Programmation Orientée Aspect

Partie I

Les bases de l'AOP

Qu’est-ce qu’une préoccupation ?

- But particulier, concept, domaine d'intérêt
- Un logiciel contient :
  - Des préoccupations métier
  - Des préoccupations de niveau système (ou techniques)

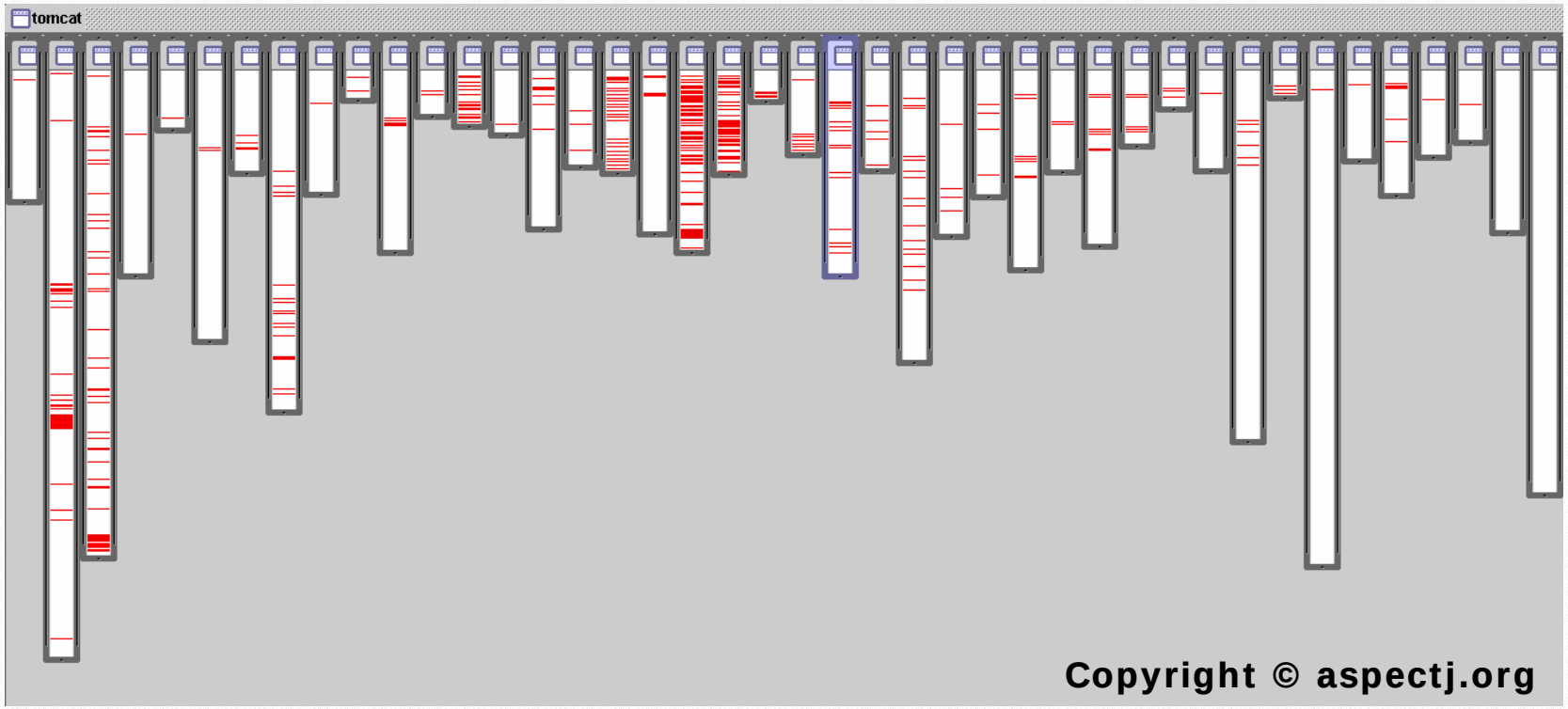
Exemple : paiement par carte de crédit

- Préoccupations métier :
  - Effectuer des paiements : débiter un montant d'un compte défini
- Préoccupations techniques :
  - Traçage
  - Intégrité de la transaction
  - Identification / authentification
  - Sécurité (confidentialité)
  - Performances
  - etc.

Dispersion / entrelacement des préoccupations

- Chaque objet métier gère ses propres contraintes techniques (à son niveau) : les préoccupations sont dispersées / entrelacées
- Les applications sont plus difficiles à :
  - Concevoir / coder
  - Comprendre
  - Maintenir
  - Faire évoluer...
- => BESOIN DE MODULARITE!

Exemple du logging



- Où est-il question de logging dans tomcat ?
  - Le rouge montre les lignes de code qui traitent du logging
  - Très mal modularisé

Les symptomes

- L'embrouillement du code
  - Les objets métiers intéragissent simultanément avec de nombreuses préoccupations techniques
  - Présence simultanée d'éléments de chaque préoccupation dans le code
- L'étalement du code
  - Les préoccupations entrelacées, s'étalent sur de nombreux modules. Leur implémentation s'étale elle aussi sur de nombreux objets
  - Exemple : un système utilisant une base de données
    - La préoccupation de performance peut affecter l'ensemble des objets qui accèdent à la base de données

Le dilemme de l'architecte

- Prévoir les besoins futurs est une tâche difficile
- Un architecte doit-il inclure un mécanisme de logging au sein d'un système qui n'en a pas besoin initialement ?
  - Où doivent se situer les point de logging ?
  - Quelles informations nécessitent authentification?
- Ou passer à côté de futures préoccupations entrelacées et devoir changer, ou même ré-implémenter, de nombreuses parties du système ?

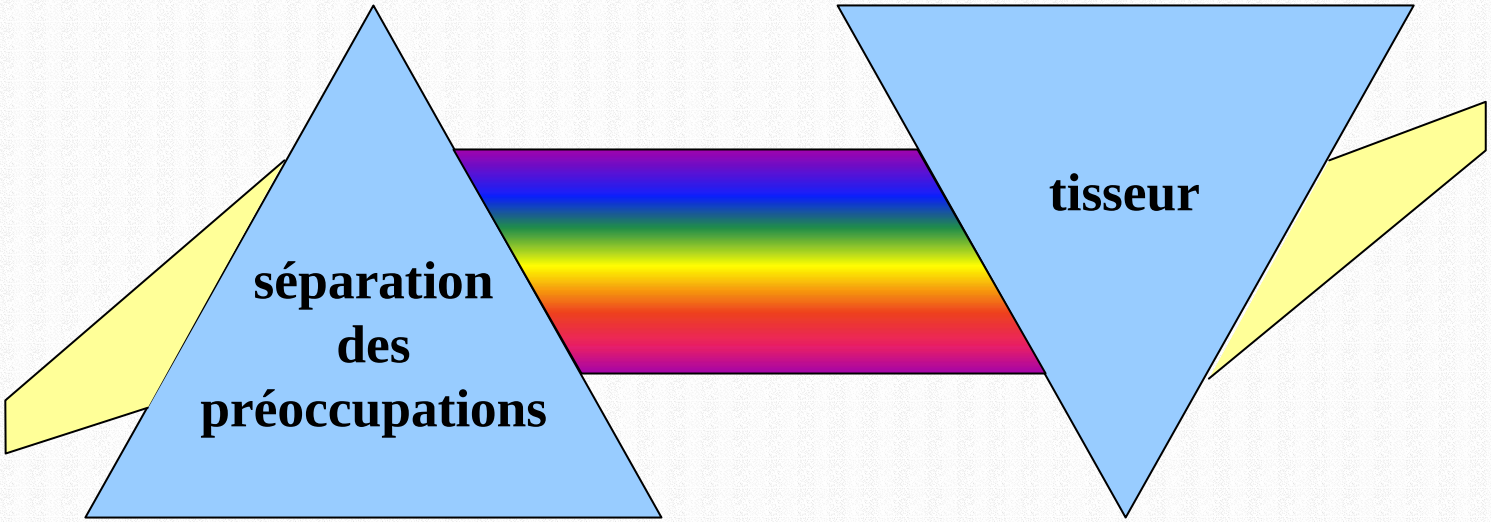
Programmation orientée aspect (AOP)

- Implémenter des préoccupations de façon indépendante
- Combiner ces implémentation pour former le système final
- L'AOP est l'évolution logique de la programmation orientée objet

OOP	AOP
Implémentations modularisées de préoccupations métier courantes	Implémentations modularisées de préoccupations métier et techniques courantes
Implémentation courante d'une préoccupation appelée une *classe*	Implémentation courante d'une préoccupation technique appelée un *aspect*

Etapes d’un développement AOP

- Identifier le préoccupations métier (classes) et techniques (aspects)
- Implémenter chaque préoccupation séparément
- *Tisser* les différentes préoccupations



Copyrigth © Ramnivas Laddad, I want my AOP!

Deux approches AOP : JAC et AspectJ

AspectJ	JAC
Extension du langage Java	Framework AOP, serveur d'applications
Nouvelle grammaire pour exprimer les aspects	Aspects écrits en pur Java
Travaille sur le code source. Chaque modification nécessite une recompilation	Travaille sur le bytecode, permet de modifier, d'ajouter ou de retirer des aspects dynamiquement
Ne gère pas la distribution	Distribue automatiquement les aspects sur des serveurs distants
Permet uniquement le développement d'aspects	Permet le développement et la configuration des aspects
Support IDE pour JBuilder, Forte, et Emacs	IDE UML gérant les aspects
Pas d'aspects pré-développés configurables	Ensemble d'aspects pré-développés configurables
Version courante 1.0.5	Version courante 0.9.1
Open Source Mozilla Public License	LGPL

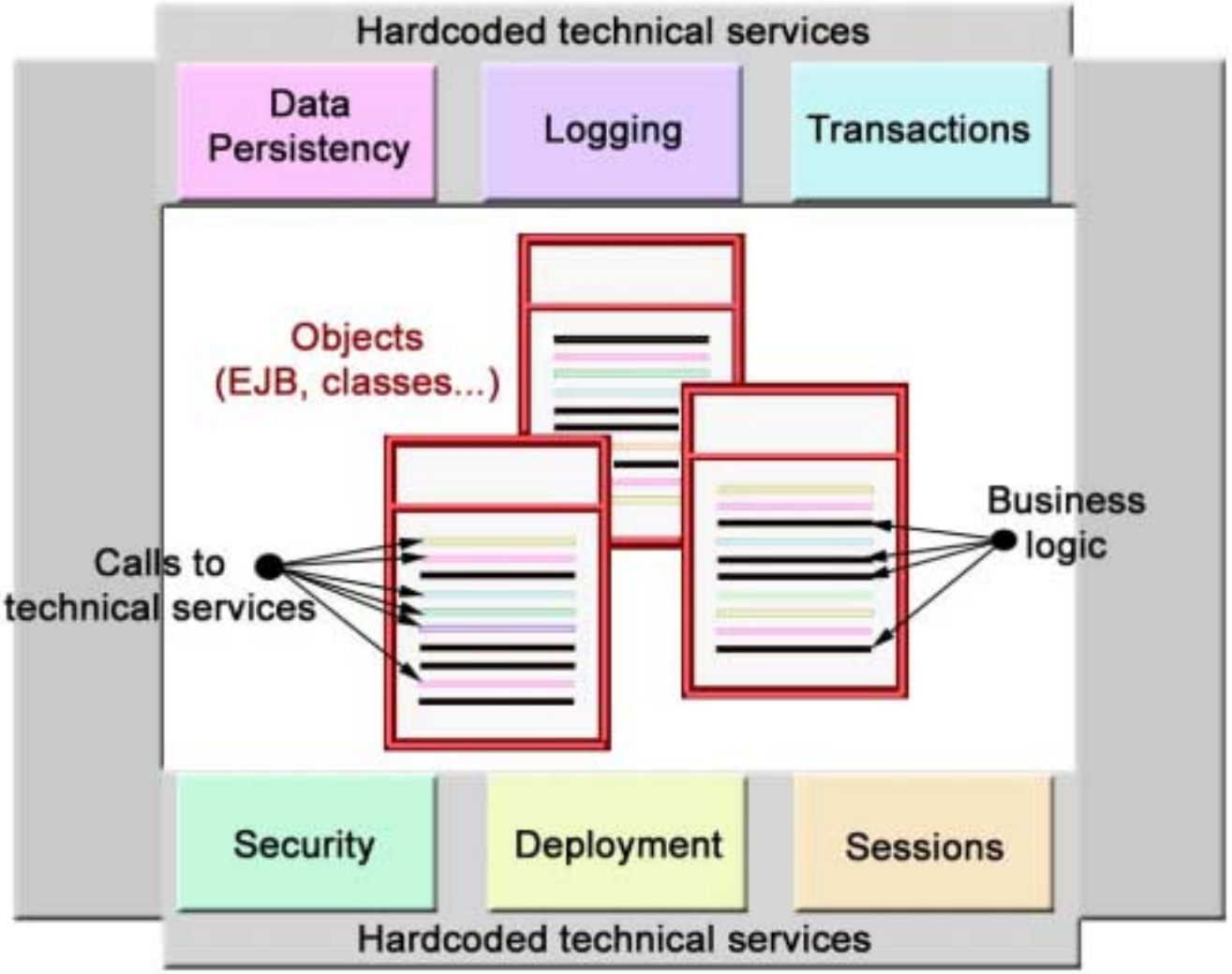
Partie II

JAC: une alternative aux serveurs d’applications

Les serveurs d'applications : une réponse actuelle

- Un serveur d'applications est :
  - Un environnement de développement intégré (IDE)
  - Un conteneur pour des objets métiers, qui fournit des services techniques
  - Complexe
  - Coûteux
  - Utilisé dans 45% des nouveaux projets
- 10 Editeurs se pertagent le marché (Websphere, Weblogic...)

Les services techniques sont modularisés mais ...



An application server container

Exemple des serveurs d'applications J2EE

- Certaines préoccupations sont bien modularisées (ex. persistance, transaction)
- Les EJBs doivent être implémentées d'une certaine manière et ont parfois besoin d'appeler les services techniques du conteneur
- Ces services techniques sont hardcodés et ne peuvent être modifiés => lorsque l'on atteint les limites du conteneur, les fonctionnalités supplémentaires doivent être introduites à la main.
- En pratique, les préoccupations sont toujours entrelacées

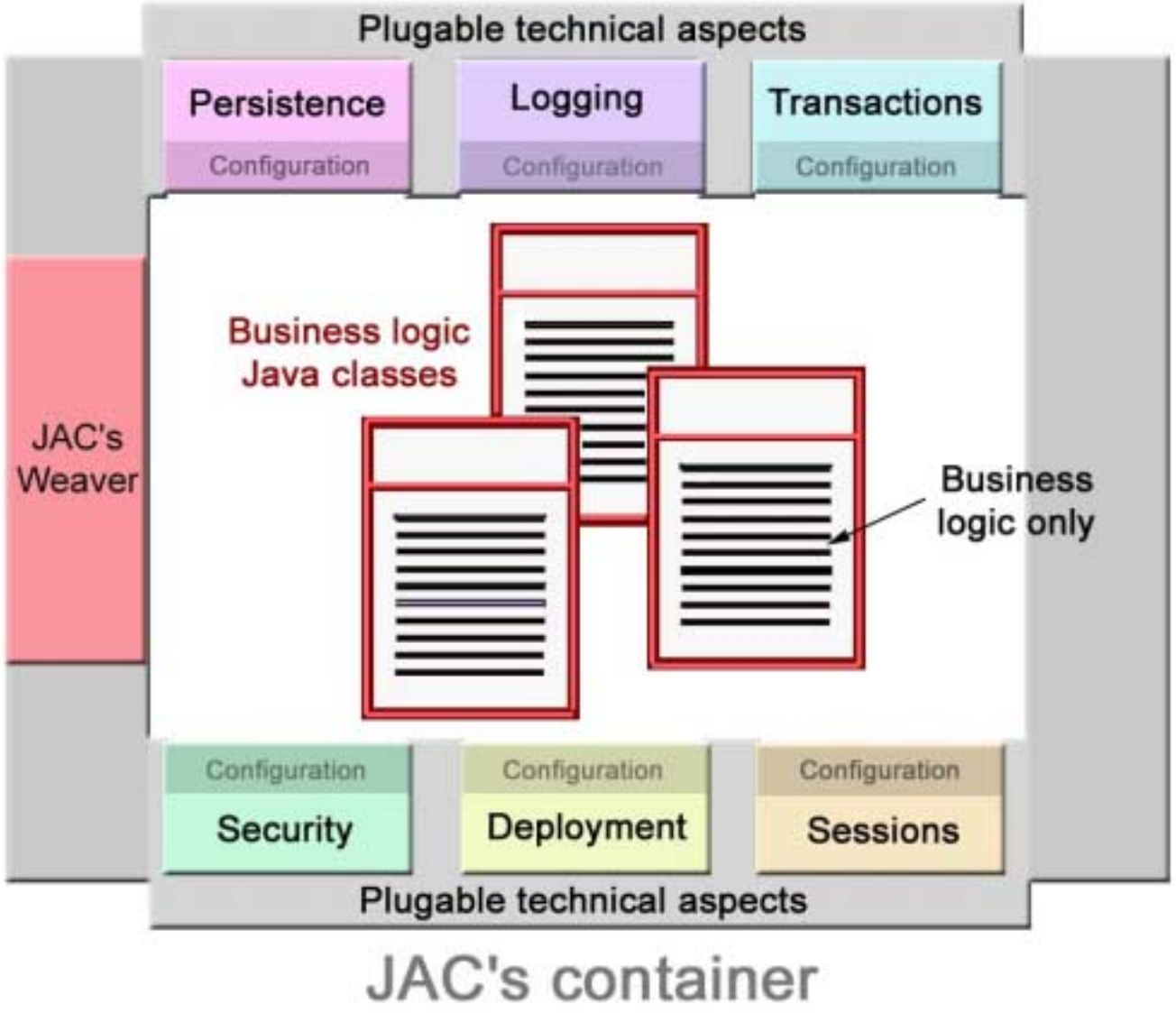
JAC: un framework AOP et un IDE...

- An framework AOP qui fournit
  - Un IDE UML qui intègre l'AOP
  - Des aspects prêts à l'emploi et configurables
  - Un conteneur pour des classes métier pures et des aspects techniques
  - Un noyau qui tisse les aspects aux objets métier à l'exécution
  - Une interface d'administration
  - Un tutoriel, un guide du développeur, guide de programmation, et un ensemble d'exemples

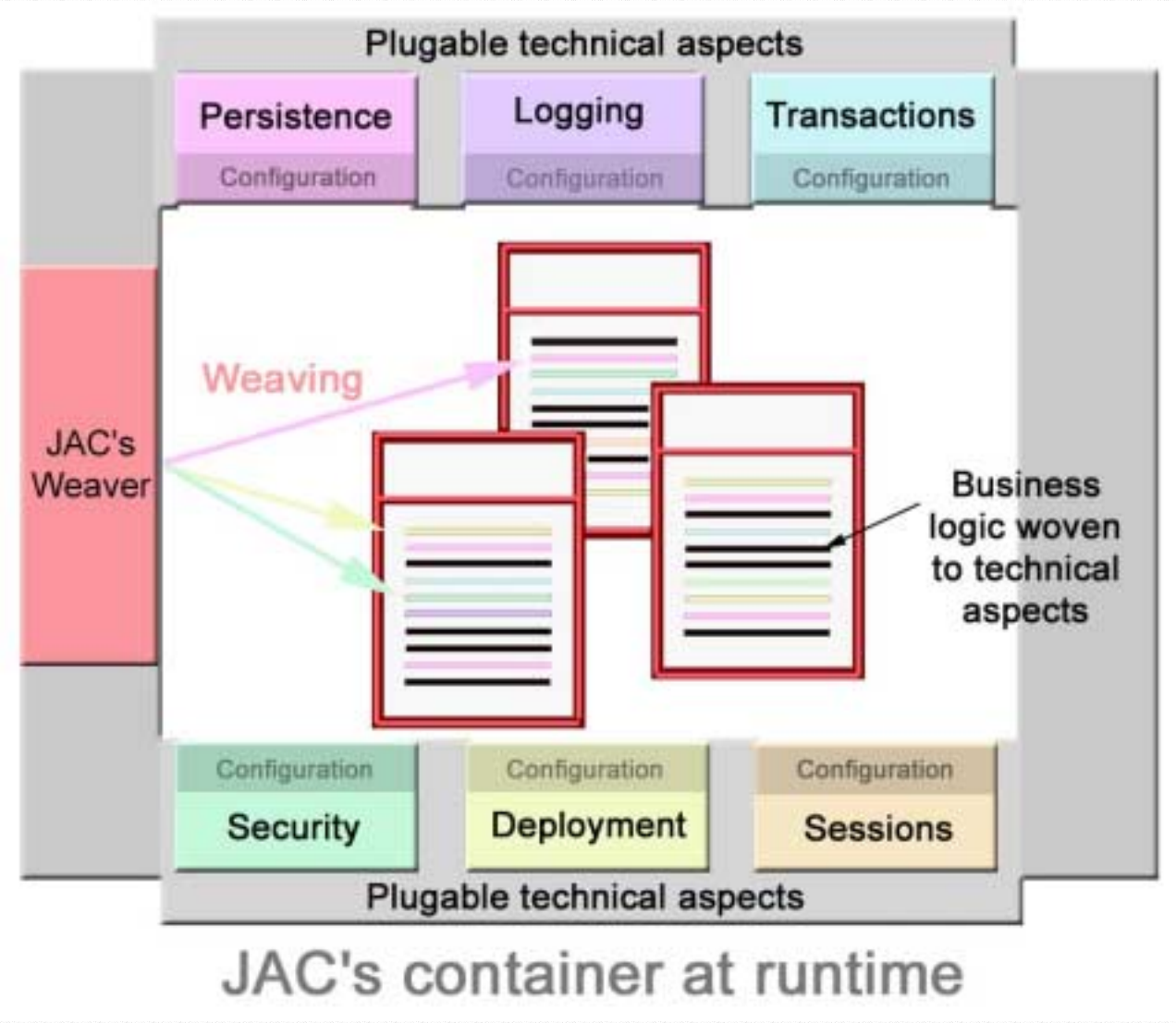
... Mais aussi un serveur d'applications ouvert et flexible

Serveur d'applications J2EE	Serveur JAC
Conteneur pour EJBs	Conteneur pour des classe métier pures et des aspects techniques
Services techniques hardcodés : ne peuvent pas être modifiés	Aspects techniques pré-développés, peuvent être modifiés
Les EJBs doivent être configurés pour appeler les services techniques : entrelacement des préoccupations	Chaque aspect technique est configuré dans un fichier qui lui est propre et tissé à l'exécution : bonne modularisation
Lourd : contient toujours tous les services techniques hardcodés	Léger : chaque aspect technique n'est ajouté qu s'il est utile
Demande de nombreuses compétences	Facile à apprendre et à utiliser
Séparation des préoccupations seulement si elles ont été prévues	Les nouvelles préoccupations peuvent être implémentées dans des aspects bien modulaires
Coûteux	Gratuit sous LGPL
JAC bénéficie des avantages de l'AOP : **tracabilité, meilleure productivité, réutilisation et qualité du code, évolutivité des applications**	

JAC: plus d'entrelacement des préoccupations



Les aspects sont tissés à l'exécution

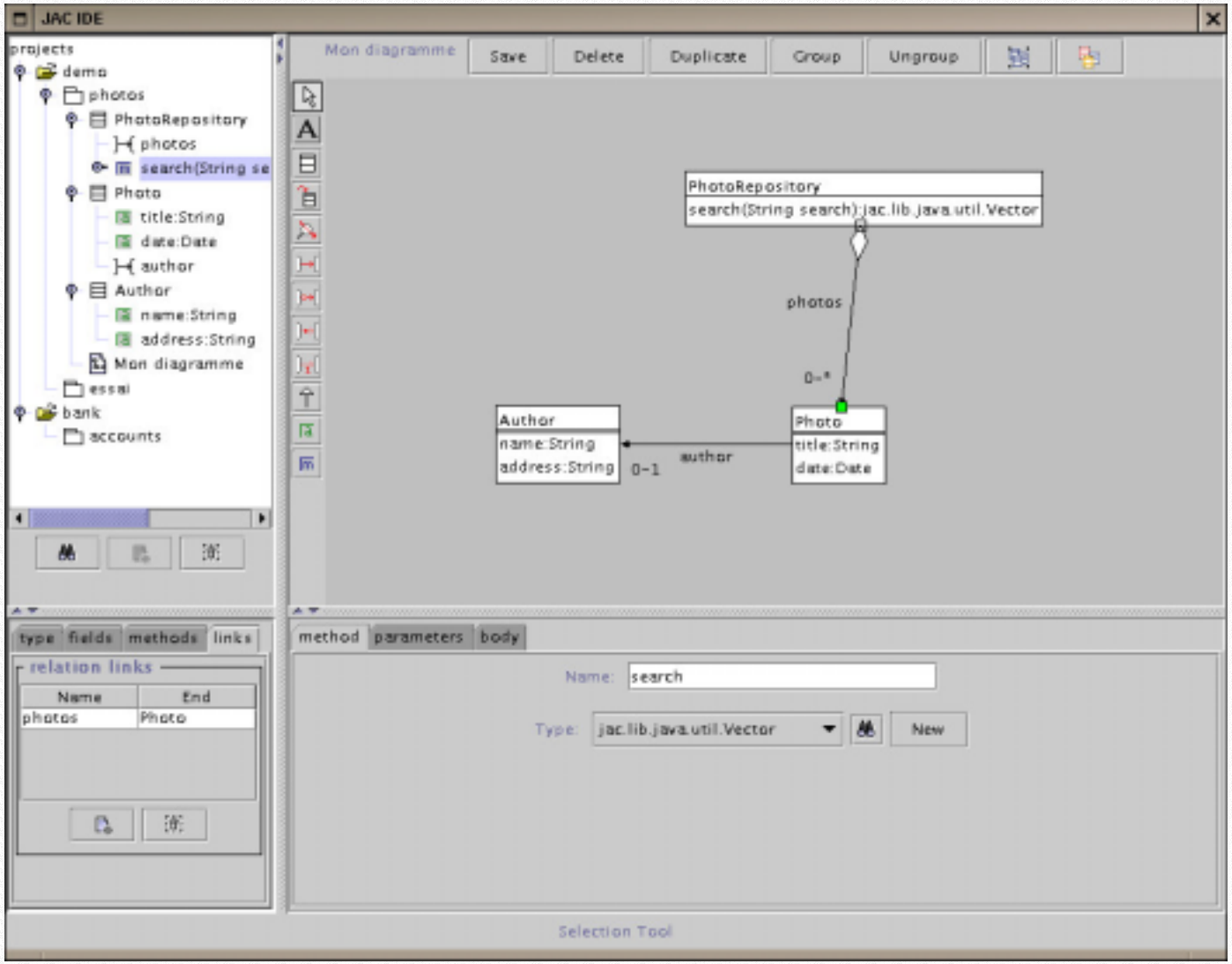


Développer avec JAC

1. Modéliser le cœur métier de l'application avec l'IDE UML. La génération des classes Java exécutables dans le conteneur JAC est automatique.
2. Choisissez les aspects techniques dont vous avez besoin. Toujours en utilisant l'IDE JAC, configurez-les pour coller à votre application.

- Votre application est prête !

UMLAF : UML Aspectual Factory

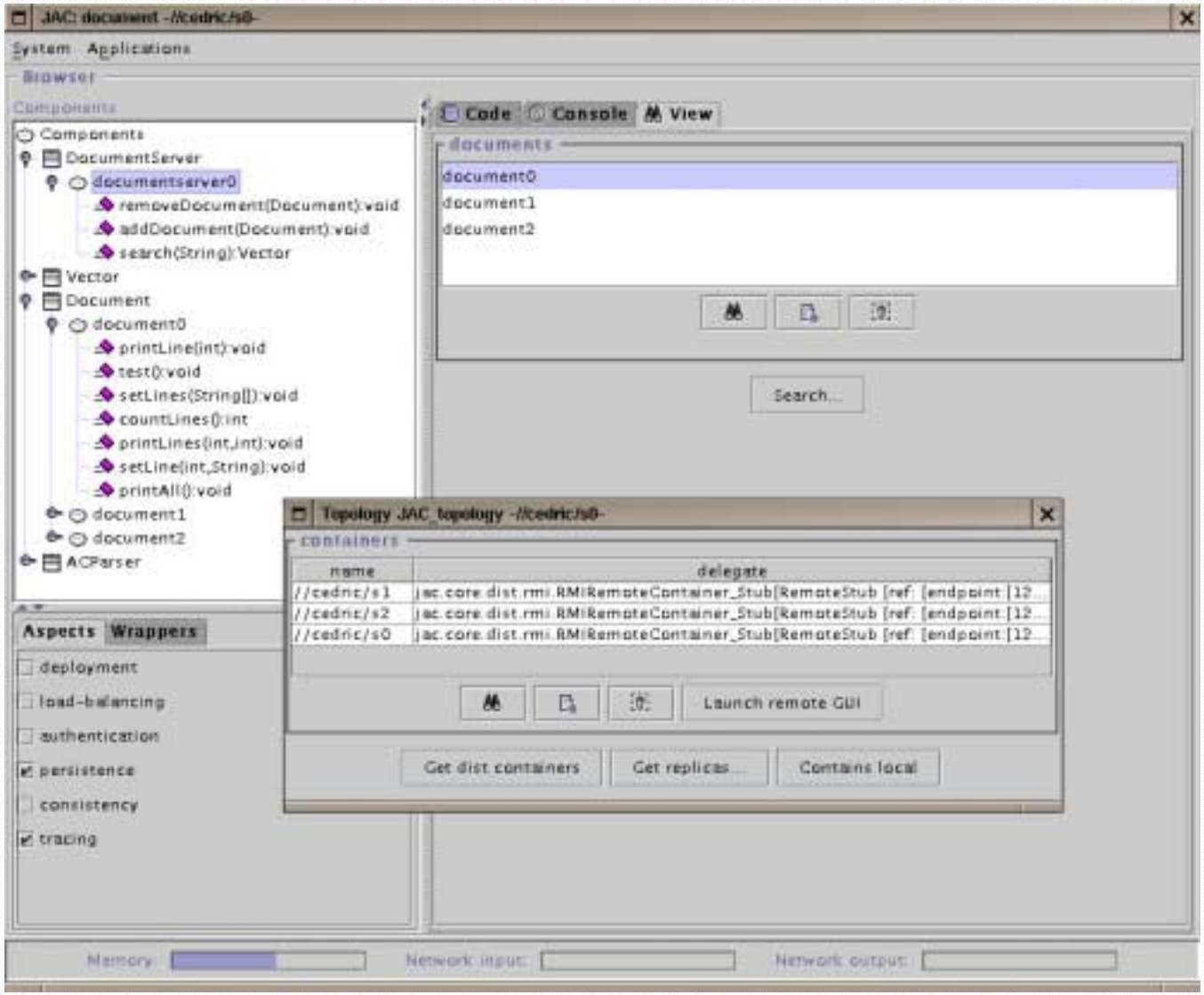


L’interface d’administration

- Vos objets métier ont été générés par UMLAF
- Vous pouvez déjà voir votre application tourner grâce à l’interface d’administration

- Elle vous permet de :
  - Avoir une vue par défaut de vos objets et d’interagir avec eux
  - Débuguer l’application
  - Tisser et détisser les aspects dynamiquement

JAC's administration interface



Les aspects techniques pré-développés

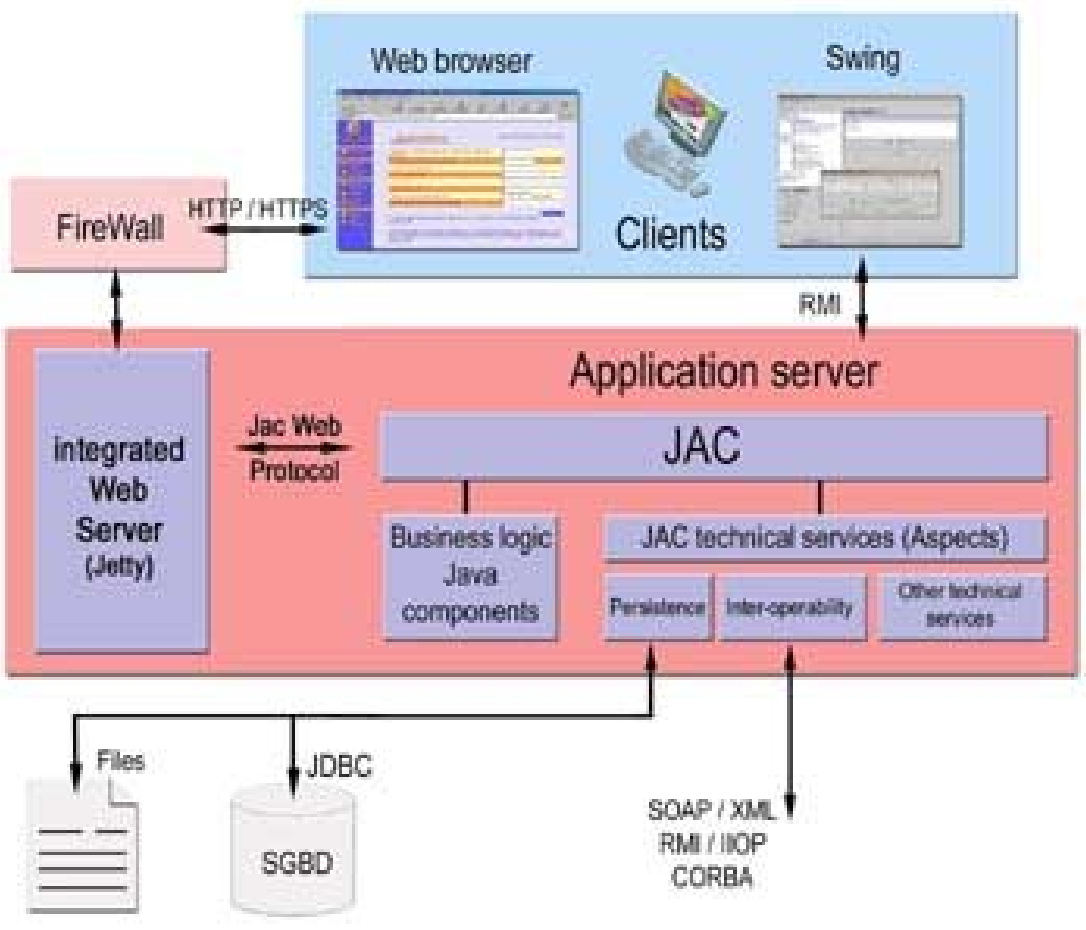
- Persistance (SGBD et Filesystem)
- Authentification
- Sessions
- Transactions
- Déploiement
- Load-balancing
- Broadcasting
- Cohérence des données
- Synchronisation
- Accès distant
- Intégrité
- Utilisateur
- GUI (SWING et Web)

Configuration des aspects

```
// This line makes persistent all instances of all the classes of the
// package jac.samples.contacts
MakePersistent jac.samples.contacts.* ALL;

// This command tells that the instance contactrepository0 is static and
// has not to be created again when opening the application but loaded
// from the database. contactrepository0 is a persistence root for this
// application.
registerStatics jac.samples.contacts.ContactRepository
contactrepository0;

// This line tells the persistence to use the file system storage. Files
// will be saved in the directory data/contacts.

configureStorage jac.aspects.persistence.FSStorage {
"data/contacts" };
```

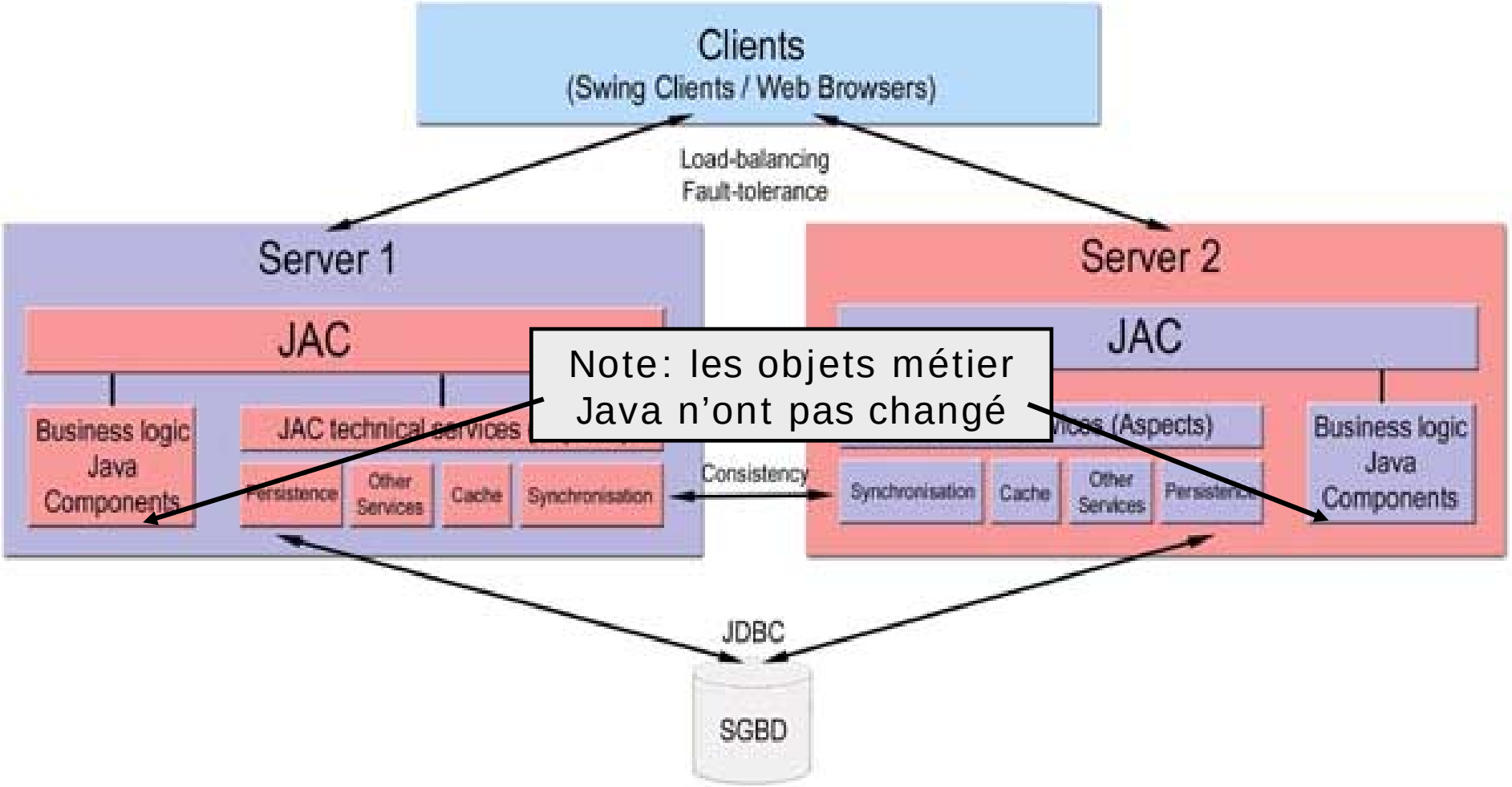
Code de configuration code de l'aspect de persistance de l'application "Contacts", un gestionnaire de contacts simple

Exemple d’architecture (1)



L’architecture de base

Exemple d'architecture (2)



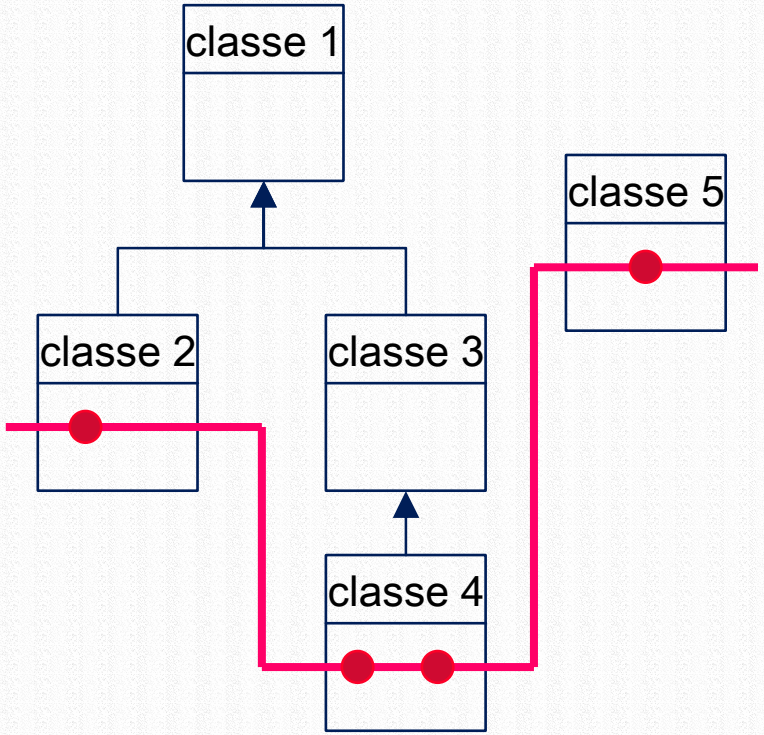
Montée en charge et tolérance aux pannes

Part III

Aperçu technique de JAC

Les concepts de base

- Point de jonction
  - élément du prog. de base
  - **classe, méthode, exception**
    interface, donnée, structure de ctrl
    argument méthode, ...
- coupe transversale
  - ens de pts de jonction ayant un lien logique entre eux
- Méthode d'aspect
  - **traitement additionnel**
- ASPECT :
  - coupes + "méthodes d'aspect" *(advice)* associées à chacun des points de jonction



Notion d’encapsuleurs

- Encapsuleurs = wrappers
  - Objets autonome dont les méthodes encapsulent les méthodes métiers => méthodes d'aspect

Programmation d'un aspect configurable

Méthode de configuration : crée une coupe transversale paramétrable

```
public class MyTracingAC extends AspectComponent {

  public void addTrace(String cExpr,String oExpr,String mExpr) {
    pointcut(cExpr,oExpr,mExpr,TracingWrapper,"trace",false);
  }

  public class TracingWrapper extends Wrapper {
    public Object trace() {
      System.out.println("Method "+method()+" is called on "+
                         wrappee());
      return proceed();
    }
  }
}
```

Méthode d'aspect : appelée par le système quand la coupe transversale correspond aux caractéristiques du point de jonction

Activation de l'aspect

- Une fois programmé, l'aspect de trace doit être configuré pour une application donnée afin que la trace soit activée pour des point de jonction donnés

```
// myApp.jac
applicationName: myApp
launchingClass: myMainClass
aspects: MyTracing trace.acc true
```

Le descripteur d'application doit déclarer que l'aspect est tissé et configuré par le fichier de configuration trace.acc

```
// trace.acc
addTrace ALL ALL "CONSTRUCTORS && FIELDSETTERS";
```

Le fichier de configuration est simple : chaque ligne correspond à l'invocation d'une méthode de configuration déclarée par l'aspect

L'installation dynamique d'aspects sous JAC