
            s.getOutputStream().write(text.getBytes());
            s.close();
         }} catch (IOException e) { throw new Error(e);}}}
```

```
@Configured public class Greeter implements Responder {
   @Attribute public String greeting;
   public String echo(String str) { return greeting+' '+str; }
}
```

Run with mvn hk2:run

HK2 Example

```
@Contract
public interface Responder {
   String echo(String str);
}
```

```
<toy-http-server>
  <http-listener port="1234">
    <upper-case>
      <greeter greeting="Salut, "/>
    </upper-case>
  </http-listener>
</toy-http-server>
```

```
@Configured public class ToyHttpServer {
   @Element("*") public List<Object> allBeans;
}
```

```
@Configured public class HttpListener implements ModuleStartup {
   @Attribute public int port;
   @Element("*") public Responder responder;
   public void setStartupContext(StartupContext context) {}
   public void run() { try {
          ServerSocket ss = new ServerSocket(port);
      while(true) {
         Socket s = ss.accept();
         String text = IOUtils.toString(s.getInputStream());
         text = responder.echo(text);
         s.getOutputStream().write(text.getBytes());
         s.close();
      }} catch (IOException e) { throw new Error(e);}}}
```

```
@Configured public class UpperCase implements Responder{
   @Element("*") public Responder responder;
   public String echo(String str) { return responder.echo(str).toUpperCase(); }}
```

```
@Configured public class Greeter implements Responder {
   @Attribute public String greeting;
   public String echo(String str) { return greeting+' '+str; }
}
```

Plexus

http://plexus.codehaus.org

- provides a full software stack for creating and executing software projects. Based on the Plexus container, the applications can utilise component-oriented programming to build modular, reusable components that can easily be assembled and reused.

- *inversion-of-control/* (IoC) and *dependency injection* framework
- Features
  - Component lifecycles
  - Component instantiation strategies
  - Nested containers
  - Component configuration
  - Auto-wiring
  - Component dependencies (versionned artifacts)
  - Dependency injection techniques
    - constructors injection, setter injection, and private field injection (based on XDocLet).
  - Class loadings
  - « Minimal » footprint : core is a 1MB jarfile

- Used by Apache Maven 2 (mojos) & Plexus Application Server

- See Spring versus Plexus
  - http://plexus.codehaus.org/ref/feature-comparison.html

Plexus example

JSR 330: Dependency Injection for Java

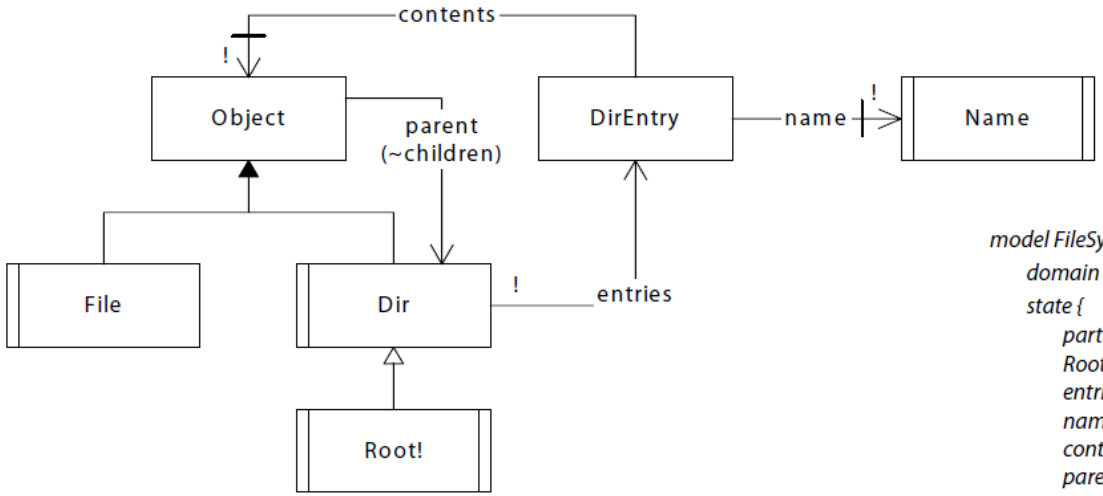
- « maximize reusability, testability and maintainability of Java code by standardizing an extensible dependency injection API ».

Alloy

- TODO
- « little language for describing structural properties »
- Formal validation of component models
- Tools
  - Analyzer Alcoa

- Reading
  - Daniel Jackson. Alloy: A Lightweight Object Modelling Notation. Technical Report 797, MIT Laboratory for Computer Science, Cambridge, MA, February 2000.
  - Daniel Jackson , Ian Schechter , Hya Shlyahter, Alcoa: the alloy constraint analyzer, Proceedings of the 22nd international conference on Software engineering, p.730-733, June 04-11, 2000, Limerick, Ireland http://dx.doi.org/10.1145/337180.337616
- Example
  - Eric Bruneton, Thierry Coupaye, Matthieu Leclercq, Vivien Quéma, Jean-Bernard Stefani: The FRACTAL component model and its support in Java. Softw., Pract. Exper. 36(11-12): 1257-1284 (2006)
    - http://dx.doi.org/10.1002/spe.767

Alloy Example




```
model FileSystem {
    domain {Object, DirEntry, fixed Name}
    state {
        partition File, Dir : static Object
        Root: fixed Dir!
        entries: Dir! -> DirEntry
        name: DirEntry -> static Name!
        contents: DirEntry -> static Object!
        parent (~children) : Object -> Dir
        }
    def parent { all o | o.parent = o.~contents.~entries }
    inv UniqueNames {all d | all e1, e2: d.entries | e1.name = e2.name -> e1 = e2}
    inv Parents {
        no Root.parent
        all d: Dir - Root | one d.parent }
    inv Acyclic {no d | d in d.+parent}
    inv Reachable {Object in Root.*children}
    cond TwoDeep {some Root.children.children}
    assert FileHasEntry {all f: File | some d | f in d.entries.contents}
    assert AtMostOneParent {all o | sole o.parent}
    op NewDirEntries (d: Dir, es: DirEntry') {
        no es & DirEntry
        d.entries' = d.entries + es
        all x: Dir - d | x.entries' = x.entries }
    op Create (d: Dir!, o: Object'!, n: Name) {
        n !in d.entries.name
        some e: DirEntry' | NewDirEntries (d, e) && e.contents' = o && e.name' = n }
    assert EntriesCreated {all d: Dir, e: DirEntry' | NewDirEntries (d,e) -> DirEntry' = DirEntry + e}
    assert CreateWorks {all d, o, n | Create (d,o,n) -> o in d.children'}
    }
```

From Daniel Jackson. Alloy: A Lightweight Object Modelling Notation. Technical Report 797, MIT Laboratory for Computer Science, Cambridge, MA, February 2000

Conferences

- Principalement
  - GPCE (Generative Programming and Component Engineering)
  - AOSD (Aspect-Oriented Software Development conference)
    - http://www.aosd.net/
- Egalement
  - Component-Based Software Engineering (CBSE)
    - http://www.informatik.uni-trier.de/~ley/db/conf/cbse/index.html
  - Euromicro Conference on Software Engineering and Advanced Applications (SEAA) track CBSE
    - http://www.informatik.uni-trier.de/~ley/db/conf/euromicro/index.html
  - Software Composition (SC)
    - http://www.informatik.uni-trier.de/~ley/db/conf/soco/index.html
  - Workshops de ICSE, OOPLSA, ETAPS, …

Questions & Réponses