Composants logiciels
Introduction aux composants

Sacha Krakowiak
Université Joseph Fourier
Projet Sardes (INRIA et IMAG-LSR)

http://sardes.inrialpes.fr/~krakowia

Des objets aux composants

- **Notion de composant logiciel** (unité de construction)
  - Nécessité perçue très tôt [McIlroy 1968]…
  - … mais conception et réalisation n'ont pas suivi
  - Les objets sont une première approche pour la décomposition
    - Encapsulation (séparation interface-réalisation)
    - Mécanismes de réutilisation (héritage, délégation)
- **Limitations des objets** (comme unité de construction)
  - Pas d'expression explicite des ressources requises par un objet (autres objets, services fournis par l'environnement)
  - Pas de vue globale de l'architecture d'une application
  - Pas de possibilité d'expression de besoins "non fonctionnels" (non spécifiques à une application, par ex. persistance, performances, etc.)
  - Peu d'outils pour le déploiement et l'administration

Modèles et infrastructures

- **Pour mettre en œuvre des composants, il faut**
  - Un modèle de composants, qui définit les entités et leur mode d'interaction et de composition
    - Plusieurs niveaux de modèles
      - Abstrait (définition des entités de base et de leurs relations)
      - Concret (représentation particulière du modèle abstrait)
      - Spécifique à un langage
  - Une infrastructure à composants, qui met en œuvre le modèle et permet de construire, déployer, administrer et exécuter des applications conformes au modèle
- **Situation actuelle**
  - Les infrastructures industrielles (EJB, CCM, .NET, OSGi) n'ont pas de base formelle
  - Des modèles issus de la recherche commencent à être proposés (Fractal, Piccola, …), ainsi que des infrastructures prototypes (cf plus loin, Fractal/Julia)

Que doit fournir un modèle de composants ?

- **Encapsulation**
  - Interfaces = seule voie d'accès, séparation interface-réalisation
  - Possibilité de définir des interfaces multiples
- **Composabilité**
  - Dépendances explicites (expression des ressources fournies et requises)
  - Composition hiérarchique (un assemblage de composants est un composant)
- **Capacité de description globale**
  - Si possible exprimée formellement (langage de description)
- **Réutilisation et évolution**
  - Modèles génériques de composants
  - Adaptation (interface de contrôle)
  - Reconfiguration

Que doit fournir une infrastructure à composants ?

- **Couverture du cycle de vie**
  - Non limitée aux phases de développement et d'exécution
  - Administration
    - Déploiement : installation et activation des composants
    - Surveillance : collecte et intégration de données, tenue à jour de la configuration
    - Contrôle : réaction aux événements critiques (alarme, surcharge, détection d'erreur)
  - Maintenance
    - Maintien de la disponibilité de l'application
    - Évolution (redéploiement, reconfiguration) pour réagir à l'évolution des besoins et de l'environnement
- **Services communs**
  - Propriétés non-fonctionnelles (persistance, transactions, QoS)

Paradigmes de la composition

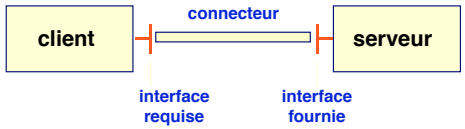
- **Éléments de la composition**
  - **Composant**. Unité de **composition** et de **déploiement**, qui remplit une fonction spécifique et peut être assemblé avec d'autres composants. À cet effet, il porte une description des interfaces requises et fournies.
  - **Connecteur**. Élément permettant d'assembler des composants en utilisant leurs interfaces fournies et requises. Remplit 2 fonctions : **liaison** et **communication**.
  - **Configuration**. Un assemblage de composants (peut ou non, selon le modèle, être lui-même un composant).

N.B.1 La notion de composant est préservée à l'exécution pour faciliter
- l'adaptation dynamique
- la répartition

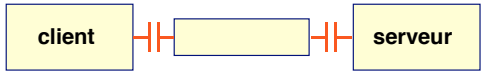
N.B.2 La différence entre composant et connecteur est une différence de fonction, non de nature : un connecteur est lui-même un composant

Exemples de schémas de composition (1)

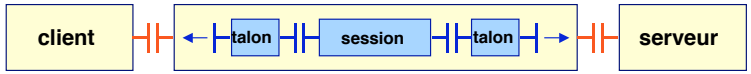
Client-serveur



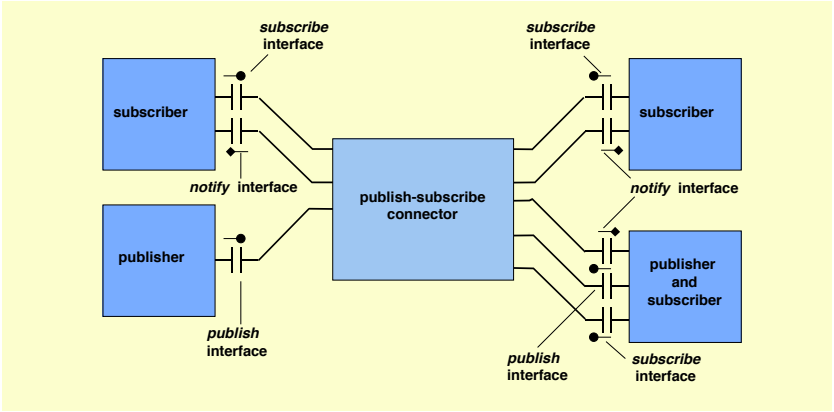
Une autre vue : le connecteur comme composant



Une autre vue : le connecteur comme composant composite



Exemples de schémas de composition (2)



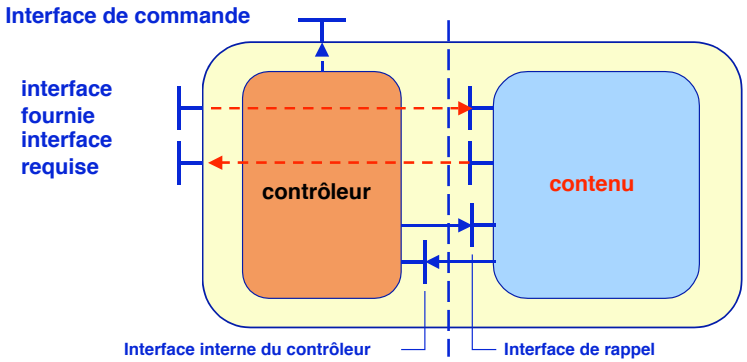
Décrire la composition

- Pourquoi une description formelle ?
  - Donner un support concret aux notions d'architecture logicielle
    - Composants, interfaces, connecteurs, configuration
  - Permettre un traitement formel (vérification, preuves)
    - En particulier contrôle de types
  - Permettre d'automatiser (ou d'assister) des opérations globales (mettant en jeu des configurations)
    - Déploiement
    - Reconfiguration
  - Servir d'aide à la documentation (description globale, visualisation)
- Situation actuelle
  - Notion d'ADL (*Architecture Description Language*)
  - Pas de standard reconnu (des tentatives)
    - ACME, ADML, Xarch, UML-2
  - Vers l'utilisation de XML comme support commun (invisible) ?

Éléments d'un ADL

- Description des composants
  - Définition des interfaces (fournies, requises)
  - Types (synchrone, asynchrone, "porte", etc.)
  - Signatures, contrats, annotations
- Description des configurations
  - Association interfaces fournies-requises
    - Directe (invisible, réalisée par édition de liens)
    - Via un connecteur : description, "rôle" = interface
  - Composition hiérarchique (si le modèle l'autorise)
    - Interfaces exportées/importées
  - Interface graphique (visualisation, action)
  - Aspects dynamiques (expérimental)
- Exemples, cf plus loin (Fractal)

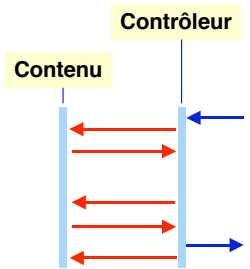
Organisation logique d'un composant



La séparation entre contrôleur et contenu est motivée par la séparation des préoccupations : isoler la partie purement fonctionnelle (propre à l'application)

Fonctions d'un contrôleur

- Gestion du cycle de vie
  - Création, destruction
  - Activation, passivation
- Gestion des composants inclus
  - … si le modèle comporte des composants composites
- Liaison
  - Connexion avec d'autres composants
- Médiation
  - Gestion des interactions avec d'autres composants
  - Médiation pour la fourniture de services externes
  - Met en œuvre l'inversion du contrôle
- Autres opérations réflexives
  - … si le modèle le permet



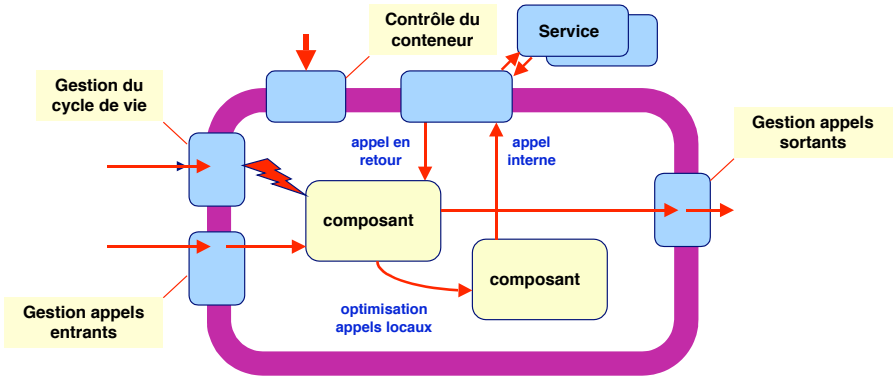
Canevas pour composants

- **Infrastructures à conteneurs**
  - Canevas de base pour le support des composants
  - Séparation des préoccupations
    - La partie "contenu" réalise les fonctions de l'application
    - La partie "conteneur" fournit les fonctions de l'infrastructure
  - Fonctions du conteneur
    - Mise en œuvre des fonctions du contrôleur (gestion, médiation)
    - Allocation des ressources aux composants
    - Réalisation des aspects "non fonctionnels" (services communs)
- **Exemples** (sous des formes diverses)
  - *Enterprise Java Beans* (EJB)
  - *CORBA Component Model* (CCM)
  - Julia (réalisation du modèle Fractal)
  - OSGi
  - Avalon (Apache-Djakarta)

Conteneur

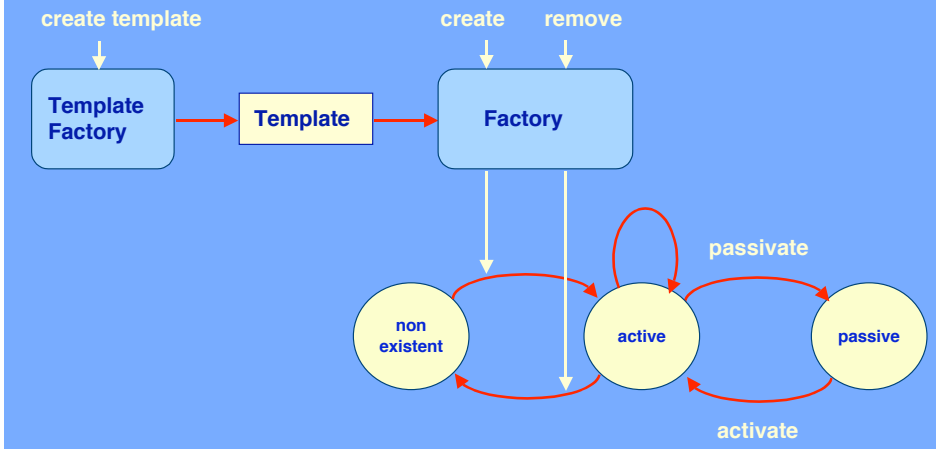
- **Mise en œuvre du contrôleur de composants**
  - Conteneur : infrastructure de gestion pour un ou plusieurs composants
    - Par composant ou par "type"
  - Rappel des fonctions : cycle de vie, médiation, gestion de services, composition (si composants composites)
  - Génération automatique des conteneurs à partir
    - des descriptions d'interfaces des composants (cf. talons)
    - de paramètres de configuration fournis
  - "Contrats" entre le conteneur et les composants inclus
    - Interface interne du contrôleur
    - Interfaces de rappel des composants

Schéma générique de conteneur



Illustrations spécifiques pour différents modèles : EJB, CCM, Fractal/Julia, OSGi, …

Cycle de vie des composants (1)



Cycle de vie des composants (2)

- **Mécanismes pour économiser les ressources**
  - Activation-passivation
    - *passivate* : Élimine le composant de la mémoire si non utilisé (en stockant son état) - appelé par le conteneur
    - *activate* : Restitue l'état du composant ,qui peut être réutilisé
  - *Pool* d'instances
    - Le conteneur maintient un *pool* fixe d'instances de composants qui peuvent donner vie à des composants "virtuels"
    - Un composant virtuel incarné par une instance du *pool* devient utilisable, à condition de charger son état => méthodes de rappel pour sauver-restaurer l'état
    - Évidemment plus simple pour les composants sans état
  - Différences entre passivation et *pool*
    - Passivation : géré par le conteneur, pas de méthodes de gestion de l'état

cf illustrations plus tard dans le cours (EJB)

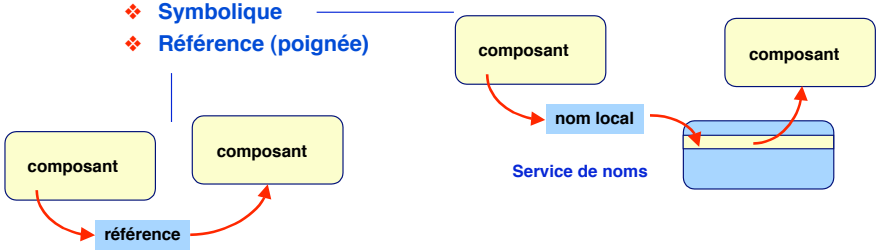
Cycle de vie des composants (3)

- **Maison (*home*)**
  - Partie du conteneur (visible de l'extérieur) qui gère le cycle de vie des composants qu'il inclut (du même type)
    - Engendré automatiquement (à partir d'un IDL)
  - Fonctions
    - Créer / détruire les composants : implémente *Factory*, utilise les mécanismes d'économie (*pool*, passivation)
    - Gérer les composants pendant leur existence (en particulier nommer, localiser)
  - Interface fournie
    - *create*, *find*, *remove*
    - Utilise une interface de rappel (à implémenter par le programmeur)

cf illustrations plus tard dans le cours (EJB)

Désignation des composants

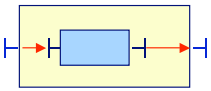
- **Principe : désignation contextuelle**
  - Désignation dans un contexte global
    - Service de noms (CORBA, J2EE/JNDI, …)
    - Clé pour les objets persistants (clé primaire SGBD)
  - Désignation locale pour éviter noms globaux "en dur"
    - Symbolique
    - Référence (poignée)



Liaison des composants

Les principes généraux de la liaison s'appliquent ; quelques spécificités.

- **Plusieurs types de liaison selon localité**
  - Entre composants de même niveau dans un même espace : référence dans cet espace (exemple : référence Java)
  - Entre composants dans des espaces différents : objet de liaison (avec éventuellement optimisations locales)
    - Exemple EJB :
      - Dans un même conteneur : LocalObject
      - Entre deux conteneurs : RemoteObject
      - Entre client et serveur : RMI (stub + skeleton)
  - Entre composants inclus (dans les modèles qui l'autorisent)
    - Exportation - Importation



Appel d'un composant

- **Principe**
  - Le conteneur réalise une médiation pour l'appel d'une méthode d'un composant, via un objet intermédiaire (intercepteur)
  - Comme le reste du conteneur, l'intercepteur est engendré automatiquement (fait partie de la chaîne de liaison)
  - Le conteneur fournit en général une optimisation locale pour les appels entre composants qu'il contient
- **Fonctions réalisées**
  - Lien avec gestion de ressources (activation d'un composant passivé)
  - Sécurité
  - Fonctions supplémentaires "à la carte" (intercepteurs programmables)

Vers des infrastructures à composants adaptables

- **Motivations**
  - Les systèmes commerciaux à composants actuels sont peu adaptables (structure fermée de conteneurs)
- **Quelques pistes …**
  - Conteneurs ouverts
    - Rendre visible la structure interne des conteneurs (séparation des fonctions)
    - Définir des interfaces de contrôle internes au conteneur
    - Piste suivie dans OpenCCM
  - Intercepteurs programmables
    - Permettre l'insertion d'intercepteurs dans la chaîne d'appel
      - Flexinet, Julia, JBoss
  - AOP (*Aspect Oriented Programming*)
    - Insertion de code en des points prédéfinis

cf illustrations plus tard dans le cours

Exemple de système à composants : Fractal

Modèle, implémentation, utilisation

Sacha Krakowiak
Université Joseph Fourier
Projet Sardes (INRIA et IMAG-LSR)

http://sardes.inrialpes.fr/~krakowia

Le modèle à composants Fractal

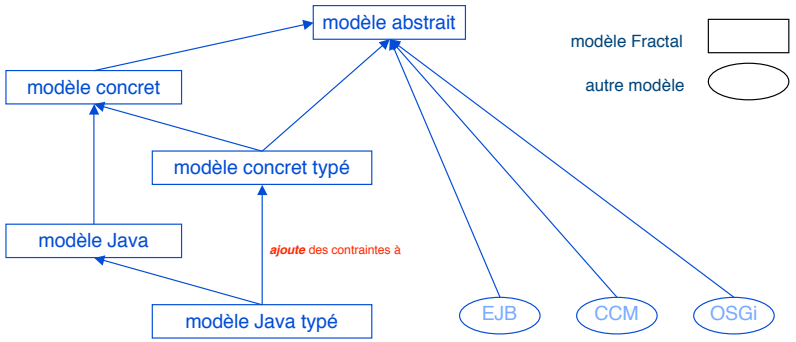
- **Pourquoi étudier ce modèle ?**
  - Modèle général, peu de restrictions
  - Fondé sur une base mathématique rigoureuse
  - Il existe une implémentation de référence (Julia)
    - Disponible en logiciel libre
  - Commence à être diffusé dans la communauté de recherche
- **Source**
  - http://fractal.objectweb.org
    - Contient le code source et la documentation
- **Auteurs**
  - Éric Bruneton, Thierry Coupaye (France Telecom R&D)
  - Jean-Bernard Stefani (INRIA Rhône-Alpes)
- **Historique**
  - Début du projet : janvier 2002
  - Première version : juillet 2002
  - Version actuelle (V2) : novembre 2003

Introduction à Fractal

- **Qu'est ce que Fractal ?**
  - Modèle de composants + canevas logiciel (APIs)
    - Indépendant de toute implémentation …
    - … mais une implémentation doit lui être conforme
      - Aujourd'hui, une implémentation de référence
    - Peut servir de base à diverses extensions
- **Pourquoi Fractal ?**
  - Limitations des modèles actuels de composants
    - Pas de fondement rigoureux
    - Expressivité restreinte
      - Pas de composition hiérarchique
      - Pas de partage de composants
    - Peu de possibilités d'adaptation et de reconfiguration
    - Pas d'outils de description globale

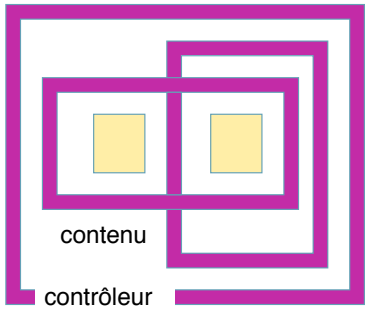
Modèles : aperçu

- **Hiérarchie de modèles de plus en plus concrets**
  - hiérarchie extensible



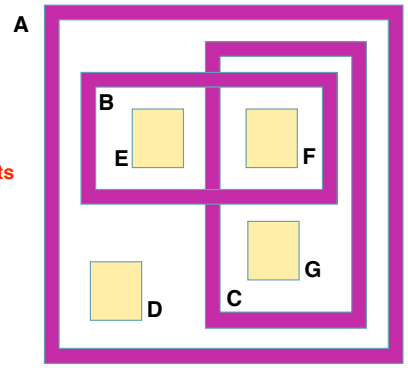
Modèles : modèle abstrait (1)

- **Concepts :**
  - composants, signaux, noms
- **Composant :**
  - contrôleur + contenu
  - peuvent être imbriqués :
    - sous-composants
    - composants partagés
    - composants composites / primitifs
- **Contrôleur :**
  - peut fournir une représentation du composant :
    - introspection, intercession
  - peut intercepter les signaux entrants et sortants
  - peut modifier le comportement des sous contrôleurs



Modèles : modèle abstrait (2)

- **Deux sortes de composants**
  - Primitifs
    - Encapsule du code exécutable
    - Peut servir à encapsuler du code existant (*legacy*)
  - Composite
    - Encapsule un ensemble de composants
    - Niveau d'emboîtement quelconque …
    - … d'où le nom de *Fractal*
- **Un composant peut être partagé**
  - A contient B, C, D
  - B contient E, F
  - C contient F, G
  - F partagé entre B et C

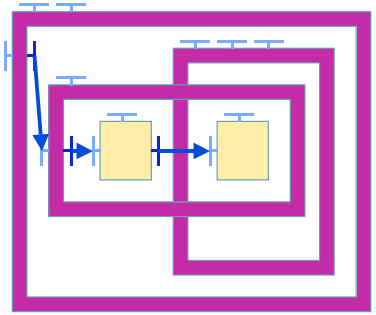


Modèles : modèle abstrait

- Composant = entité à l'exécution
  - Service, ressource, donnée, activité, processus, protocole, liaison, …
  - Pas de « cibles » particulières (application / middleware / système d'exploitation)
- Contrôleur
  - Introspection et reconfiguration dynamique
    - Administration, outillage
  - Gestion des aspects techniques ou « non fonctionnels »
    - Transactions, sécurité, …
- Récursion
  - Gestion à grain arbitraire : passage à l'échelle, homogénéité
- Partage : modélisation des
  - Resources : données, pools, caches, connections, …
  - Activités : threads, transactions, procédés / tâches, …
  - Domaines de contrôle : défaillance, sécurité, persistance, mobilité, …

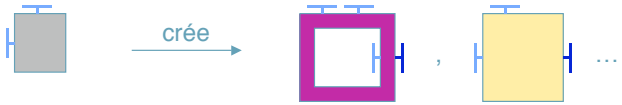
Modèles : modèle concret

- Nouveaux concepts
  - interface, référence d'interface, liaison
  - instanciation, typage
- Interface : point d'accès
  - cliente (requise) / serveur (fournie)
    - émet / reçoit des signaux
  - interne, externe
- Référence d'interface
  - nom désignant une interface
  - a le type de l'interface désignée
- Liaison : interactions
  - simple :
    - relie exactement deux interfaces *locales*
  - composée (ensemble de liaisons simples et de composants) :
    - relie plusieurs interfaces locales ou distantes
  - normale, import, export



Modèles : modèle concret

- Instanciation : deux méthodes possibles
  - Factory : crée n'importe quel type de composant



  - Template : crée des composants similaires à lui même

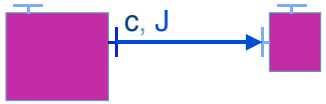
- Typage : minimal

Modèles : modèle Java

- Définit une API Java pour le modèle concret
  - Noms
    - Name, NamingContext, Binder
  - Interfaces et typage
    - ComponentIdentity, InterfaceReference, Type
  - Instanciation et bootstrap
    - Factory, Template, Fractal
  - Interfaces de contrôle
    - AttributeController
    - BindingController, UserBindingController
    - ContentController
    - LifeCycleController

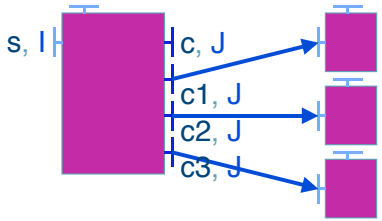
Modèles : modèle concret typé

- **Type de composant**
  - ensemble de types d'interface
  - *non modifiable* à l'exécution



c : interface de type singleton

- **Type d'interface**
  - nom
  - signature
  - contingence
    - obligatoire, optionnelle
  - cardinalité
    - singleton, collection

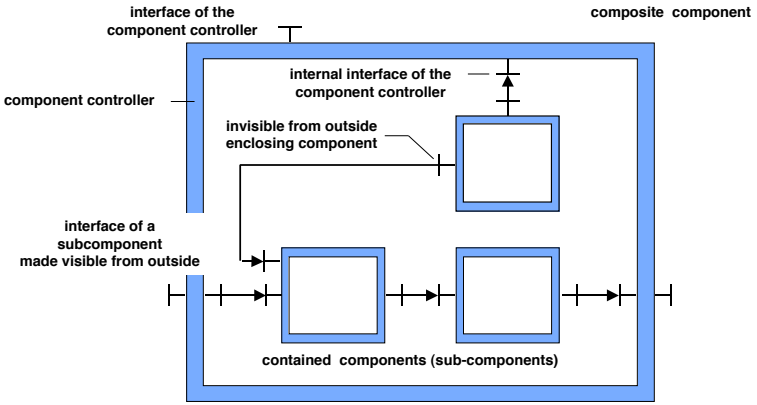


c : interface de type collection

Interfaces (1)

- **Un composant a trois sortes d'interfaces**
  - Fournies (serveur)
    - Décrivent les services fournis (réalisés) par le composant
  - Requises (client)
    - Décrivent les services dont le composant a besoin pour réaliser ses propres services
  - De contrôle
    - Fournissent des services permettant de gérer le composant
    - Exemple: *BindingController* pour contrôler les liaisons du composant avec d'autres composants

Interfaces (2)



[From Fractal documentation : http://fractal.objectweb.org/]

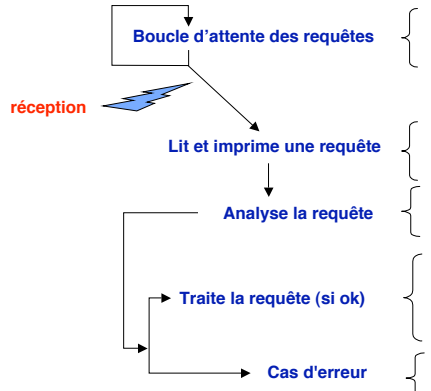
Exemple d'utilisation de Fractal

Extrait du tutorial Fractal ("Developing with Fractal", Éric Bruneton, nov. 2003) : fractal.objectweb.org

- **Application simple : un serveur Web élémentaire**
  - Reçoit, analyse, traite les requêtes
- **Étapes d'une conception à base de composants**
  - Identifier les composants
    - Statiques (existent toute la vie de l'application)
      - Correspondent aux services
    - Dynamiques (créés et détruits dynamiquement)
      - Correspondent aux données
  - Définir l'architecture globale
    - Dépendances et relations entre composants
  - Définir les contrats entre composants
    - Spécification précise des interfaces
      - Définir des interfaces stables (qui ne changeront pas en cas d'évolution)
      - Éliminer toute dépendance par rapport à une implémentation particulière

Un exemple extrait du tutorial Fractal

Programmation "classique" d'un serveur Web




```
public class Server implements Runnable {
  private Socket s;
  public Server (Socket s) { this.s = s; }
  public static void main (String[] args) throws IOException {
    ServerSocket s = new ServerSocket(8080);
    while (true) { new Thread(new Server(s.accept())).start(); }
  }
  public void run () {
    try {
      InputStreamReader in =
                  new InputStreamReader(s.getInputStream());
      PrintStream out = new PrintStream(s.getOutputStream());
      String rq = new LineNumberReader(in).readLine();
      System.out.println(rq);
      if (rq.startsWith("GET ")) {
        File f = new File(rq.substring(5, rq.indexOf(' ', 4)));
        if (f.exists() && !f.isDirectory()) {
          InputStream is = new FileInputStream(f);
          byte[] data = new byte[is.available()];
          is.read(data);
          is.close();
          out.print("HTTP/1.0 200 OK\n\n");
          out.write(data);
        } else {
          out.print("HTTP/1.0 404 Not Found\n\n");
          out.print("<html>Document not found.</html>");
        }
      }
      out.close();
      s.close();
    } catch (IOException _) { }
  }
}
```

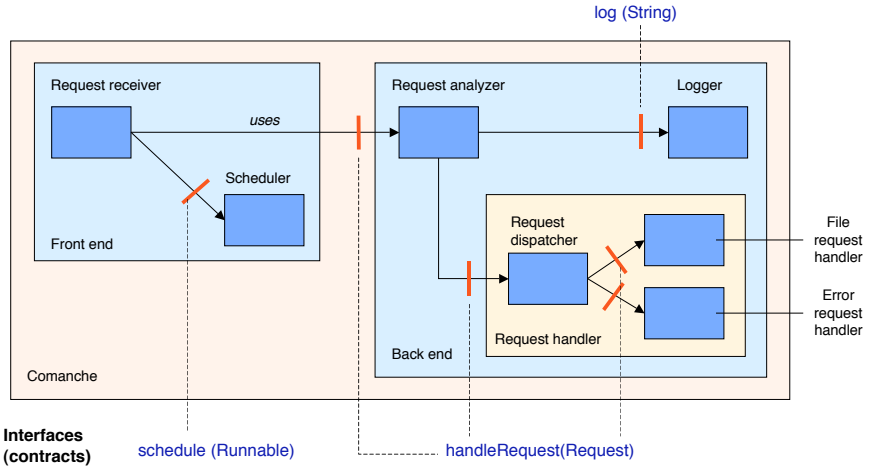
Extracts the file name from the request

Looks for the first occurrence of " " starting at index 4

Une première expérience de décomposition

- **Objectif**
  - **Illustrer les principes de décomposition**
    - **Choix des composants**
    - **Définition des interfaces**
  - **Faciliter la mise en œuvre et l'évolution de l'application**
    - **Remplacement de composants**
    - **Construction et déploiement**
- **Ce qu'on va faire**
  - **Définir des "pseudo-composants" (en Java, pas en Fractal), intégrant une partie fonctionnelle et une partie non-fonctionnelle (c'est cette dernière qui sera fournie par Fractal, dans un deuxième temps)**
  - **On montrera ainsi certaines de fonctions fournies par Fractal pour l'adaptation et pour l'aide au déploiement**

Architecture globale du serveur Web



Composants du serveur Web (1)

Définitions d'interfaces (en Java)

```
public interface Logger {void log (String msg)}
public interface Scheduler
      {void schedule (Runnable task)}
public interface RequestHandler
      {void handleRequest (Request r)  throws  IOException}
```

```
public class Request {
  public Socket s;
  public Reader in;
  public PrintStream out;
  public String url;
  public Request (Socket s)
            {this.s = s}
}
```

Implémentation

N.B. L'implémentation est un processus indépendant de la conception

Les composants qui ne dépendent d'aucun autre peuvent être programmés comme des classes Java

```
public class BasicLogger implements Logger {
  public void log (String msg) { System.out.println(msg); }
}

public class SequentialScheduler implements Scheduler {
  public synchronized void schedule (Runnable task) { task.run(); }
}

public class MultiThreadScheduler implements Scheduler {
  public void schedule (Runnable task) { new Thread(task).start(); }
}
```

Composants du serveur Web (2)

Comment implémenter un composant ayant des dépendances ?
Implémentation par une simple classe :

```
public class RequestReceiver {
  private Scheduler s = new MultiThreadScheduler();
  private RequestHandler rh = new RequestAnalyzer();
  // rest of the code not shown
}
```

Inconvénient : aucune indépendance par rapport aux implémentations des composants utilisés. D'où la solution ci-dessous (inversion du contrôle) :

```
public class RequestReceiver {
  private Scheduler s;
  private RequestHandler rh;
  public RequestReceiver (Scheduler s, RequestHandler rh) { this.s = s; this.rh = rh; }
  // rest of the code not shown
}
```

Résout le problème ci-dessus, mais seulement au déploiement (appel du constructeur de la classe RequestReceiver**; pas possible de changer les implémentations en cours d'exécution**

Composants du serveur Web (3)

Implémentation en Fractal du composant `RequestReceiver`

interface fonctionnelle ↘ Runnable, BindingController ↙ interface non fonctionnelle

```
public class RequestReceiver implements Runnable, BindingController {
  private Scheduler s;
  private RequestHandler rh;
  // configuration concern
  public String[] listFc () { return new String[] { "s", "rh" }; }
  public Object lookupFc (String itfName) {
    if (itfName.equals("s")) { return s; }
    else if (itfName.equals("rh")) { return rh; }
    else return null;
  }
  public void bindFc (String itfName, Object itfValue) {
    if (itfName.equals("s")) { s = (Scheduler)itfValue; }
    else if (itfName.equals("rh")) { rh =
          (RequestHandler)itfValue; }
  }
  public void unbindFc (String itfName) {
    if (itfName.equals("s")) { s = null; }
    else if (itfName.equals("rh")) { rh = null; }
  }
  // functional concern
  public void run ()
}
```

Doit implémenter l'interface `BindingController`, qui fournit 4 méthodes : `listFc`, `lookupFc`, `bindFc`, `unbindFc` pour gérer les composants liés

```
public void run () {
  try {
    ServerSocket ss = new ServerSocket(8080);
    while (true) {
      final Socket socket = ss.accept();
      s.schedule(new Runnable () {
        public void run () {
          try { rh.handleRequest(new Request(socket)); }
            catch (IOException _) { }
        }
      });
    }
  } catch (IOException e) { e.printStackTrace(); }
}
```


Composants du serveur Web (4)

Implémentation en Fractal du composant `RequestDispatcher`

```
public class RequestDispatcher implements RequestHandler, BindingController {
  private Map handlers = new TreeMap();
  // configuration concern
  public String[] listFc () {
    return (String[])handlers.keySet().toArray(new String[handlers.size()]);
  }
  public Object lookupFc (String itfName) {
    if (itfName.startsWith("h")) { return handlers.get(itfName); }
    else return null;
  }
  public void bindFc (String itfName, Object itfValue) {
    if (itfName.startsWith("h")) { handlers.put(itfName, itfValue); }
  }
  public void unbindFc (String itfName) {
    if (itfName.startsWith("h")) { handlers.remove(itfName); }
  }
  // functional concern
  public void handleRequest (Request r) throws IOException {}
}
```

```
public void handleRequest (Request r)
                  throws IOException {
  Iterator i = handlers.values().iterator();
  while (i.hasNext()) {
    try {
      ((RequestHandler)i.next()).handleRequest(r);
      return;
    } catch (IOException _) { }
  }
}
```

Ici lescomposants liés le sont à travers une interface de type "Collection". Convention de nommage : un préfixe commun (ici "h").

Composants du serveur Web (5)

Implémentation des composants `FileRequestHandler` **et** `ErrorRequestHandler`

```
public class FileRequestHandler implements RequestHandler {
  public void handleRequest (Request r) throws IOException {
    File f = new File(r.url);
    if (f.exists() && !f.isDirectory()) {
      InputStream is = new FileInputStream(f);
      byte[] data = new byte[is.available()];
      is.read(data);
      is.close();
      r.out.print("HTTP/1.0 200 OK\n\n");
      r.out.write(data);
    } else { throw new IOException("File not found"); }
  }
}

public class ErrorRequestHandler implements RequestHandler {
  public void handleRequest (Request r) throws IOException {
    r.out.print("HTTP/1.0 404 Not Found\n\n");
    r.out.print("<html>Document not found.</html>");
  }
}
```

Composants du serveur Web (6)

Implémentation du composant `RequestAnalyzer`

```
public class RequestAnalyzer
      implements RequestHandler, BindingController {
  private Logger l;
  private RequestHandler rh;
  // configuration concern
  public String[] listFc ()
       { return new String[] { "l", "rh" }; }
  public Object lookupFc (String itfName) {
    if (itfName.equals("l")) { return l; }
    else if (itfName.equals("rh")) { return rh; }
    else return null;
  }
  public void bindFc (String itfName, Object itfValue) {
    if (itfName.equals("l")) { l = (Logger)itfValue; }
    else if (itfName.equals("rh")) { rh =
                (RequestHandler)itfValue; }
  }
  public void unbindFc (String itfName) {
    if (itfName.equals("l")) { l = null; }
    else if (itfName.equals("rh")) { rh = null; }
  }
```

```
// functional concern
  public void handleRequest (Request r)
                throws IOException {
    r.in = new InputStreamReader(r.s.getInputStream());
    r.out = new PrintStream(r.s.getOutputStream());
    String rq = new LineNumberReader(r.in).readLine();
    l.log(rq);
    if (rq.startsWith("GET ")) {
      r.url = rq.substring(5, rq.indexOf(' ', 4));
      rh.handleRequest(r);
    }
    r.out.close();
    r.s.close();
  }
}
```


Configuration du serveur Web (1)

Configuration par programme

```
public class Server {
  public static void main (String[] args) {
    RequestReceiver rr = new RequestReceiver();
    RequestAnalyzer ra = new RequestAnalyzer();
    RequestDispatcher rd = new RequestDispatcher();
    FileRequestHandler frh = new FileRequestHandler();
    ErrorRequestHandler erh = new ErrorRequestHandler();
    Scheduler s = new MultiThreadScheduler();
    Logger l = new BasicLogger();
    rr.bindFc("rh", ra);
    rr.bindFc("s", s);
    ra.bindFc("rh", rd);
    ra.bindFc("l", l);
    rd.bindFc("h0", frh);
    rd.bindFc("h1", erh);
    rr.run();
  }
}
```

Ce programme crée une instance de chaque composant nécessaire et relie ces instances entre elles en utilisant les interfaces de contrôle `BindingController`

Puis il démarre l'application en appelant la méthode run() sur `RequestReceiver`

Inconvénients

- **Programmé "à la main" : risque d'erreurs**
- **Architecture d'ensemble non visible (hiérarchie)**
- **Mélange description d'architecture et déploiement**

Configuration du serveur Web (2)

Configuration utilisant un ADL *(Architecture Description Language)*
Un ADL est une description déclarative d'une architecture

Exemple 1 : définition de type pour RequestHandlers (File et Error) sans dépendance

```
<component-type name="HandlerType">
  <provides><interface-type name="rh" signature="comanche.RequestHandler"/></provides>
</component-type>
```

Exemple 2 : définition de type pour RequestDispatcher avec dépendance (extension du type ci-dessus)

```
<component-type name="DispatcherType" extends="HandlerType">
  <requires>
    <interface-type name="h" signature="comanche.RequestHandler" cardinality="collection"/>
  </requires>
</component-type>
```

Exemple 2 : définition d'un générateur d'instances (*template*) pour un composant simple (FileHandler)

```
<primitive-template name="FileHandler" implements="HandlerType">
  <primitive-content class="comanche.FileRequestHandler"/>
</primitive-template>
```


Configuration du serveur Web (3)

Configuration utilisant un ADL *(Architecture Description Language) - suite*
Construction d'un composant composite

Définit les liaisons avec les sous-composants

```
<composite-template name="Comanche" implements="RunnableType">
  <composite-content>
    <components>
      <component name="fe" type="FrontendType" implementation="Frontend"/>
      <component name="be" type="HandlerType" implementation="Backend"/>
    </components>
    <bindings>
      <binding client="this.r" server="fe.r"/>
      <binding client="fe.rh" server="be.rh"/>
    </bindings>
  </composite-content>
</composite-template>
```

Que faire avec une description en ADL ?

- **Compilation (fournit une classe Java analogue à la classe Server (déjà vue cf Flip 45)**
- **Interprétation (construit une instance du composant décrit)**

Il existe également un outil graphique qui construit une image de l'architecture globale

Reconfiguration dynamique (1)

Objectif : changer la configuration d'une application (par exemple par remplacement de composants) sans arrêter l'application

Méthode :

suspendre temporairement l'exécution de la partie modifiée
réaliser la modification
reprendre l'exécution

Problèmes :

limiter l'étendue de la partie suspendue
préserver la cohérence
ces problèmes ne sont pas entièrement résolus

Mise en œuvre :

Dans une interface de contrôle `LifeCycleController` spécifiée dans Fractal
Méthodes
`stopFc` : suspendre l'exécution des activités dans le composant
`startFc` : reprendre l'exécution suspendue par `stopFc`
`getFcState` : connaître l'état courant d'exécution du composant

Reconfiguration dynamique (2)

```
public class RequestDispatcher implements RequestHandler, BindingController, LifeCycleController
{
  private boolean started;
  private int counter;
  public String getFcState () {
    return started ? STARTED : STOPPED;
  }
  public synchronized void startFc () {
    started = true;
    notifyAll();
  }
  public synchronized void stopFc () {
    while (counter > 0) { try { wait(); } catch (InterruptedException _) {} }
    started = false;
  }
  public void handleRequest (Request r) throws IOException {
    synchronized (this) {
      while (counter == 0 && !started) { try { wait(); } catch (InterruptedException _) {} }
      ++counter;
    }
    try {
      // original code
    } finally {
      synchronized (this) {
        --counter;
        if (counter == 0) { notifyAll(); }
      }
    }
  }
  // rest of the class unchanged
```

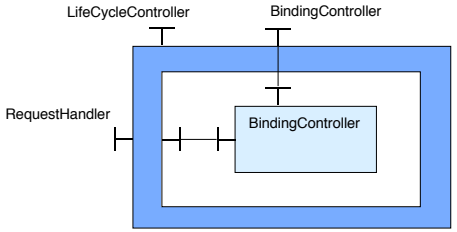
Une implémentation possible de startFc, stopFc, et getFcState

Reconfiguration dynamique (3)

```
public class Interceptor implements RequestHandler, LifeCycleController {
  public RequestHandler delegate;
  private boolean started;
  private int counter;
  public String getFcState () { /* as above */ }
  public synchronized void startFc () { /* as above */ }
  public synchronized void stopFc () { /* as above */ }
  public void handleRequest (Request r) throws IOException {
    synchronized (this) { /* as above */ }
    try {
      delegate.handleRequest(r);
    } finally {
      synchronized (this) { /* as above */ }
    }
  }
}
```

Une autre réalisation, qui sépare la gestion du cycle de vie de la partie fonctionnelle. L'intercepteur est interposé entre RequestAnalyzer et RequestDispatcher (inchangé)

On peut enfin imposer à l'intercepteur d'implémenter BindingController **par délégation**, en plus de LifeCycleManager
Le composant encapsulé est un RequestDispatcher comportant une interface BindingController mais pas de LifeCycleController
Cet intercepteur (qui joue le rôle de contrôleur) pourra être automatiquement engendré



Contrôleur d'un composant

Rôle du contrôleur d'un composant

- **Fournit les interfaces de contrôle pour la gestion du composant**
  - **Lecture/écriture des attributs**
  - **Cycle de vie (start, stop)**
  - **Configuration, reconfiguration (ajout, retrait de composants)**
  - **Liaison**
- **Rend visibles (exporte) certaines interfaces de composants inclus (fournies ou requises)**
- **Gère les relations du composant avec l'extérieur (interception)**
- **Fournit des interfaces internes (services) aux composants inclus**
- **Gère le partage de composants inclus**

Interfaces de contrôle (1)

```
interface AttributeController{}
        // interface vide : chaque composant doit en fournir un sous-type
        // exemple : "getters" et "setters" : fooType  getFoo(), setFoo(fooType foo)

interface BindingController{
    string[ ] listFc ();
    any lookupFc (string clientItfName)                    // return bound server interface (if any)
        throws NoSuchInterfaceException;
    void bindFc (string clientItfName, any serverItf)      // bind to server interface
        throws NoSuchInterfaceException, IllegalBindingException, IllegaLifeCycleException;
    void unbindFc (string clientItfName)                   // unbind the interface
        throws NoSuchInterfaceException ,IllegalBindingException, IllegaLifeCycleException;
}
```


Interfaces de contrôle (2)

```
interface ContentController{
    any[ ] getFcInternalInterfaces ();
    any getFcInternalInterface (string ItfName)
        throws NoSuchInterfaceException;
    Component[ ] getFcSubComponents ();                    // return list of subcomponents
    void addFcSubComponent (Component c)                   // add component c to contents
        throws IllegalContentException, IllegaLifeCycleException;
    void removeFcSubComponent (Component c)                // remove component c from contents
        throws IllegalContentException, IllegaLifeCycleException;
}

interface SuperController{
    Component[ ] getFcSuperComponents ();
}

interface LifeCycleController{
    string  getFcState ();
    void startFc () throws IllegaLifeCycleException;
    void stopFc () throws IllegaLifeCycleException;
}
```


Construction de composants

- **Deux mécanismes**
  - *Factory*
    - Générique (tout type de composants, paramétrée par le type)
    - Standard (un type de composants)
  - *Template*
    - Une sorte particulière de *Factory* standard
    - Crée des composants isomorphes au *Template* (mêmes interfaces fonctionnelles sauf *Factory*, mais interfaces de contrôle éventuellement différentes)
    - Un *Template* composite crée des composants composites de même structure interne
- **Amorçage**
  - Un composant *Bootstrap* doit être fourni (qui fonctionne comme une *Factory* générique)

Système de types de Fractal

Rappel : le type d'une entité est une assertion concernant cette entité, qui peut être vérifiée (statiquement ou dynamiquement). Sert à assurer le "bon usage" des entités.

- **Définitions**
  - Type d'un composant = ensemble des types de ses interfaces
  - Type d'une interface = signature + rôle (client ou serveur) + autres attributs
- **Deux attributs supplémentaires**
  - Contingence : spécifie une garantie sur la disponibilité de l'interface
    - *mandatory* : si interface serveur, doit être disponible ; si interface client, doit être liée à une interface serveur
    - *optional* : pas de garantie, ni d'obligation
  - Cardinalité : nombre d'interfaces de ce type pour un composant
    - *singleton* : exactement une interface de ce type
    - *collection* : nombre quelconque d'interfaces, créées sur demande
      - Convention de nommage : les noms de ces interfaces ont un préfixe commun

Interfaces du système de types

```
package org.objectweb.fractal.api.type;
interface ComponentType {
  InterfaceType[ ] getFcInterfaceTypes ();
  InterfaceType getFcInterfaceType (string itfName)
            throws NoSuchInterfaceException;
}
```

Introspection sur type de composant

```
interface InterfaceType {
  string getFcItfName ();
  string getFcItfSignature ();
  boolean isFcClientItf ();
  boolean isFcOptionalItf ();
  boolean isFcCollectionItf ();
}
```

Introspection sur type d'interface

```
interface TypeFactory {
  InterfaceType createFcItfType (string name, string signature, boolean isClient,
    boolean isOptional, boolean isCollection)
            throws InstantiationException;
  ComponentType createFcType (InterfaceType[] itfTypes)
            throws InstantiationException;
}
```

Création d'interface à partir des constituants élémentaires

Sous-typage

- **Motivation : la substitution**
  - Dans quelles conditions peut-on remplacer un composant A par un composant B (on dit que B est conforme à A) ?
  - Réponse intuitive : B doit fournir au moins autant que A et demander au plus autant
  - Réponse pour les interfaces : conformité = sous-typage
    - Un type d'interface I1 est sous-type d'un type d'interface serveur I2 ssi :
      - I1 a le même nom et le même rôle que I2
      - L'interface langage de I1 est sous-type de l'interface langage de I2 [fournit au moins autant]
      - Si contingence(I2) = mandatory, alors contingence(I1) = mandatory
      - Si card(I2) = collection, alors card(I1) = collection
    - Un type d'interface I1 est sous-type d'un type d'interface client I2 ssi :
      - I1 a le même nom et le même rôle que I2
      - L'interface langage de I1 est super-type de l'interface langage de I2 [requiert au plus autant]
      - Si contingence(I2) = optional, alors contingence(I1) = optional
      - Si card(I2) = collection, alors card(I1) = collection
  - Pour les composants
    - Un type de composant T1 est sous-type d'un type de composant T2 ssi
      - Tout type d'interface client de T1 est sous-type d'un type d'interface de T2
      - Tout type d'interface serveur de T2 est super-type d'un type d'interface de T1

ADL Fractal : composants

Description du composant externe Root

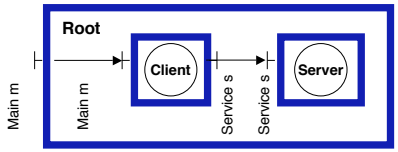
```
<component-type name="RootType">
  <provides>
    <interface-type name="m" signature="Main"/>
  </provides>
</component-type>
```

Description du composant Client

```
<component-type name="ClientType">
  <provides>
    <interface-type name="m" signature="Main"/>
  </provides>
  <requires>
    <interface-type name="s" signature="Service"/>
  </requires>
</component-type>
```

Une autre description du composant Client (par extension)

```
<component-type name="ClientType" extends="RootType">
  <requires>
    <interface-type name="s" signature="Service"/>
  </requires>
</component-type>
```



Description du composant Server

```
<component-type name="ServerType">
<provides>
<interface-type name="s" signature="Service"/>
</provides>
</component-type>
```


ADL Fractal : *Templates* primitifs

Template d'un composant (Client) encapsulant une classe Java

```
<primitive-template name="ClientImpl" implements="ClientType">
  <primitive-content class="ClientImpl"/>
</primitive-template>
```

Template d'un composant (Server) comportant des attributs

```
<primitive-template name="ServerImpl" implements="ServerType">
  <primitive-content class="ServerImpl"/>
  <controller>
    <attributes signature="ServiceAttributes">
      <attribute name="Header" value="-> "/>
      <attribute name="Count" value="1"/>
    </attributes>
  </controller>
</primitive-template>
```

ADL Fractal : *Templates* composites

```
<composite-template name="ClientServer" implements="RootType">
  <composite-content>
    <components>
      <component name="client" type="ClientType"/>
      <component name="server" type="ServerType"/>
    </components>
    <bindings>
      <binding client="this.m"   server="client.m"/>
      <binding client="client.s" server="server.s"/>
    </bindings>
  </composite-content>
</composite-template>
```

} **Spécification des liaisons**

Pour spécifier l'implémentation :

```
<composite-template name="ClientServerImpl" extends="ClientServer">
  <composite-content>
    <components>
      <component name="client" implementation="ClientImpl"/>
      <component name="server" implementation="ServerImpl"/>
    </components>
  </composite-content>
</composite-template>
```


Utilisation pratique de l'ADL

L'analyseur (*parser*) Fractal charge les définitions de types et de *templates* et crée les *templates* correspondant, ce qui permet d'instancier et de démarrer l'application.

```
Parser parser = Launcher.getBootstrapParser();                  // construit l'analyseur d'amorçage

Component tmpl = parser.loadTemplate("ClientServerImpl", true); // charge le template de l'application

Component comp = Fractal.getFactory(tmpl).newFcInstance();      // instancie le template
Fractal.getLifeCycleController(comp).startFc();                 // lance l'application
```

On peut aussi utiliser son propre analyseur à la place de l'analyseur d'amorçage. On commence par le charge et l'instancier avec l'analyseur d'amorçage, puis on l'utilise à la place de celui-ci.

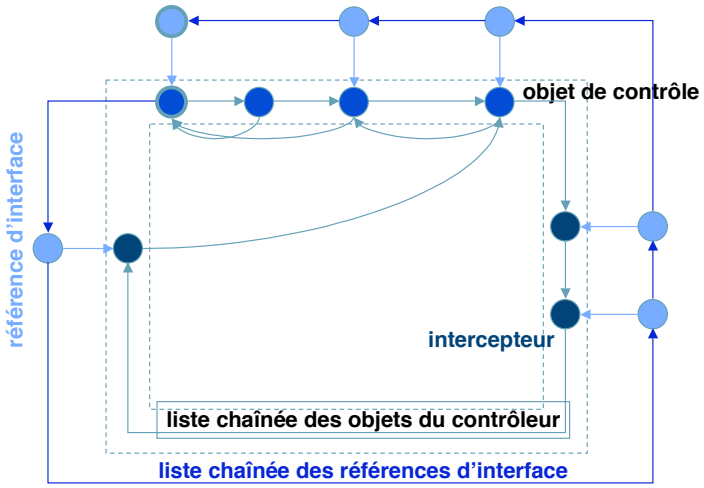
```
Parser parser = Launcher.getBootstrapParser();
Component parserTmpl = parser.loadTemplate("MyParser");
Component parserComp = Fractal.getFactory(parserTmpl).newFcInstance();
parser = (Parser)parserComp.getFcInterface("parser");

Component tmpl = parser.loadTemplate("ClientServerImpl");
Component comp = Fractal.getFactory(tmpl).newFcInstance();
Fractal.getLifeCycleController(comp).startFc();
```


Implémentation du modèle Fractal

- **Julia**
  - implémentation de référence du modèle Java typé
- **Objectifs**
  - réaliser un *framework* extensible pour programmer des contrôleurs
  - fournir des objets de contrôle variés :
    - offrant un *continuum* entre configuration statique et reconfiguration dynamique
    - pouvant être optimisés et désoptimisés dynamiquement
  - implanter les objets de contrôle de façon à minimiser, en priorité :
    - le surcoût en temps sur les applications,
    - le surcoût en mémoire sur les applications,
    - les coûts en temps et en mémoire des méthodes de l'API du modèle Java

Implémentation : structures de données

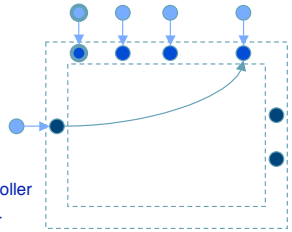


Implémentation : fichier de configuration

- **Définit les descripteurs de contrôleur Fractal**
  - Tels que : primitive, parametricPrimitive, composite…
  - Utilisés dans Factory.createFcTemplate(…)
- **Contient une liste de définitions**
  - Définition = (*name definition*)
  - Une définition peut référencer d'autres définitions :
    - (x (1 2 3)) # x est égal à (1 2 3)
    - (y (a b c 'x)) # y est égal à (a b c (1 2 3))

Implémentation : fichier de configuration

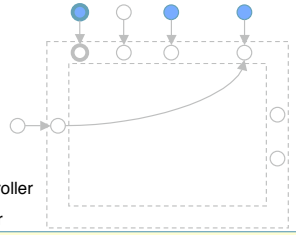
```
(primitive
 ( ('component-identity-itf
   'binding-controller-itf
   'lifecycle-controller-itf)
  'primitive-itf-ref-gen
  (org.objectweb.fractal.julia.BasicComponentIdentity
   org.objectweb.fractal.julia.control.content.BasicSubComponentController
   org.objectweb.fractal.julia.control.binding.ContainerBindingController
   org.objectweb.fractal.julia.control.lifecycle.GenericLifeCycleController)
  ( 'mixins )
  ( (org.objectweb.fractal.julia.asmgen.InterceptorClassGenerator
     org.objectweb.fractal.julia.asmgen.LifeCycleCodeGenerator
     org.objectweb.fractal.julia.asmgen.TraceCodeGenerator) )
  org.objectweb.fractal.julia.asmgen.MergeClassGenerator
  none
 ) )
```



Implémentation : fichier de configuration

types des interfaces de contrôle publiques

```
(primitive
 ( ('component-identity-itf
   'binding-controller-itf
   'lifecycle-controller-itf)
  'primitive-itf-ref-gen
  (org.objectweb.fractal.julia.BasicComponentIdentity
   org.objectweb.fractal.julia.control.content.BasicSubComponentController
   org.objectweb.fractal.julia.control.binding.ContainerBindingController
   org.objectweb.fractal.julia.control.lifecycle.GenericL
  ( 'mixins )
  ( (org.objectweb.fractal.julia.asmgen.InterceptorClas
     org.objectweb.fractal.julia.asmgen.LifeCycleCode
     org.objectweb.fractal.julia.asmgen.TraceCodeGe
  org.objectweb.fractal.julia.asmgen.MergeClassGene
  none
 ) )
```

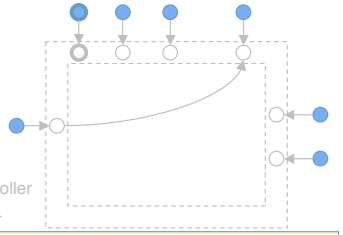


```
(component-identity-itf
 (org.objectweb.fractal.julia.type.BasicInterfaceType
  component-identity
  org.objectweb.fractal.api.ComponentIdentity
  server
  mandatory
  single
 ) )
```


Implémentation : fichier de configuration

générateur de classes pour les références d'interface

```
(primitive
 ( ('component-identity-itf
   'binding-controller-itf
   'lifecycle-controller-itf)
  'primitive-itf-ref-gen
  (org.objectweb.fractal.julia.BasicComponentIdentity
   org.objectweb.fractal.julia.control.content.BasicSubComponentController
   org.objectweb.fractal.julia.control.binding.ContainerBindingController
   org.objectweb.fractal.julia.control.lifecy
  ( 'mixins )
  ( (org.objectweb.fractal.julia.asmgen.In
     org.objectweb.fractal.julia.asmgen.Li
     org.objectweb.fractal.julia.asmgen.Tr
  org.objectweb.fractal.julia.asmgen.Merg
  none
 ) )
```

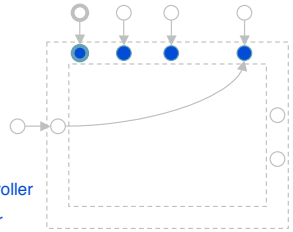


```
(primitive-itf-ref-gen
 (org.objectweb.fractal.julia.asmgen.InterfaceReferenceClassGenerator
  org.objectweb.fractal.julia.BasicInterfaceReference
  org.objectweb.fractal.julia.BasicInterfaceReference
  org.objectweb.fractal.julia.BasicInterfaceReference
  org.objectweb.fractal.julia.HiddenInterfaceReference
  externals
 ) )
```

Implémentation : fichier de configuration

```
(primitive
 ( ('component-identity-itf
    'binding-controller-itf
    'lifecycle-controller-itf)
   'primitive-itf-ref-gen
   (org.objectweb.fractal.julia.BasicComponentIdentity
    org.objectweb.fractal.julia.control.content.BasicSubComponentController
    org.objectweb.fractal.julia.control.binding.ContainerBindingController
    org.objectweb.fractal.julia.control.lifecycle.GenericLifeCycleController)
   ( 'mixins )
   ( (org.objectweb.fractal.julia.asmgen.InterceptorClassGenerator
      org.objectweb.fractal.julia.asmgen.LifeCycleCodeGenerator
      org.objectweb.fractal.julia.asmgen.TraceCodeGenerator) )
   org.objectweb.fractal.julia.asmgen.MergeClassGenerator
   none
 ) )
```

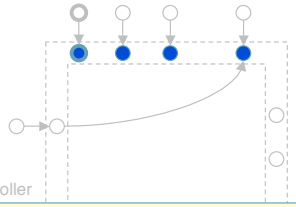
classes des objets de contrôle



Implémentation : fichier de configuration

```
(primitive
 ( ('component-identity-itf
    'binding-controller-itf
    'lifecycle-controller-itf)
   'primitive-itf-ref-gen
   (org.objectweb.fractal.julia.BasicComponentIdentity
    org.objectweb.fractal.julia.control.content.BasicSubComponentController
    org.objectweb.fractal.julia.control.bind
    org.objectweb.fractal.julia.control.lifec
   ( 'mixins )
   ( (org.objectweb.fractal.julia.asmgen.I
      org.objectweb.fractal.julia.asmgen.L
      org.objectweb.fractal.julia.asmgen.T
   org.objectweb.fractal.julia.asmgen.Me
   none
 ) )
```

classes mixins pour les aspects non orthogonaux

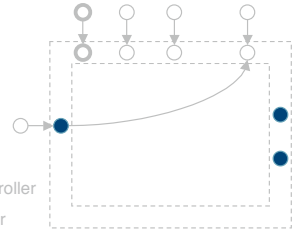


```
(mixins
 (org.objectweb.fractal.julia.asmgen.MixinClassGenerator
  org.objectweb.fractal.julia.control.attribute.AttributeTemplateMixin
  org.objectweb.fractal.julia.control.container.ContainerRootMixin
  org.objectweb.fractal.julia.control.binding.BindingMixin
  org.objectweb.fractal.julia.control.lifecycle.LifeCycleBindingMixin
  org.objectweb.fractal.julia.control.content.ContentBindingMixin
  org.objectweb.fractal.julia.control.lifecycle.LifeCycleContentMixin
 ) )
```


Implémentation : fichier de configuration

```
(primitive
 ( ('component-identity-itf
    'binding-controller-itf
    'lifecycle-controller-itf)
   'primitive-itf-ref-gen
   (org.objectweb.fractal.julia.BasicComponentIdentity
    org.objectweb.fractal.julia.control.content.BasicSubComponentController
    org.objectweb.fractal.julia.control.binding.ContainerBindingController
    org.objectweb.fractal.julia.control.lifecycle.GenericLifeCycleController)
   ( 'mixins )
   ( (org.objectweb.fractal.julia.asmgen.InterceptorClassGenerator
      org.objectweb.fractal.julia.asmgen.LifeCycleCodeGenerator
      org.objectweb.fractal.julia.asmgen.TraceCodeGenerator) )
   org.objectweb.fractal.julia.asmgen.MergeClassGenerator
   none
 ) )
```

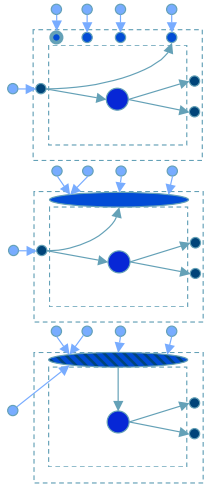
générateur de classes pour les intercepteurs



Implémentation : fichier de configuration

```
(primitive
 ( ('component-identity-itf
    'binding-controller-itf
    'lifecycle-controller-itf)
   'primitive-itf-ref-gen
   (org.objectweb.fractal.julia.BasicComponentIdentity
    org.objectweb.fractal.julia.control.content.BasicSubComponentController
    org.objectweb.fractal.julia.control.binding.ContainerBindingController
    org.objectweb.fractal.julia.control.lifecycle.GenericLifeCycleController)
   ( 'mixins )
   ( (org.objectweb.fractal.julia.asmgen.InterceptorClassGenerator
      org.objectweb.fractal.julia.asmgen.LifeCycleCodeGenerator
      org.objectweb.fractal.julia.asmgen.TraceCodeGenerator) )
   org.objectweb.fractal.julia.asmgen.MergeClassGenerator
   none
 ) )
```

options d'optimisation



Conclusion : apports d'usage attendus

- **Facilité de développement et d'intégration des plates-formes**
  - En développement
  - Configuration au déploiement
  - Administration et reconfiguration en production
- **Productivité des développeurs & intégrateurs**
  - Simplicité
  - Méthodes standards
  - Outillage standard
- **Pérennité des plate-formes**
  - Adaptabilité & réutilisabilité
  - Maintenabilité & évolutivité
  - Support des standards par déclinaison

Conclusion : état actuel

- **Principes architecturaux pour l'ingénierie des systèmes**
  - ✓ Construction par composition de briques logicielles : les composants
  - ✓ Vision homogène de la topologie des systèmes logiciels
    - ❖ applications, intergiciel, systèmes d'exploitation
      - ▲ exemples : FractalGUI, Speedo, Jabyce, THINK
  - × Gestion uniforme des ressources, des activités et des domaines de contrôle
  - ✓ Couverture du cycle de vie complet :
    - ❖ développement, déploiement (uniquement centralisé), supervision (API)
- **Réalisation d'un support d'exécution pour composants**
  - Support d'exécution pouvant être
    - ✓ spécialisé, outillé, étendu
    - × composé avec d'autres canevas : persistance, réplication,sécurité, …

Conclusion : perspectives

- **Administration : JMX**
- **Déploiement distribué**
- **Outils multi-cibles (Java ou C)**
- **Evaluation : application à ObjectWeb**
- **Recherches en cours**
  - FTR&D : comportements (spécification/observation TLA, composition d'automates), événements/réactions (composition, auto-adaptation…), domaines de sécurité, personnalité EJB
  - INRIA : optimisations, transactions, formalisation
  - EMN : auto-adaptation, ADL
  - I3S : contrats, assertions
  - Université Valenciennes : propriétés non fonctionnelles

Annexe 1 : Julia séparation des aspects

classe de base

```
class BasicBindingController {
  // …
  String bindFc (…) {
    // … NO lifecycle related code …
  }
}
```

classe mixin

```
abstract class LifeCycleBindingMixin {
  String bindFc (…) {
    if (getFcState() != STOPPED)
      throw new Exception();
    return _super_bindFc(…);
  }
  abstract String _super_bindFc (…);
}
```

classe mélangée, générée dynamiquement

```
class XYZ extends BasicBindingController {
  String bindFc (…) {
    if (getFcState() != STOPPED)
      throw new Exception();
    return super.bindFc(…);
  }
}
```

classe mixin : classe dont la super classe est spécifiée par les champs et les méthodes qu'elle doit avoir

Annexe 1 : Julia
séparation des aspects

- **InterceptorClassGenerator**
  - **Arguments :**
    - un nom d'interface Java
    - un ou plusieurs générateurs de code
  - **Génère une classe d'intercepteur :**
    - qui implante l'interface spécifiée
    - dont les méthodes sont générées par les générateurs de code
- **Chaque générateur de code :**
  - **Transforme le bytecode produit par le générateur suivant; toutes les transformations possibles sont autorisées.**
  - **Est indépendent des autres, mais leur ordre est important**

Annexe 1 : Julia
séparation des aspects

- **TraceCodeGenerator seul (simplifié) :**

```
void m () {                    void m () {
  delegate.m();                  System.err.println("entering m()");
}                                delegate.m();
                                 System.err.println("leaving m()");
                               }
```

- **LifeCycleCodeGenerator seul (simplifié) :**

```
void m () {                    void m () {
  delegate.m();                  lc.incrementInvocationCounter();
}                                try {
                                   delegate.m();
                                 } finally {
                                   lc.decrementInvocationCounter();
                                 }
                               }
```


Annexe 1 : Julia
séparation des aspects

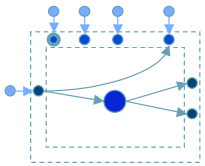
- **LifeCycleCodeGenerator + TraceCodeGenerator :**

```
void m () {
  lc.incrementInvocationCounter();
  try {
    System.err.println("entering m()");
    delegate.m();
    System.err.println("leaving m()");
  } finally {
    lc.decrementInvocationCounter();
  }
}
```

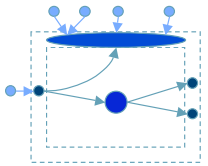
ou

```
void m () {
  System.err.println("entering m()");
  lc.incrementInvocationCounter();
  try {
    delegate.m();
  } finally {
    lc.decrementInvocationCounter();
  }
  System.err.println("leaving m()");
}
```

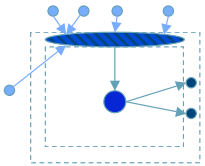

Annexe 1 : Julia
optimisation des contrôleurs



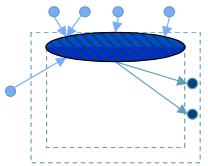
pas d'optimisation



fusion des objets de contrôle

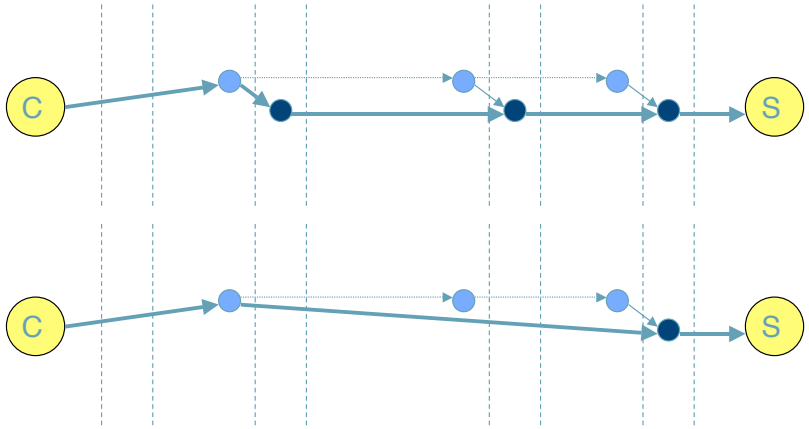


fusion des objets de contrôle et des intercepteurs



fusion des objets de contrôle, des intercepteurs et du contenu

Annexe 1 : Julia
optimisation des liaisons



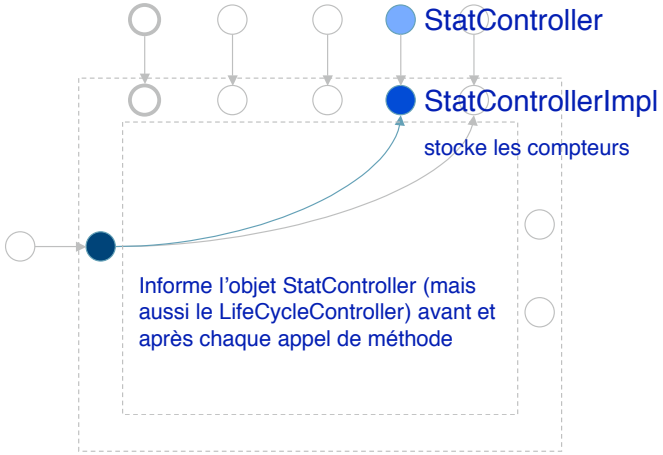
Liaisons raccourcies si intercepteurs absents

Annexe 2 : extension de Julia

- **Exemple :**
  - **Ajout d'une interface pour collecter des statistiques :**

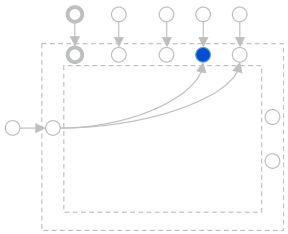
```
public interface StatController {
  String STAT_CONTROLLER = "stat-controller";
  int getNumberOfMethodCall ();
  int getNumberOfMethodSuccess ();
  long getTotalExecutionTime ();
}
```


Annexe 2 : extension de Julia



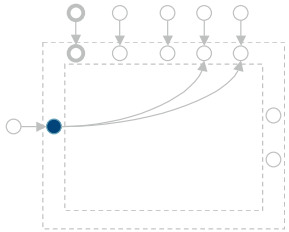
Annexe 2 : extension de Julia

```
public class StatControllerImpl extends BasicController implements StatController {
  private int calls;
  private int success;
  private long totalTime;
  // implementation of the StatController interface
  public int getNumberOfMethodCall () { return calls; }
  public int getNumberOfMethodSuccess () { return success; }
  public long getTotalExecutionTime () { return totalTime; }
  // methods called by the associated interceptor
  public synchronized long statPreMethod (final String method) {
     ++calls; return System.currentTimeMillis();
  }
  public synchronized void statPostMethod (final String method, final long start) {
     ++success; totalTime += System.currentTimeMillis() - start;
  }
}
```



Annexe 2 : extension de Julia

```
public class StatCodeGenerator extends SimpleCodeGenerator {
  protected String getControllerInterfaceName () {
    return StatController.STAT_CONTROLLER;
  }
  protected String getControllerClass () {
    return "stat.StatControllerImpl";
  }
  protected String getPreMethodName () {
    return "statPreMethod";
  }
  protected String getPostMethodName () {
    return "statPostMethod";
  }
  protected Class getContextType () { return Long.TYPE; }
  protected String getMethodName (final Method m) { return m.getName(); }
}
```



Annexe 2 : extension de Julia

```
(statPrimitive
  ( ('component-identity-itf
     'binding-controller-itf
     'lifecycle-controller-itf
     'stat-controller-itf)
    'primitive-itf-ref-gen
    (org.objectweb.fractal.julia.BasicComponentIdentity
     org.objectweb.fractal.julia.control.content.BasicSubComponentController
     org.objectweb.fractal.julia.control.binding.ContainerBindingController
     org.objectweb.fractal.julia.control.lifecycle.GenericLifeCycleController
     stat.StatControllerImpl)
    ( 'mixins )
    ( (org.objectweb.fractal.julia.asmgen.InterceptorClassGenerator
       org.objectweb.fractal.julia.asmgen.LifeCycleCodeGenerator
       stat.StatCodeGenerator) )
    org.objectweb.fractal.julia.asmgen.MergeClassGenerator none ) )
```

```
void m () {
  lc.incrementInvocationCounter();
  try {
    long time = sc.statPreMethod("m");
    delegate.m();
    sc.statPostMethod("m", time);
  } finally {
    lc.decrementInvocationCounter();
  }
}
```

Fractal vs …

- **Fractal vs EJB et CCM**
  - Modèle hiérarchique, partage, introspection
  - Pas de granularité, de types de composants fixés à priori
  - Propriétés non fonctionnelles : très peu mais extensible
- **Fractal vs OSGi**
  - Modèle hiérarchique, partage, introspection/contrôle
  - Propriétés non fonctionnelles
  - « Service oriented » possible mais pas imposé
- **Fractal vs Avalon**
  - Partage, API minimale