Serenitec

Un atelier de refactoring java automatisé

ORANGE OBS IT&Labs, CAPS Entreprise, ALF IRISA/INRIA

date de la présentation : 4/12/2009

lieu de la réunion : RENNES

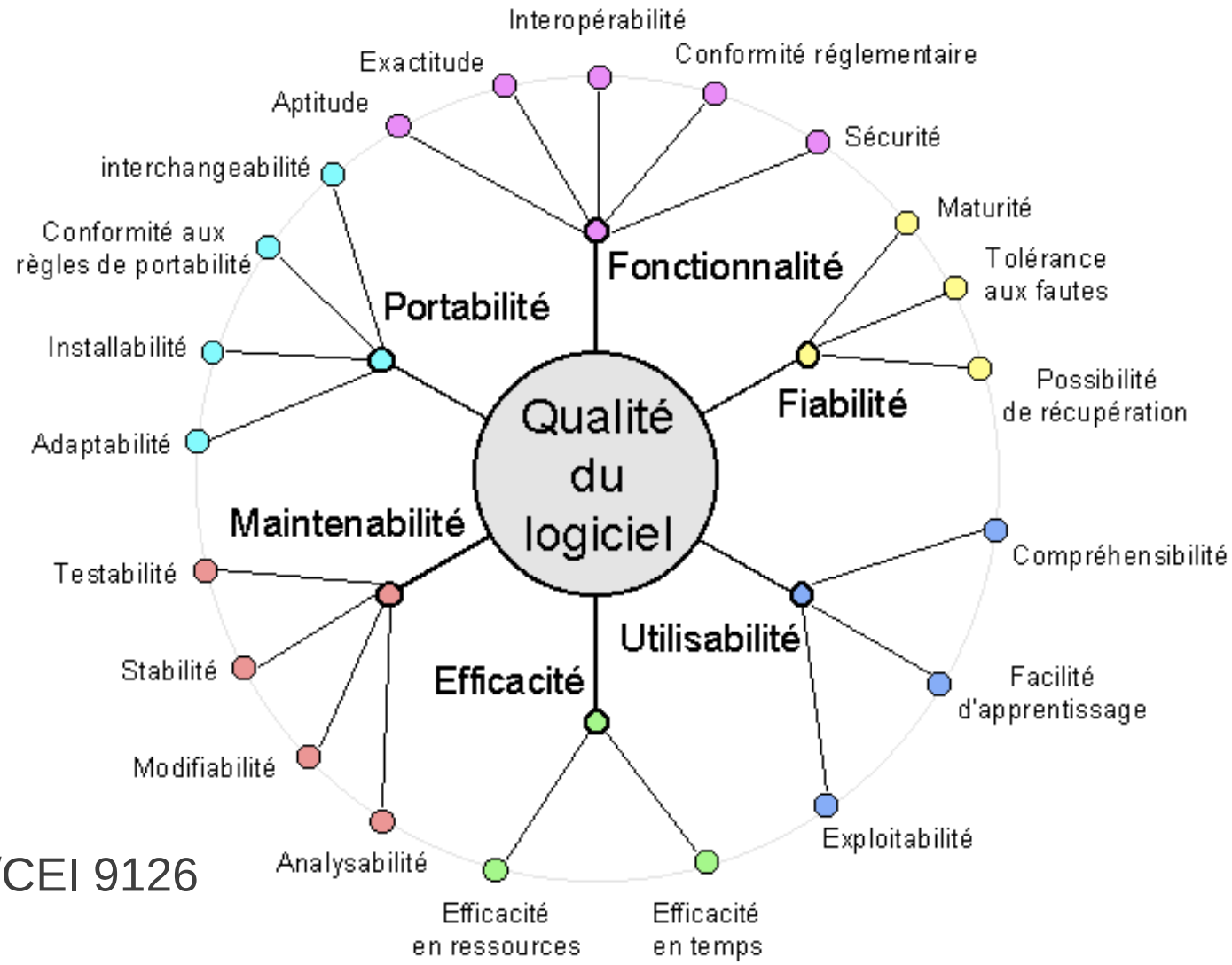
Introduction

Les acteurs du projet

- ALF équipe de recherche INRIA:
  - Compilation Architecture Parallèles et Systèmes
  - 19 Chercheurs,
  - Contact: François Bodin, bodin@irisa.fr.
- CAPS Entreprise, start-up émanation de CAPS INRIA:
  - Développement d'outillage spécifiques pour l'optimisation de code.
  - Année de création 2002, 20 collaborateurs.
  - Contact: Laurent Bertaux, laurent.bertaux@caps-entreprise.com., Directeur Général.
- ORANGE OBS IT&Labs (SILICOMP AQL):
  - Société de services en ingénierie logicielle : SI, industriel et embarqué.
  - 1984, filiale de France Télécom depuis 2007, 900 collaborateurs, CA: 70 M€.
  - Contact: Frank Rousée, frank.rousee@aql.fr, Directeur de Projet.

Introduction

Motivations: **Qualité logicielle polymorphe**



Source : ISO/CEI 9126

Solutions d'audit de code java existantes

- Multitude d'outils disponibles (Checkstyle, PMD, Findbugs, Coverity, Klocwork…).
  - → Configurations, modes d'utilisations
- Peu ou pas de flexibilité. (inadaptés a certains process de dév.)
- Faible couverture fonctionnelle
- Règles d'analyse superficielles, manquent de généricité (PMD, Checkstyle..)
- Analyse de l'application finale le plus souvent limitée au langage java → faux positifs
- En sécurité quelques solutions (Fortify, Insure, Klocwork) efficaces mais:
  - Chères
  - Pointues, difficiles d'emploi, (expert)
  - Corrections parfois complexes (encore besoin expert)

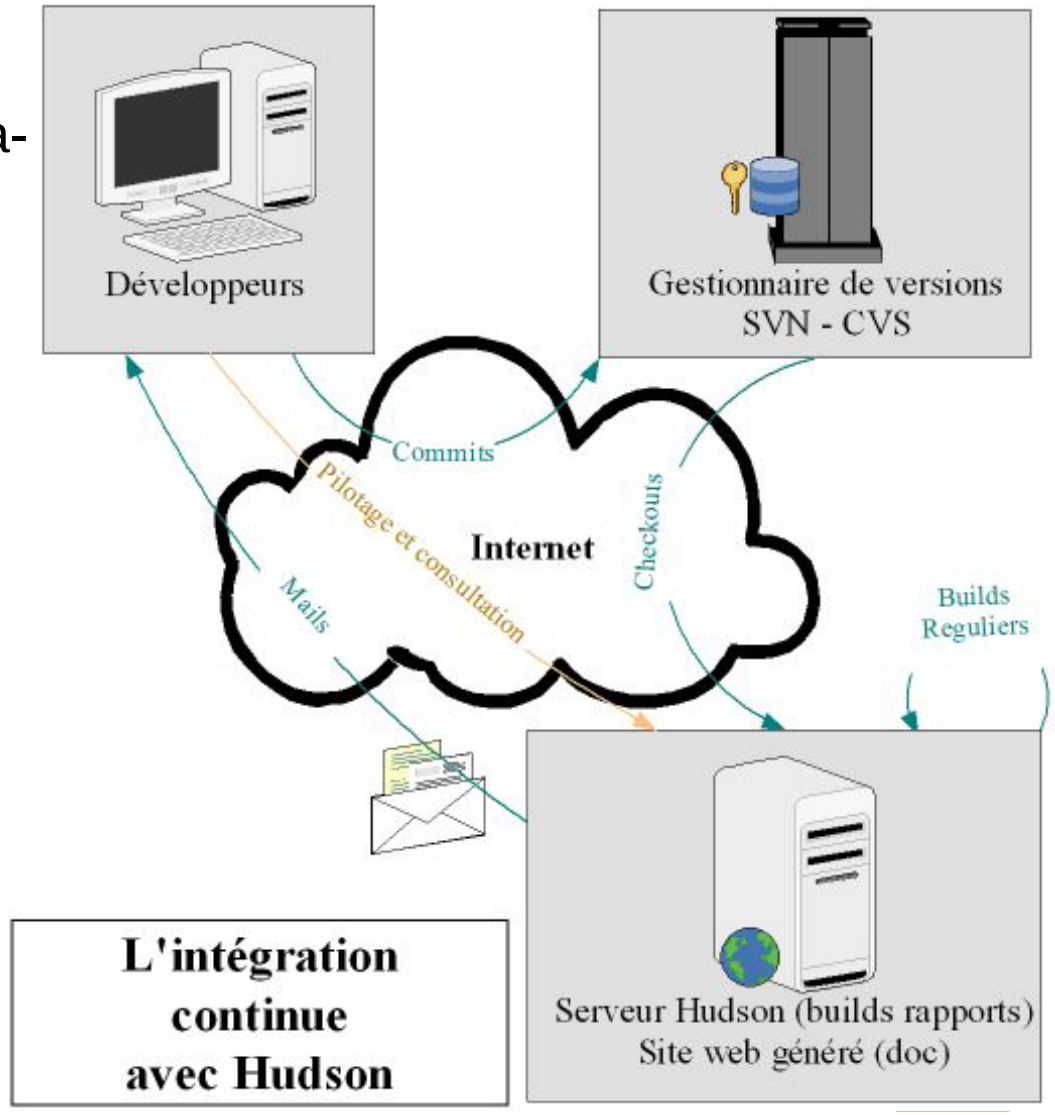
Le refactoring

- **Transformation de code source pour:**
  - Mettre en forme, respecter des normes
  - Améliorer les performances
  - Restructurer, améliorer la qualité (métriques,...)
  - Corriger les bugs
  - Supprimer les failles de sécurité

→ Sans changer de comportement de l'application
→ Sans ajouter de nouvelles fonctionnalités

Solutions de refactoring java existantes

- Refactoring très limité vis-à-vis des résultats de l'audit
- Pas d'automatisation → correction manuelle
- Pas de solution adaptée à l'intégration continue



Analyses: Différentes stratégies

- **Type d'analyse**
  - **Analyse dynamique**
    - Instrumentation / détection à l'exécution
    - ☹ Couverture partielle du code
    - ☹ Exposition aux attaques imprévues
  - **Analyse statique**
    - ☺ Parcours exhaustif du code
    - 😐 Certains comportements dynamiques imprévisibles
    - ☹ Résultats parfois peu précis
- **Langage cible**
  - **Sous ensemble de java**
    - Temps réel,
    - Bytecode uniquement
    - Embarqué..
  - **Ancienne version <5.0 (généricité! Annotations)**

Le projet Serenitec

- **Quoi?**
  - Serenitec est un outil d'analyse statique et de refactoring de code Java.
  - Permet d'accéder aux possibilités du refactoring précitées de manière automatique.
- **Comment?**
  - Prend en compte des applications Java/JEE dans leur intégralité
  - Langage 5.O complet + bytecode
- **Sous quelle forme?**
  - Outil générique adaptatif
  - Composé de règles et de référentiels configurables

Une approche holistique nécessaire

De l'information capitale n'est pas dans le code java!

```
<table>
   <thead>
     <tr>
        <g:sortableColumn property="name" title="Name" />
        <g:sortableColumn property="imagethumburl" title="Photo" />
        <g:sortableColumn property="price" title="Price" />
     </tr>
   </thead>
   <tbody>
     <g:each in="${itemList}" status="i" var="item">
        <tr class="${(i % 2) == 0 ? 'odd' : 'even'}">
           <td>
             <g:link action="show" id="${item.id}">
                ${item.name?.encodeAsHTML()}</g:link>
```

```
// Print Date

def mydate = new java.util.Date()

println mydate


//Iterate through a map


def numbersMAP = ['1':'ONE', '2':'TWO']

for (entry in numbersMAP) {

    println "${entry.key} = ${entry.value}"

}
```

- Jsp
- Code groovy (GSP - Grails)
- Injection de dépendance
- Fichiers de règles de filtrage struts

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xsi:schemaLocation="http://www.springframework.org/schema/beans
           http://www.springframework.org/schema/beans/spring-beans-2.5.xsd">

  <bean id="logger" class="fr.ssa.log.imp.BeanFileLogger"
      init-method="init" destroy-method="close">
      <property name="fileName">
          <value>/tmp/myapp.log</value>
      </property>
  </bean>

  <bean id="business" class="fr.ssa.bus.imp.SimpleBusiness"
      init-method="init" destroy-method="close">
      <property name="logger" ref="logger" />
  </bean>
</beans>
```

Approche Serenitec

Toute information liée à l'application est accessible et analysable

- **Prend en compte des applications Java/JEE dans leur intégralité**
  - Java Complet 5.0
  - Adaptation à l'application analysée :
    - Traite les diverses classes d'applications existantes. (web , intranet, frameworks …)
  - Analyse du code source Java et fichiers non-java:
    - Analyse du bytecode sous forme de .class ou compressé en jar
    - Fichiers jsp, jsf, fichiers de configuration xml, d'injection de dépendances, fichiers de politique de sécurité, fichiers de ressources textuelles, listing des ressources (sons, images…)
    - Représentation du contexte d'exécution: droits, flux sensibles, dmz…
- **Ensemble des informations réunies dans un unique framework d'analyse et de refactoring**

Conséquences

- **Marge de manœuvre étendue**
  - Rends possible n'importe quelle analyse et transformation associée

    →Exemple de la validation struts

```
<form name="logonForm">
<field property="username"
    depends="required">
    <arg key="logonForm.username"/>
   </field>
  <field property="password"
     depends="required,mask">
     <arg key="logonForm.password"/>
     <var>
       <var-name>mask</var-name>
       <var-value>^[0-9a-zA-Z]*$</var-value>
     </var>
```

  - Plus d'informations pour guider l' analyse

    →Minimise risque de faux positifs

  - Donne accès à la découverte de bugs complexes car répartis dans plusieurs sources d'informations (ex:validation)

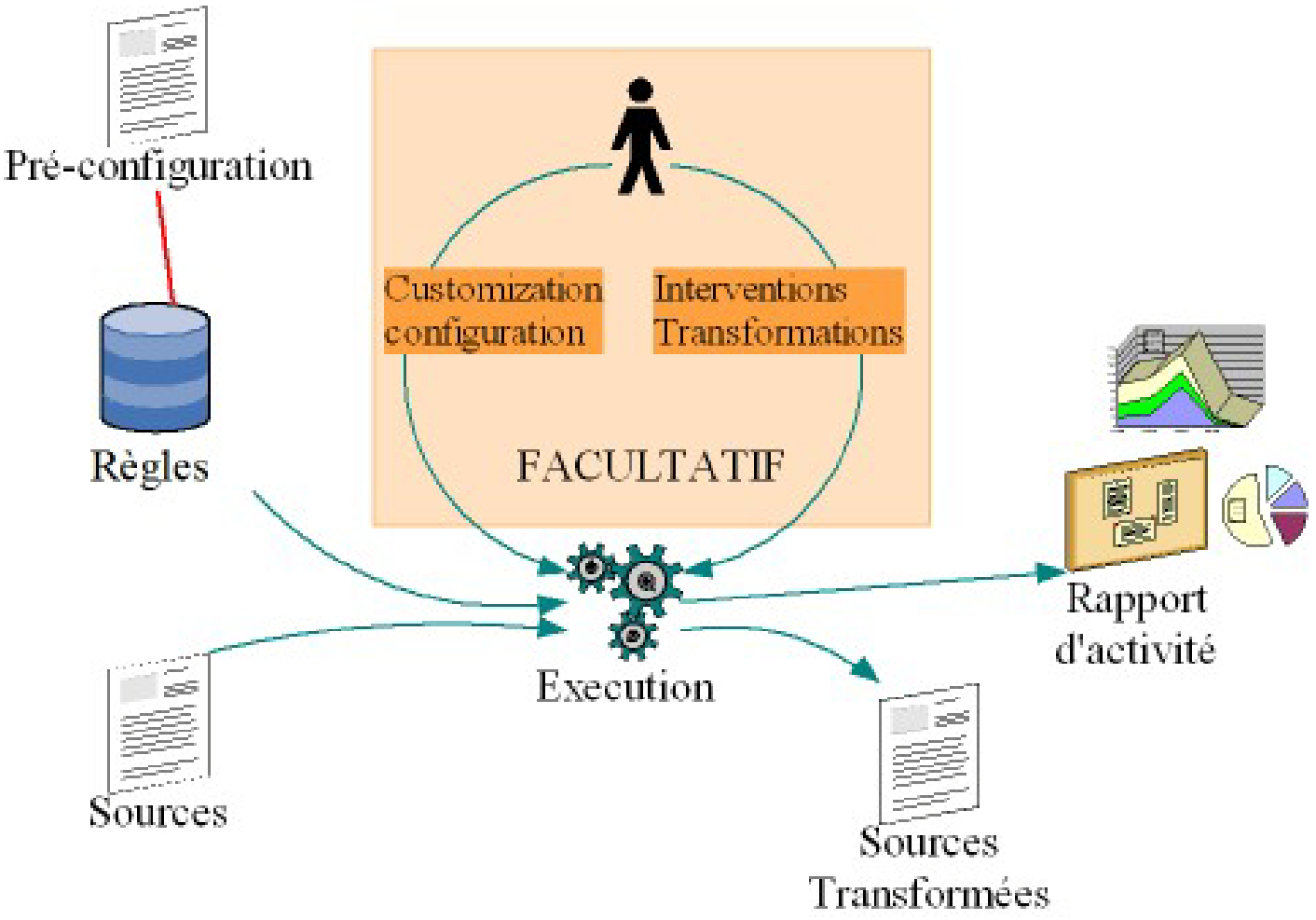
```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xsi:schemaLocation="http://www.springframework.org/sch
           http://www.springframework.org/schema/beans/spring

  <bean id="logger" class="fr.ssa.log.imp.BeanFileLogger"
      init-method="init" destroy-method="close">
      <property name="fileName">
          <value>/tmp/myapp.log</value>
      </property>
  </bean>

  <bean id="business" class="fr.ssa.bus.imp.SimpleBusiness"
      init-method="init" destroy-method="close">
      <property name="logger" ref="logger" />
  </bean>
</beans>
```

Outil générique adaptatif

Schéma simple de fonctionnement



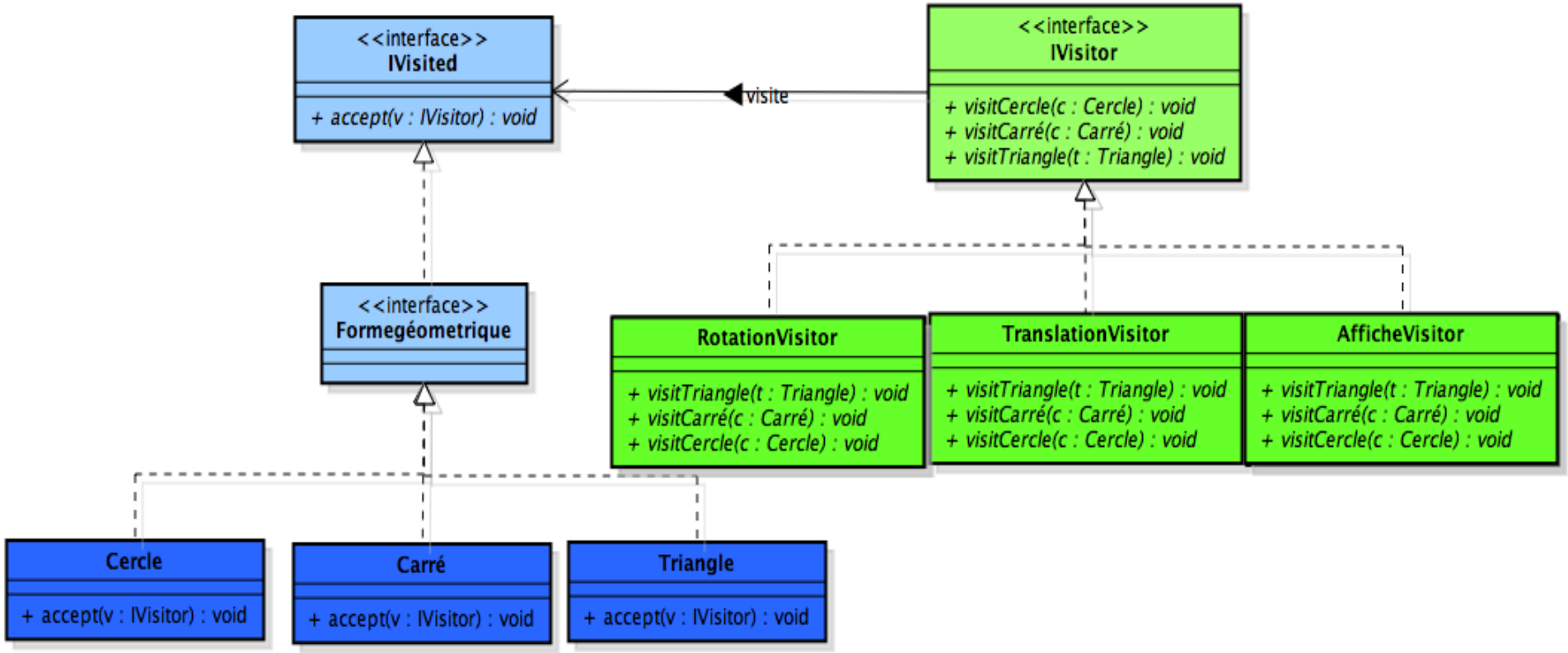
Application - Exemples

- Implémentation d'un détecteur / validateur de design pattern

- Correcteur de faille d'injection en cours de développement

Application - Exemples

Description UML d'un pattern visiteur



Détection / validation de design pattern

- Difficulté d'utilisation des patterns → mauvais usages fréquents
  - Méconnaissance du pattern
  - Difficulté de le reconnaitre dans le code
- Pratiques anti-pattern au cours du développement nuisibles
- Actuellement, aucun outil ne permet de détecter un design pattern visiteur dans un code

Correction de failles de sécurité

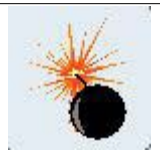
Injections de données (Xss, sql, splitting…) Illustration dans une servlet

```
BufferedInputStream in…;
String a=in.readLine();
String b=‘’saved value:’’+a;
…
..
HttpServletResponse res….;
res.getWriter().println(’’<li><a href=www. example.com/... >’’+b+’’</a>
</li>’’);
```

Donnée entrante

Méthode potentiellement dangereuse

FAILLE !



Correction de failles de sécurité

Injections de données (Xss, sql, splitting…)

- Identification automatique des éléments à contrôler(principe de teinture de données)
  - Identification des ‘sources’ du programme
    - Saisies manuelles
    - Flux entrants (réseaux, disque…)
  - Identification des faiblesses potentielles
    - Appels systèmes
    - Flux sortants (sérialisation, écriture…)
  - Identification des failles réelles
    - Un élément ‘entrant’ est mis en relation avec une ‘sortie’
    - Cet élément peut prendre des valeur connues comme étant dangereuses → Mise en œuvre de contrats et de filtres par AOP

Conclusion

- Ambition industrielle dans le domaine du génie logiciel à court terme - 2010
  - Correction de règles classique automatiquement
  - Autonomie→Utilisation qqsoit le contexte de développement
  - Parallélisation pour le traitement de gros projets
- Ambition dans le domaine de la sécurité à moyen terme:
  - Une grand partie du top ten d'OWASP (7) pourra être traitée par SERENITEC
  - Système capable de déduire la vulnérabilité potentielle d'une application et d'y remédier dans la plus part des cas
  - Potentiel supérieur aux outils existants (Fortify, Klocwork, Parasoft, ...)

Top 10 OWASP 2010(RFC):

- **A1 –Injection**
- **A2 –Cross Site Scripting (XSS)**
- **A3 –Broken Authentication and Session Management**
- **A4 –Insecure Direct Object References**
- **A5 –Cross Site Request Forgery (CSRF)**
- **A6 –Security Misconfiguration(NEW)**
- **A7 –Failure to Restrict URL Access**
- **A8 –UnvalidatedRedirects and Forwards (NEW)**
- **A9 –Insecure Cryptographic Storage**
- **A10 -Insufficient Transport Layer Protection**