Composition flexible et efficace de transformations de programmes

par

Romain Lenglet

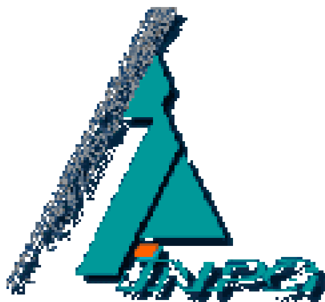
jeudi 25 novembre 2004

Thèse effectuée sous la direction de

Thierry Coupaye (France Telecom)

et

Daniel Hagimont (INRIA)

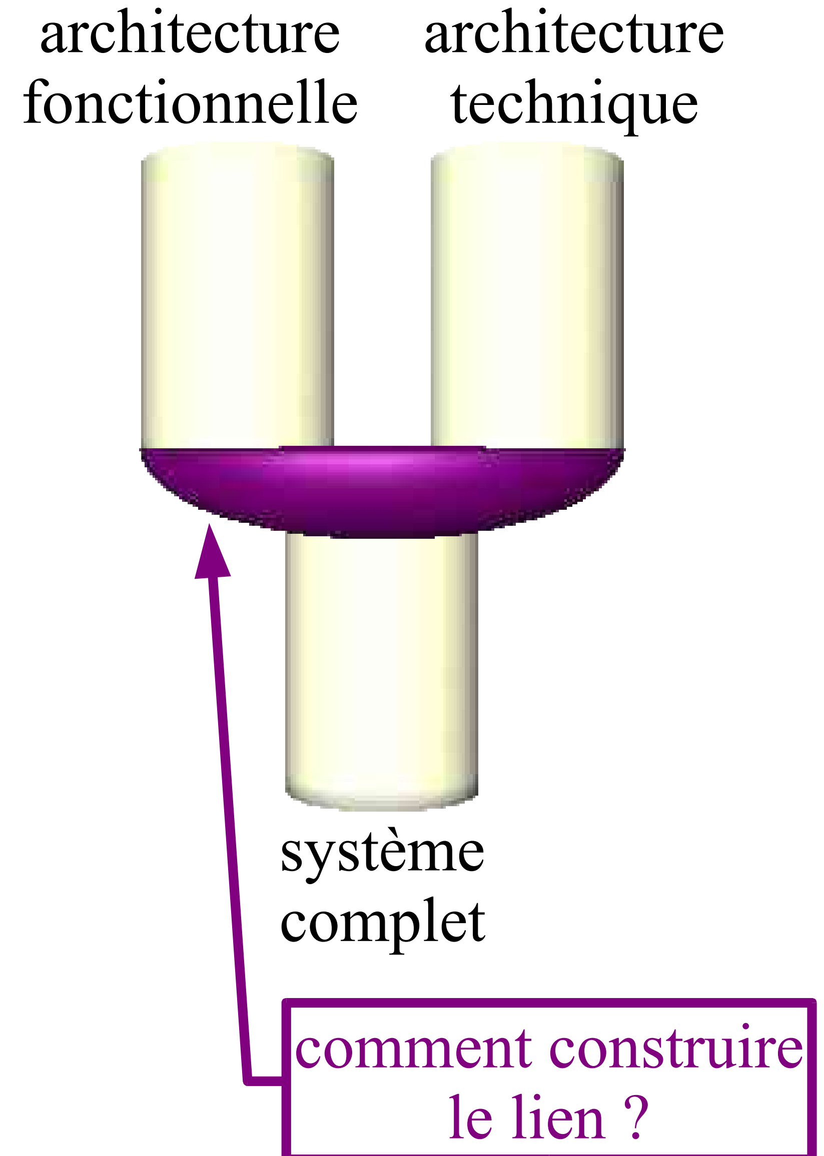


Plan

- Problématique
- Le système de transformation Jabyce
  - Mécanisme de représentation des programmes transformés
  - Modèle d'implantation de transformateurs
  - Comparaison qualitative et quantitative aux autres systèmes
- Sûreté de fonctionnement des systèmes de transformation
- Validations, conclusion et perspectives

Séparation des préoccupations

- Principe de Séparation des Préoccupations [Hursch95]
  - Spécification séparée
  - Lien automatique
- Séparation classique d'un logiciel en deux parties :
  - Architecture *fonctionnelle* : “métier”
  - Architecture *technique* = ensemble de services techniques : reprise après défaillance, duplication d'objets, gestion de transactions, persistance, etc.



Approches pour le lien fonctionnel / technique

- Conteneurs et modèles de composants : EJB, CCM, .Net
- Programmation par aspects (AOP) : AspectJ, JAC
- Réflexion (MOP)

s'appuient sur ou sont des formes de

- Transformation de programmes
  - (programmes manipulant des programmes)

- Domaine très vaste [Visser]:
  - Simplification
  - Mélange d'aspects
  - Optimisation
  - Remaniement (refactoring)
  - (Dé)compilation
  - Ingénierie inverse
  - Génération de documentation
- Approche retenue : transformation de prog.
  - Approche la plus générale
  - La plus efficace

Conception d'un système de transformation de programmes

- Problématique : conception d'un système de transformation

majorité des travaux actuels

contexte de nos travaux

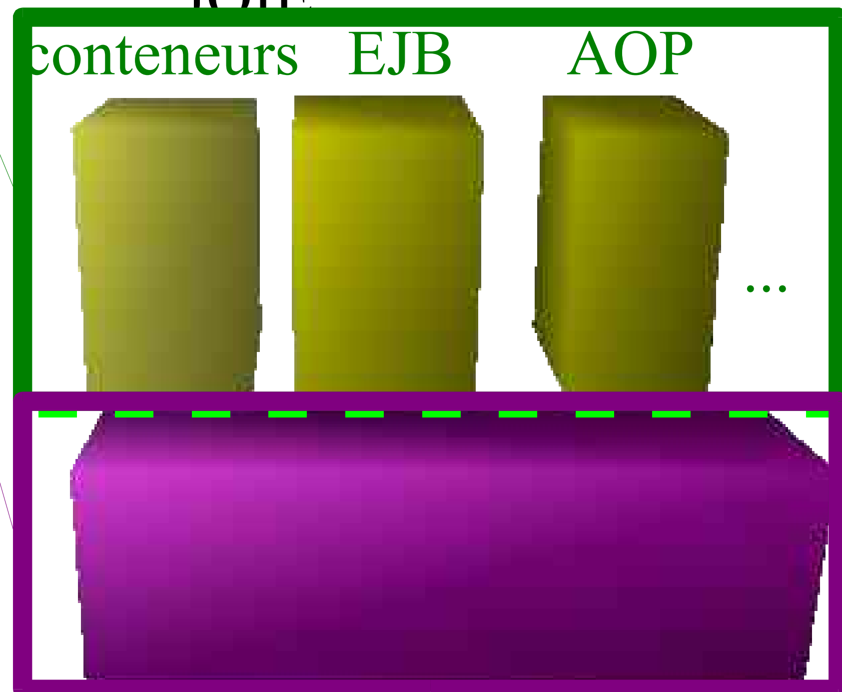
systèmes spécialisés :
langages ou APIs spécifiques

objectifs / problèmes : simplicité, vérification, sémantique

objectifs / problèmes : généralité, efficacité

système de transformation généraliste

- Généralement conception à deux niveaux (ou plus) :
  - ASM / Cglib / Nanning
  - JMangler / JMunger
  - Javassist
  - JOIE



Plan

- Problématique
- Le système de transformation Jabyce
  - Mécanisme de représentation des programmes transformés
  - Modèle d'implantation de transformateurs
  - Comparaison qualitative et quantitative aux autres systèmes
- Sûreté de fonctionnement des systèmes de transformation
- Validations, conclusion et perspectives

Propositions pour la conception de Jabyce (1/2)

- 3 objectifs :
  - **Généralité** : toute transformation est implantable
  - **Réutilisabilité** et **composabilité** de transformateurs
  - **Composition efficace** et flexible de transformateurs
- Ces 3 objectifs ne sont pas atteints simultanément par les systèmes existants

	Généralité	Réutilisabilité	Composition efficace
JOIE			
JMangler			
Javassist			
ASM			
Jabyce			
Serp			
BCEL			

Propositions pour la conception de Jabyce (2/2)

- Transformation au chargement de code :
  - Transformation de code compilé Java (bytecode)
  - Technique classique : Javassist, JOIE, etc.
  - **Problèmes de performances : l'objectif de composition efficace est important**
- 2 caractéristiques originales :
  - Les programmes transformés sont représentés par des **séquences d'interactions** (au lieu de graphes d'objets)
  - Les transformateurs sont des **composants logiciels** (Fractal)

Plan

- Problématique
- Le système de transformation Jabyce
  - Mécanisme de représentation des programmes transformés
  - Modèle d'implantation de transformateurs
  - Comparaison qualitative et quantitative aux autres systèmes
- Sûreté de fonctionnement des systèmes de transformation
- Validations, conclusion et perspectives

Représentation des programmes par interactions (1/3)

- Exemple de classe Java :

```
public class A {

    private String chaine =
        "hello world";

    public void uneMethode() {
        System.out.println(
            chaine);
    }
}
```

Comment représenter ces éléments de programmes ?

Cf documents XML :
- DOM vs. SAX
- graphes d'objets vs. appels

- Code compilé (bytecode) à transformer :

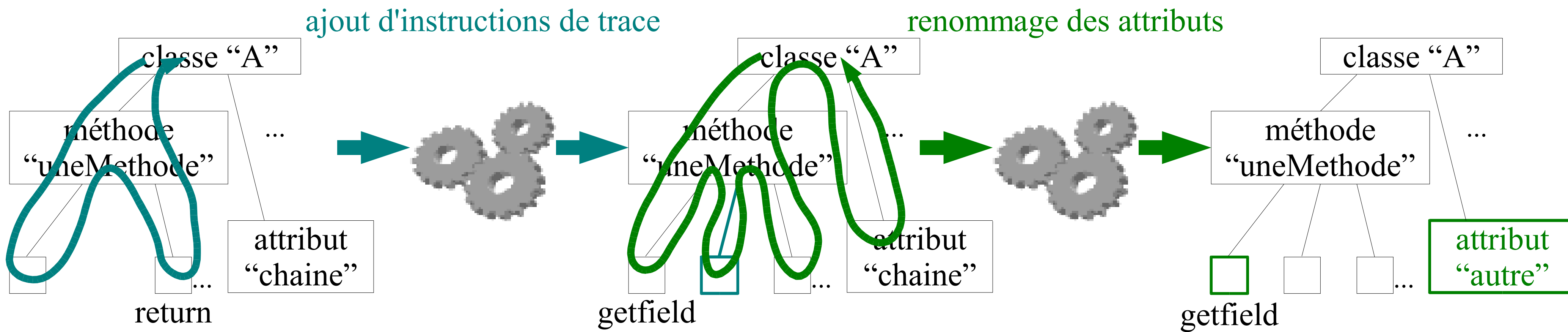
```
public class A extends
   java.lang.Object{
..
public void uneMethode();
  Code:
   0:    getstatic         #4;
  //Field
  java/lang/System.out:Ljava/io/PrintStream;
   3:    aload_0
   4:    getfield          #3;
  //Field chaine:Ljava/lang/String;
   7:    invokevirtual     #5;
  //Method java/io/PrintStream.println:
  (Ljava/lang/String;)V
   10:   return

}
```

Représentation des programmes par interactions (2/3)

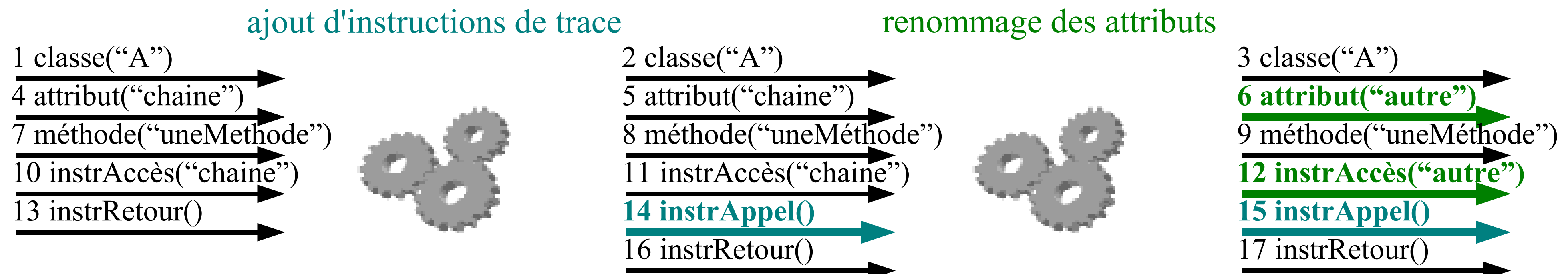
- Représentations classiques : par graphes d'objets
  - 1 élément = 1 objet

- Problèmes :
  - Grand nombre d'objets créés
  - Chaque transformateur balaye chaque programme : **composition non efficace**



Représentation des programmes par interactions (3/3)

- Notre proposition : représentation par interactions :
  - 1 élément = 1 appel de méthode
  - Représentation duale de celle en graphe d'objets
  - Mise en oeuvre dans ASM et Jabyce
- Meilleures performances :
  - Peu d'objets créés
  - 1 seul balayage de chaque programme, quelque soit le nombre de transformateurs

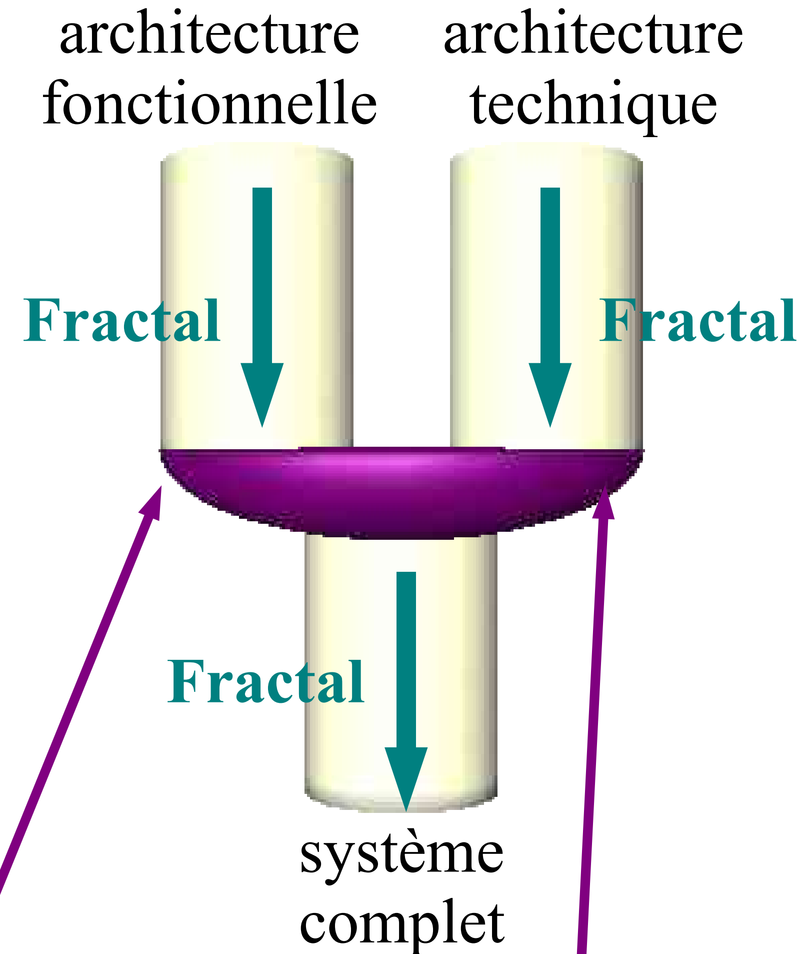


Plan

- Problématique
- Le système de transformation Jabyce
  - Mécanisme de représentation des programmes transformés
  - Modèle d'implantation de transformateurs
  - Comparaison qualitative et quantitative aux autres systèmes
- Sûreté de fonctionnement des systèmes de transformation
- Validations, conclusion et perspectives

Conception par composants

- Approche générale FT / INRIA :
  - La configuration est la préoccupation fondamentale
  - Modèles de composants : **séparation de la configuration**, de l'implantation
  - Un seul modèle de composant pour structurer tout le système de manière homogène : Fractal

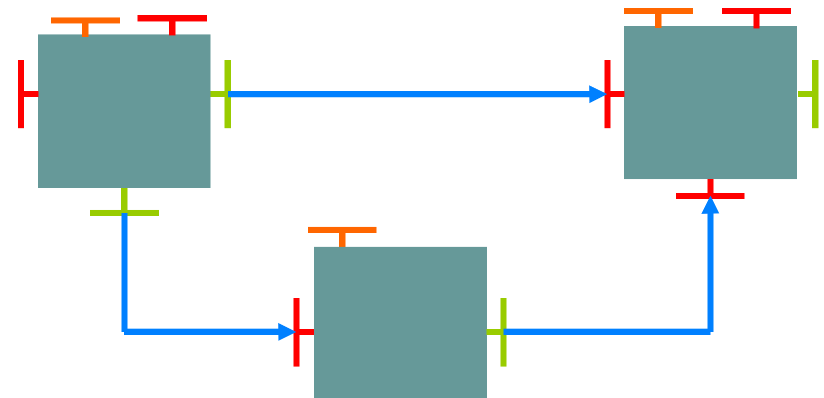


Contexte de cette thèse :
lien pour les autres préoccupations

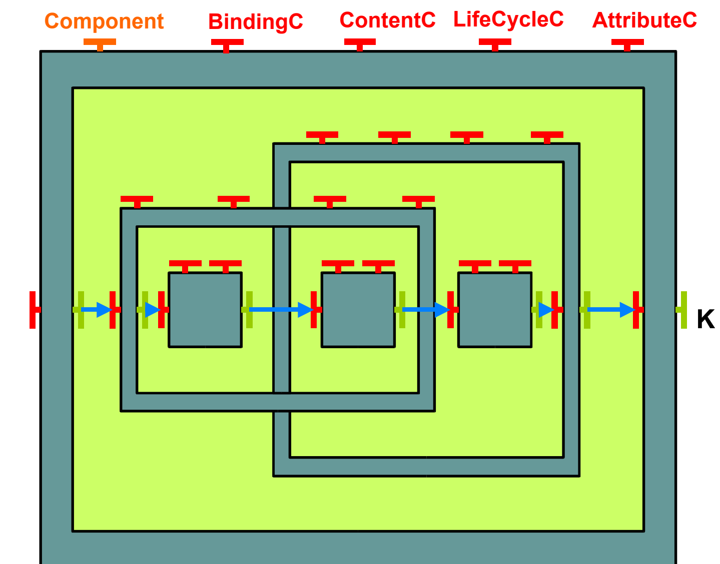
Conception de Jabyce
avec Fractal

Le modèle de composants Fractal

- Modèle :
  - Composant
    - = membrane [contenu]
    - Entité présente à l’exécution
  - Interface
    - Seuls points d’accès à un composant
    - Permet d’émettre et recevoir des invocations d’opérations
  - Liaison : canal de comm. primitif local

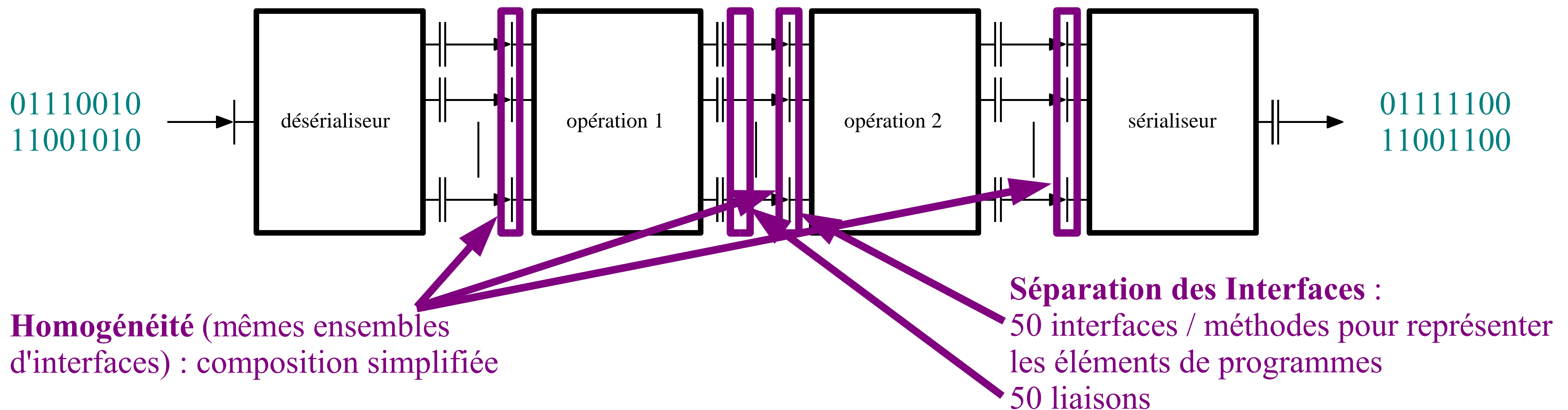


- Propriétés intéressantes ici :
  - Récursivité (auto-similarité)
    - composites
    - primitifs : encapsulation d'objets Java
  - (Re)configuration dynamique



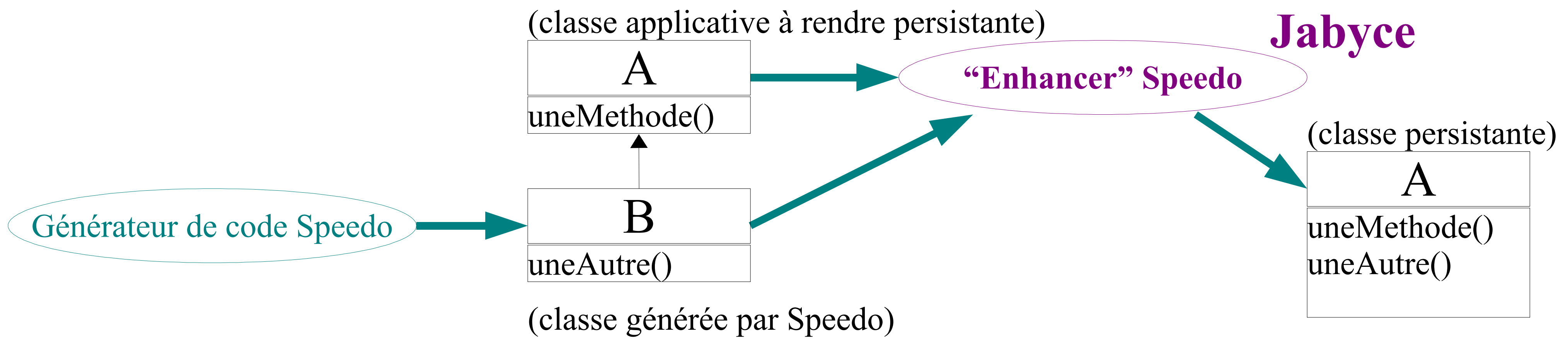
Utilisation de Fractal dans Jabyce

- Chaîne de composants de transformations :
  - **désérialiseur** = analyse de code compilé
  - **opérations** = transformation de code (unité de traitement)
  - **sérialiseur** = génération de code compilé
- Stratégie de transformation = assemblage de composants
  - Outils standards de Fractal : ADL, outils de visualisation
  - Stratégies arbitraires



Cas d'application : mélange de classes (1/3)

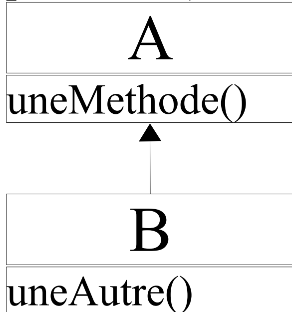
- Contexte : Speedo
  - Implantation de JDO (Java Data Objects)
  - Service de persistance transparente d'objets Java
  - Développé par France Telecom
- Construction du lien fonctionnel / Speedo avec plusieurs transformations de programmes :
  - Génération de code de sous-classes des classes persistantes
  - Extraction de l'état
  - **Mélange des sous-classes**



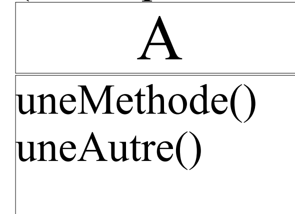
Cas d'application : mélange de classes (2/3)

- “Copie” d'une classe dans sa super-classe
- Transformateur efficace
  - Représentation par interactions
  - Chaque classe n'est balayée qu'une seule fois

(classe applicative à rendre persistante)



(classe persistante)



(classe générée par Speedo)

A

A'

B

transformation des classes de base et déclenchement du mélange

déserialisation et balayage des sous-classes

transformation optionnelle des sous-classes

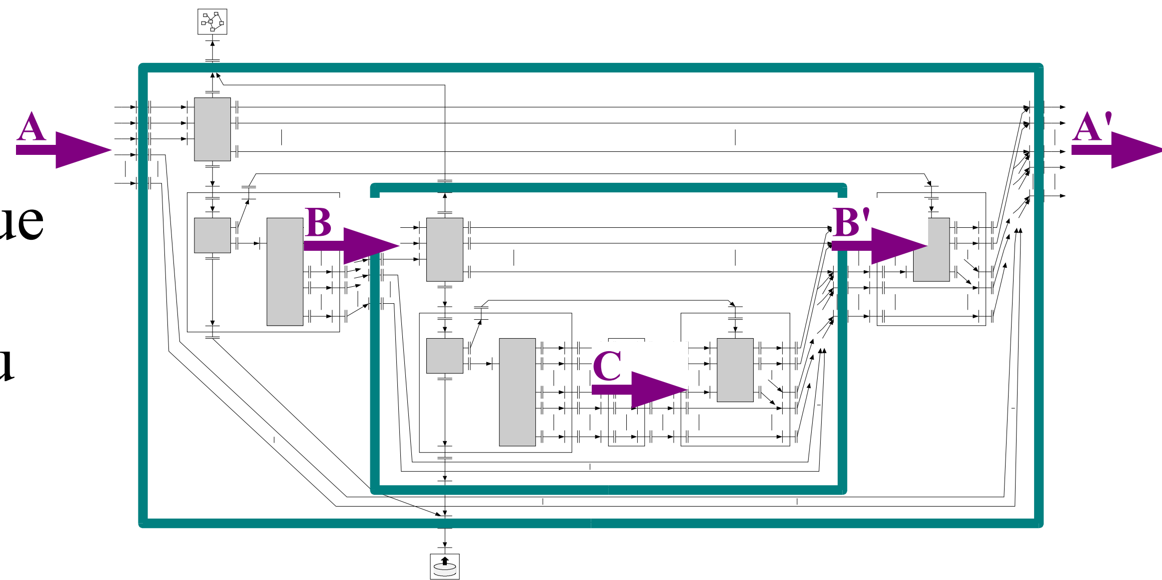
transformation de mélange des sous-classes

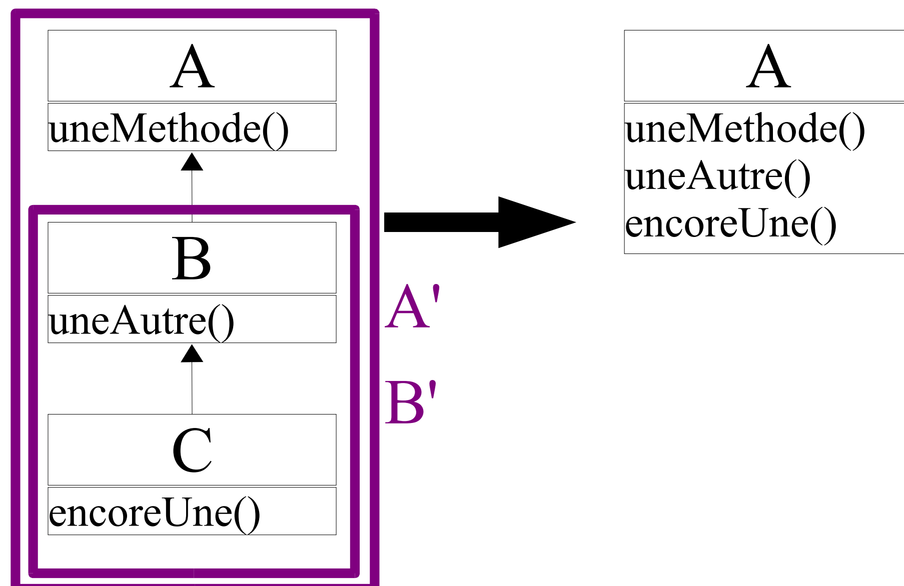
Difficultés :

- Renommage des sous-classes
- Mélange des constructeurs

Cas d'application : mélange de classes (3/3)

- Mélange à N niveaux
- Architecture “fractale”
  - Utilisation de la composition hiérarchique dans Fractal
  - Aucune modification du code

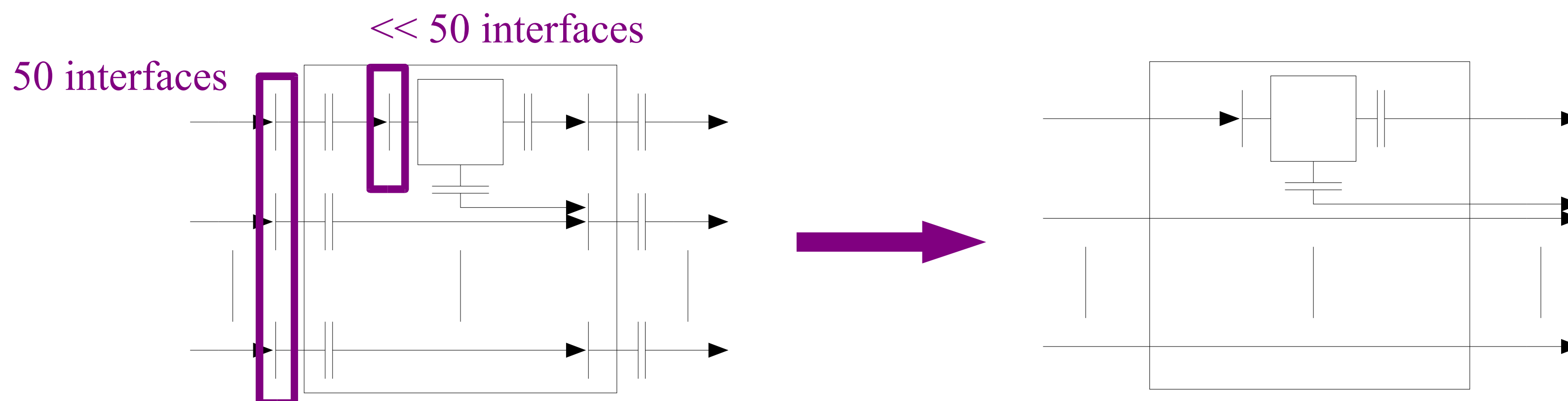




Utilisation possible des capacités de **configuration dynamique** de Fractal : assemblage récursif dynamique d'opérations de mélange en fonction de la profondeur de l'arbre d'héritage

Optimisation des liaisons Fractal

- Dans Jabyce :
  - Les opérations sont généralement des **composites "creux"**
  - Séparation des interfaces
  - Liaisons fortement optimisables
- Dans Julia (Fractal / Java) :
  - **Optimisation automatique et transparente des liaisons** à travers les membranes de composants composites
  - La composition hiérarchique a un coût négligeable



Plan

- Problématique
- Le système de transformation Jabyce
  - Mécanisme de représentation des programmes transformés
  - Modèle d'implantation de transformateurs
  - Comparaison qualitative et quantitative aux autres systèmes
- Sûreté de fonctionnement des systèmes de transformation
- Validations, conclusion et perspectives

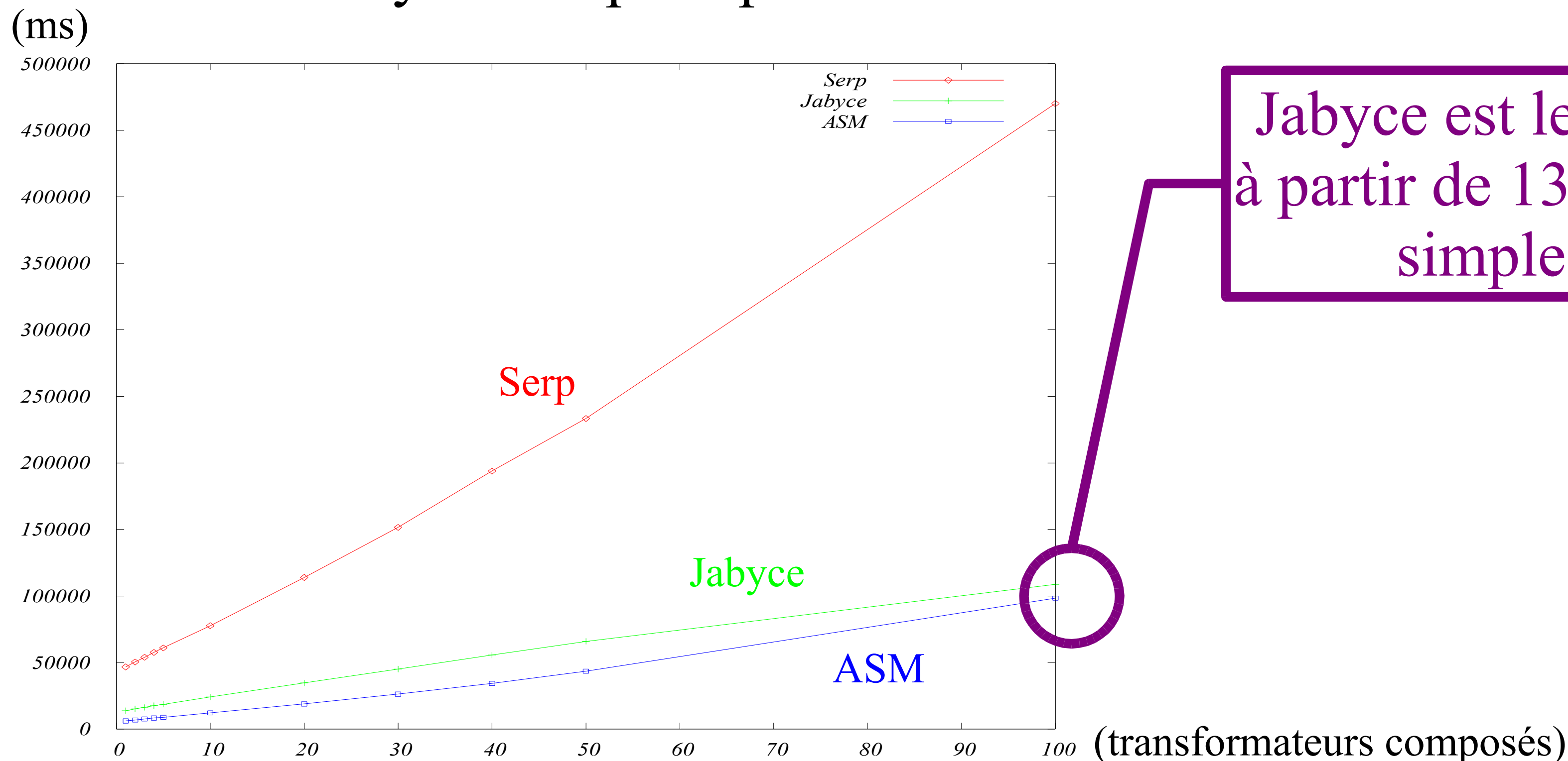
Comparaison avec les systèmes (1/2)

Javassist, JOIE, JMangler, ASM, Serp, BCEL

- Extension de la catégorisation classique [E. Visser] :
  - **portée** (généralité) : JOIE et Javassist sont limités
  - **mécanisme de représentation** : Jabyce & ASM offrent une représentation par interactions, vs. les autres systèmes
  - **modèle conceptuel** : Jabyce offre l'un des modèles les plus abstraits (p.ex. : abstraction des adresses de saut)
  - **représentation de fragments** : seul Javassist offre un langage pour faciliter la spécification de fragments de programmes
  - **paradigme de transformation** : Jabyce utilise Fractal, ce qui facilite la conception de transformateurs complexes
  - **stratégie** : Jabyce est le systèmes le plus flexible pour la composition des transformateurs
  - **vérification des transformations** : Jabyce offre les mécanismes de localisation et de diagnostic les plus précis

Comparaison avec les systèmes (2/2)
ASM et Serp

- Performances :
  - Transformateurs simples : trace avant instructions “return”
  - Jabyce : courbe linéaire
  - Autres systèmes : courbes paraboliques
  - Jabyce compose plus efficacement les transformateurs



Jabyce est le plus performant à partir de 132 transformateurs simples composés

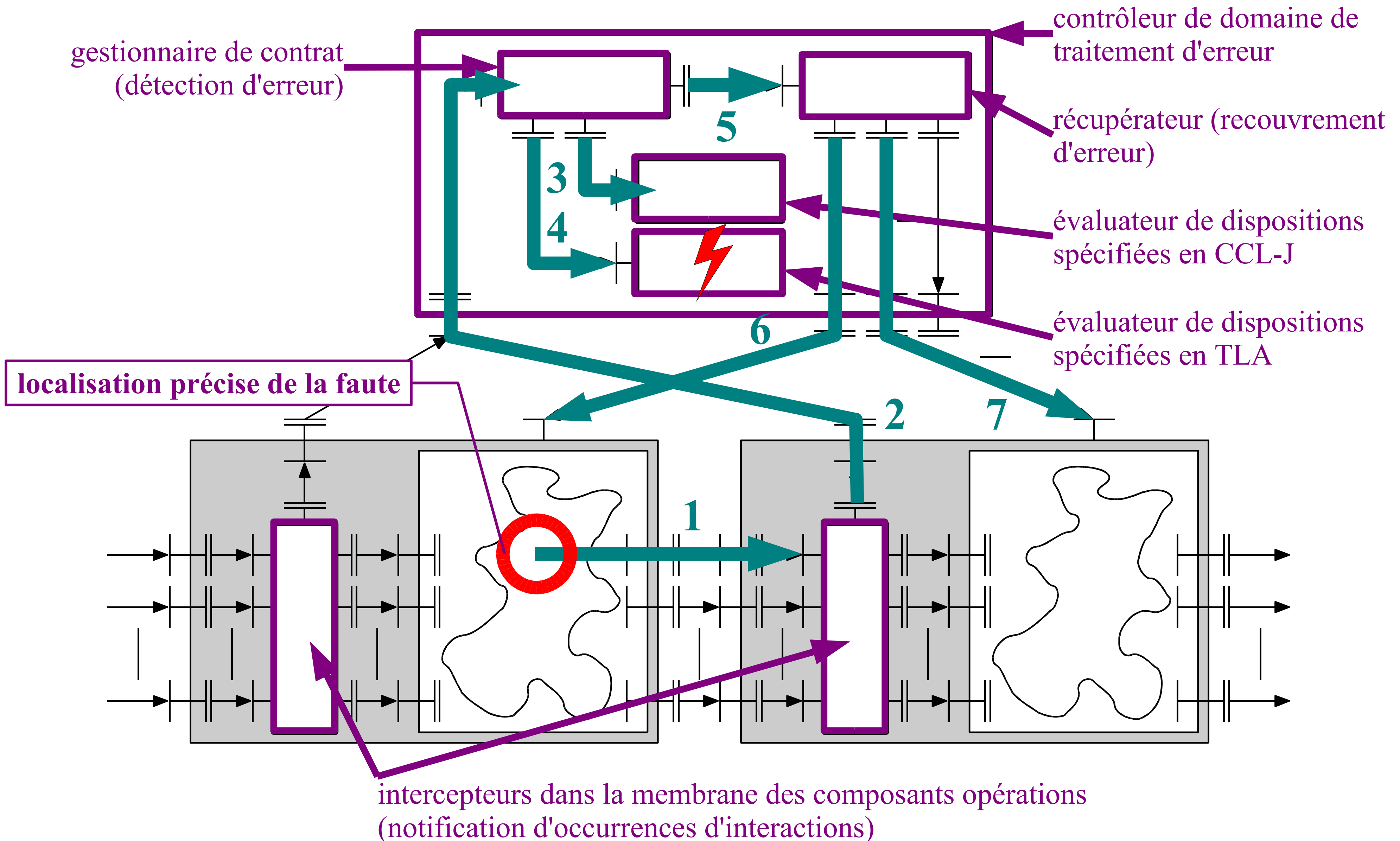
Plan

- Problématique
- Le système de transformation Jabyce
  - Mécanisme de représentation des programmes transformés
  - Modèle d'implantation de transformateurs
  - Comparaison qualitative et quantitative aux autres systèmes
- Sûreté de fonctionnement des systèmes de transformation
- Validations, conclusion et perspectives

Vérification des transformations (1/2)

- Sûreté de fonctionnement :
  - Seuls des **programmes corrects** doivent être représentés (cf. les vérifications effectuées par la JVM)
  - Traitement d'erreur complet : détection, diagnostic, **recouvrement**
  - Problématique originale
- Proposition :
  - Architecture Fractal de traitement d'erreur : gestion de contrats pour Fractal (ConFract, TLO) + recouvrement d'erreur
  - Collaboration pour ConFract avec le laboratoire I3S, U. Nice (P. Collet, R. Rousseau)
  - Combinaison de **plusieurs formalismes** pour faciliter la spécification de contrats : CCL-J (assertions), TLA (logique temporelle)
  - Représentation par interactions : localisation précise des fautes

Vérification des transformations (2/2)



Plan

- Problématique
- Le système de transformation Jabyce
  - Mécanisme de représentation des programmes transformés
  - Modèle d'implantation de transformateurs
  - Comparaison qualitative et quantitative aux autres systèmes
- Sûreté de fonctionnement des systèmes de transformation
- Validations, conclusion et perspectives

Validations expérimentales

- Speedo : persistance d'objets Java
  - Mélange des classes : montre l'intérêt de la composition hiérarchique
  - Réification de l'état des objets
  - Ces deux transformations montrent les difficultés liées aux détails de la spéc. de la JVM : motivent le **besoin de réutilisabilité** des transformateurs
- Jivaro : JVM avec compilation Java vers C, pour l'embarqué
  - Utilisation de Jabyce pour la conception du compilateur modulaire Java vers C
  - Compilation statique : nombreuses optimisations possibles
  - Compilation “dans le réseau” à la volée : besoin de performances

Conclusion

- Problématique :
  - Contexte : construction de liens fonctionnel / technique
  - Conception d'un système de transformation avec 3 objectifs : Généralité, Réutilisabilité, Composition efficace
- Proposition : deux caractéristiques originales de Jabyce :
  - Représentation des programmes par interactions
  - Utilisation du modèle de composants Fractal : approche architecturale pour la composition
- Comparaison qualitive et quantitative aux autres systèmes :
  - Composition très efficace et flexible des transformateurs
- Etude des intersections avec d'autres domaines :
  - Architecture logicielle
  - Sûreté de fonctionnement
  - Recherche d'information

Perspectives

- Concevoir des “couches” au-dessus de Jabyce pour **simplifier la conception** de transformateurs :
  - Représentation par graphes d'objets en plus de la représentation par interactions
  - Langages de transformation (à la Stratego) avec de bonnes propriétés : réversibilité, transformation automatique des méta-classes
- Sûreté de fonctionnement :
  - Implantation de l'architecture proposée
  - Mesures de performances
  - Spécifications de contrats
  - **Transformations atomiques**, pour optimiser le recouvrement

Merci de votre attention !