ICAR'03

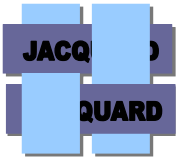
École d'été sur les Intergiciels et sur la Construction d'Applications Réparties

Le modèle de composants CORBA

Philippe Merle

Projet Jacquard (INRIA et LIFL)

http://www.lifl.fr/~merle



ObjectWeb Open Source Middleware

Plan

INRIA RHÔNE-ALPES

- Quelques rappels sur CORBA
  - Objectifs, historique, principes de fonctionnement, principaux défauts
- Introduction au modèle de composants CORBA
  - Points clés, comparaisons avec EJB, COM et .NET
  - Le cadre global de construction d'applications CCM
- Le modèle abstrait de composants CORBA
  - Concepts et nouvelles constructions du langage OMG IDL 3.0
- Le conditionnement, l'assemblage et le déploiement CORBA
  - Les archives de composant et d'assemblage
  - Le processus de déploiement et ses interfaces
- Les conteneurs CORBA
  - Politiques de gestion des services
- Conclusions
  - Implantations CCM et activités OMG en cours autour du CCM

ObjectWeb
Open Source Middleware

INRIA
RHÔNE-ALPES

Quelques rappels sur CORBA

CORBA

- **Spécification d'un modèle de composants logiciels répartis et des interfaces de l'intergiciel associé**
  - CORBA pour Common Object Request Broker Architecture
- **Fait parti d'un large ensemble de spécifications définies par le consortium international Object Management Group (OMG)**
  - Object Management Architecture (OMA)
    - Des services communs
      - Nommage, courtage, notification, transactions, persistance, sécurité, …
    - Des interfaces orientées métiers / domaines
      - Air Traffic Control, Gene Expression, Software Radio, Workflow Management, …
  - Model Driven Architecture (MDA)
    - Des technologies d'ingénierie dirigée par les modèles
      - Unifed Modeling Language (UML)
      - Meta Object Facilities (MOF)
      - XML Metadata Interchange (XMI)
      - …

Objectifs de CORBA

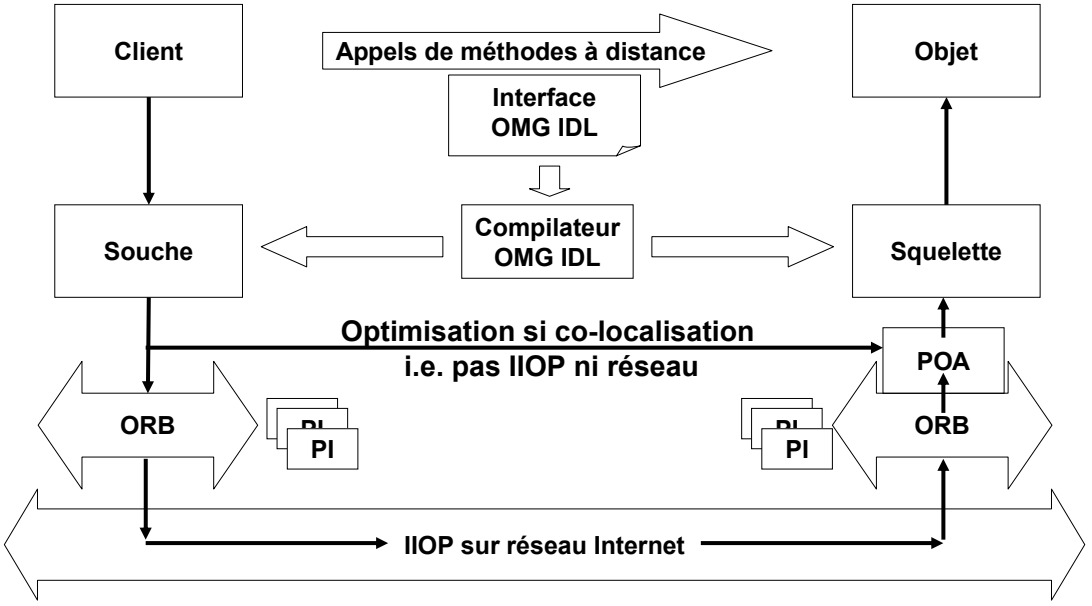
- **Ouvert**
  - Spécification indépendant des fournisseurs et des implantations
- **Hétérogénéité**
  - Multi-langages, multi-OSs, multi-réseaux, multi-fournisseurs
  - Via OMG Interface Definition Language (OMG IDL)
- **Portabilité**
  - Code applicatif indépendant des implantations CORBA
  - Projections standardisées OMG IDL vers langages de programmation
- **Interopérabilité**
  - Protocole réseau commun entre implantations CORBA
    - General Inter-ORB Protocol (GIOP)
    - Internet Inter-ORB Protocol (IIOP)
  - Vers d'autres modèles comme OLE, COM et EJB

Historique de CORBA

- **CORBA 1.0 (1991)**
  - Modèle orienté objet, OMG IDL et interfaces de l'intergiciel
- **CORBA 2.0 (1995)**
  - C++, interopérabilité GIOP/IIOP et COM
  - v2.1 (1997) : COBOL et ADA
  - v2.2 (1998) : Java et Portable Object Adaptor (POA)
  - v2.3 (1999) : révisions techniques
  - v2.4 (2000) : Asynch. Messaging, Minimum CORBA, RT CORBA
  - v2.5 (2001) : Intercepteurs portables, FT CORBA
  - v2.6 (2001) : Sécurité interopérable
- **CORBA 3.0 (2002)**
  - Composants, langages de script (e.g. OMG IDLscript)

Intergiciel CORBA

Transparence vis-à-vis de la distribution et de l'hétérogénéité



Intergiciel CORBA

- **Divers modèles d'invocations de méthodes**
  - Static Invocation Interface (SII)
    - Contrôle du typage des invocations à la compilation
    - Projection OMG IDL vers souches / squelettes
  - Dynamic Invocation Interface (DII) + Dynamic Skeleton Interface (DSI)
    - Contrôle du typage des invocations à l'exécution
    - Souche et squelette génériques
    - Découverte OMG IDL via Interface Repository
  - Asynchronous Method Invocation (AMI)
    - Appel de méthodes asynchrones
    - Résultats obtenues par scrutation périodique ou appel retour
  - Les trois approches sont compatibles et composables
- **Profils spécialisés**
  - Minimum
  - Real-Time
  - Fault-Tolerance

Langage OMG IDL

- **Types de données**
  - Elémentaires : octet, booléen, entiers, flottants, caractères, chaînes
  - Construits : typedef, énumération, structure, union, séquence, tableau
  - Polymorphes : Any et TypeCode
  - Passage par valeur
- **Exceptions**
  - Composées de champs publiques (~structures)
- **Interfaces**
  - Opérations + attributs
  - Héritage multiple
  - Passage par référence
- **Types valeurs**
  - ~ interface + structure
  - Héritage simple
  - Passage par valeur
- **Standard ISO 14750**

Projections OMG IDL

- **Règles de traduction des constructions OMG IDL en constructions d'un langage cible, standardisées pour**
  - ADA
  - C
  - C++
  - COBOL
  - CORBA Scripting Language (IDLscript)
  - Java ←→ OMG IDL
  - Lisp
  - PL/1
  - Python
  - SmallTalk
  - XML DTD/Schema → Value types
- **Non standardisées mais existantes pour Eiffel, TCL, …**

Quelques implantations de CORBA

- **Commerciales**
  - IONA ORBacus et Orbix pour C++ et Java
  - Borland VisiBroker pour C++ et Java
  - SUN Java Development Kit
  - Plates-formes J2EE
    - BEA WebLogic Enterprise
    - IBM WebSphere
  - Navigateur Netscape
  - …
- **Open source**
  - Red Hat ORBit pour C
  - MICO pour C++
  - AT&T omniORB pour C++
  - TAO pour C++
  - OpenORB pour Java
  - JacORB pour Java
  - Jonathan pour Java
  - …

Principaux défauts de CORBA 2.x

- **Non trivial à mettre en œuvre**
  - Connaître règles de projections OMG IDL vers langages de programmation
- **Pas de vision des architectures logicielles**
  - Liaisons entre objets CORBA cachées dans le code
  - Rarement configurables de l'extérieur, i.e. par architectes
  - Aucun moyen pour raisonner sur des compositions d'objets CORBA
- **Pas de séparation des aspects fonctionnels / non fonctionnels**
  - Programmation explicite des aspects non fonctionnels dans code applicatif, e.g.
    - POA, cycle de vie, (dé)activation, nommage, courtage, notification, persistance, transactions, sécurité, temps réel, tolérances aux pannes, etc.
  - Complexité accrue pour les experts métiers
- **Pas de standard pour conditionnement et déploiement**
  - Uniquement des solutions ad hoc

Solutions apportées par le modèle de composants CORBA

- **Vision des architectures logicielles**
  - Description explicite des liaisons entre composants CORBA
    - Liaisons entre types décrites via des ports en OMG IDL
    - Liaisons entre instances décrites via descripteurs XML d'assemblage
- **Meilleur séparation des aspects fonctionnels / non fonctionnels**
  - Dichotomie composant / conteneur
    - Composant contient code métier
    - Conteneur contient code technique
  - Politiques techniques décrites en XML plutôt que programmées en dur
- **Standard pour le conditionnement, assemblage et déploiement**
  - Formats d'archives et de descripteurs XML
  - API de contrôle du déploiement en réparti
- **Mais toujours pas trivial à mettre en œuvre ;-(**
  - OMG IDL composant → OMG IDL objet → langages de programmation

ObjectWeb
Open Source Middleware

INRIA
RHÔNE-ALPES

Introduction au modèle de composants CORBA

Le modèle de composants CORBA

- **Un modèle de composants répartis**
  - Multi-langages, multi-OSs, multi-ORBs, multi-fournisseurs, etc.
    - Vs le modèle EJB centré sur le langage Java
    - Vs le canevas .NET centré sur le fournisseur Microsoft
  - Un langage de définition des composants et de leurs ports d'interaction
    - Des composants clients (IHM) aux composants serveurs (métiers)
  - Une technologie XML de conditionnement et d'assemblage de composants
  - Une infrastructure de déploiement réparti
  - Un canevas de conteneurs gérant les politiques de
    - Cycle de vie, d'activation, de sécurité, de transactions, de persistance et de communication asynchrone
  - Interopérabilité avec les Enterprise Java Beans (EJB 1.1)
  - Méta modèle MOF pour l'ingénierie dirigée par les modèles (MDE / MDA)

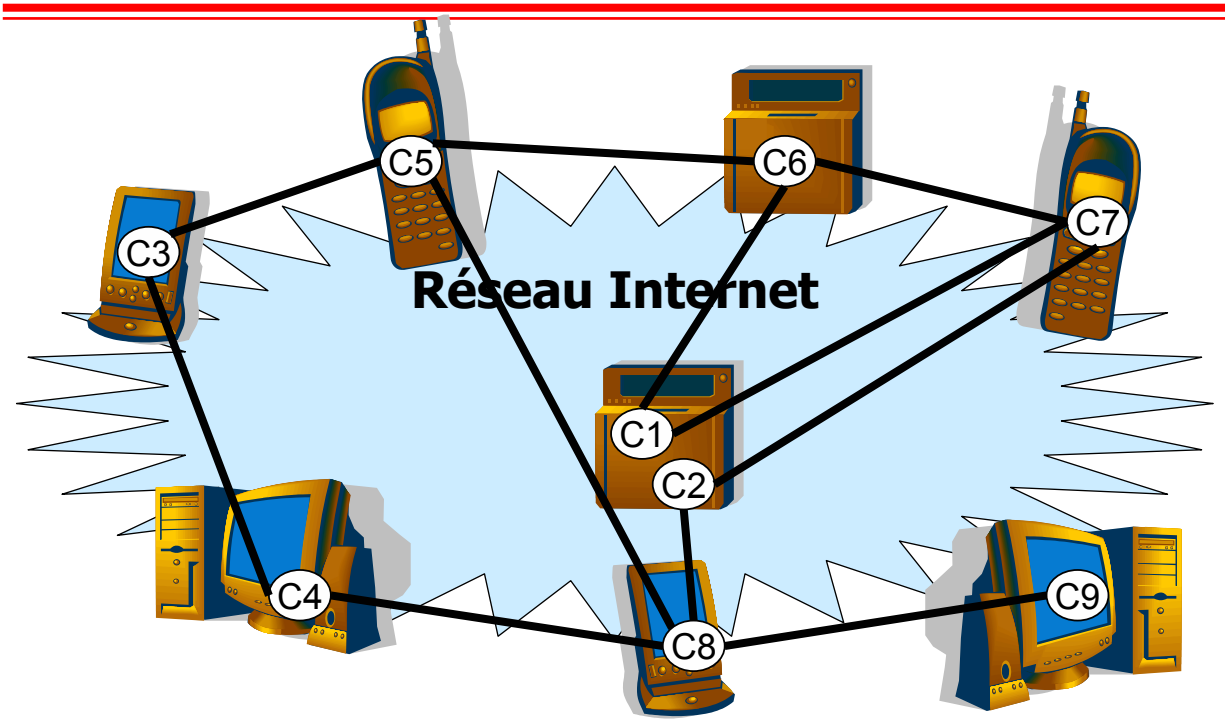
Similarités avec EJB, COM et .NET

- **Comme SUN Microsystems's Enterprise Java Beans (EJB)**
  - Composants CORBA créés et gérés par des maisons (homes)
  - S'exécutant dans des conteneurs prenant en charge les services systèmes
  - Hébergés par des serveurs d'applications
- **Comme Microsoft's Component Object Model (COM)**
  - Composants CORBA ont plusieurs interfaces offertes et requises
    - Accessibles par appel de méthodes ou par notification d'événements
  - Navigation et introspection des interfaces
- **Comme Microsoft's .NET Framework**
  - Support de différents langages de programmation
  - Technologie de conditionnement pour déploiement

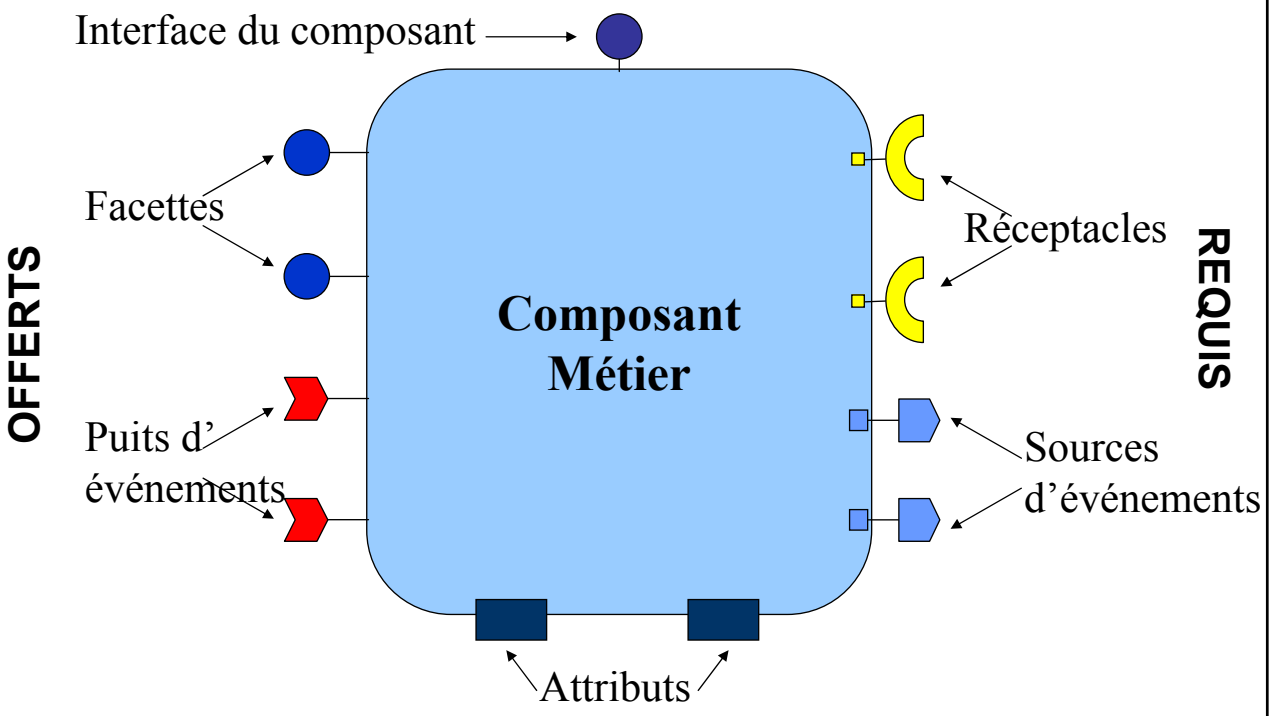
Différences avec EJB, COM et .NET

- **CCM réunit simultanément les "bonnes" fonctionnalités des EJB, de COM et de .NET**
  - +++ ouvert +++ hétérogénéité +++ portabilité +++ interopérabilité +++
- **CCM fournit un cadre global pour la construction d'applications à base de composants distribués**
  - Spécification, implantation, conditionnement, assemblage, déploiement et exécution des composants
- **Répartition des assemblages de composants CORBA**
  - Peuvent être déployés et s'exécutés sur différentes machines
- **Segmentation de l'implantation des composants**
  - Plusieurs classes au lieu d'une seule
  - Un état persistant par segment possible

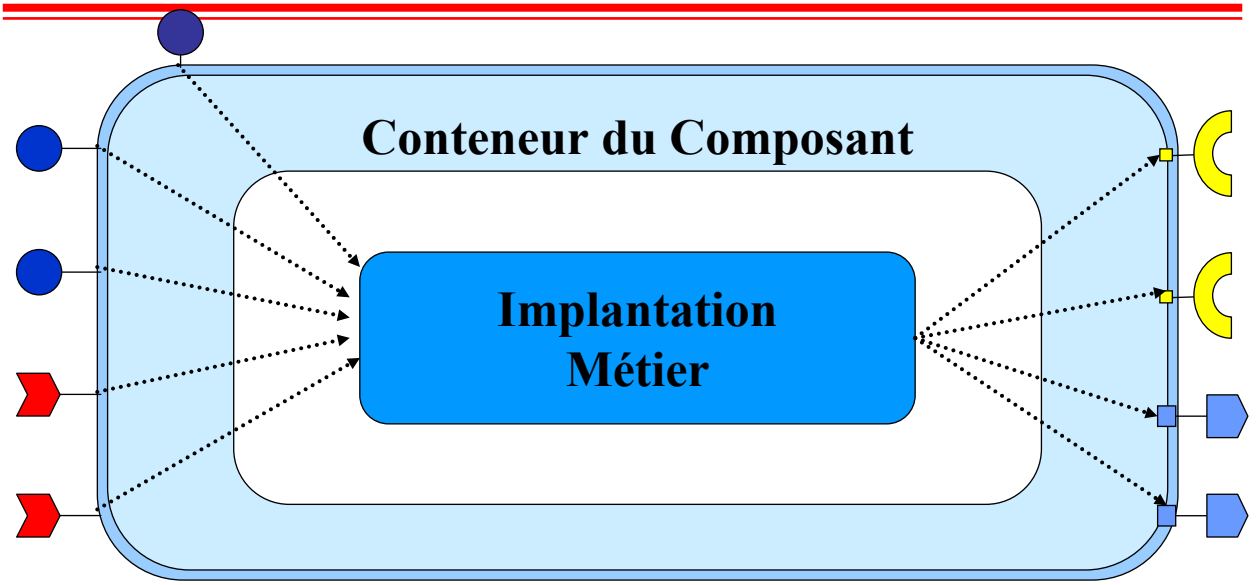
Une application CCM



Un composant CORBA et ses ports

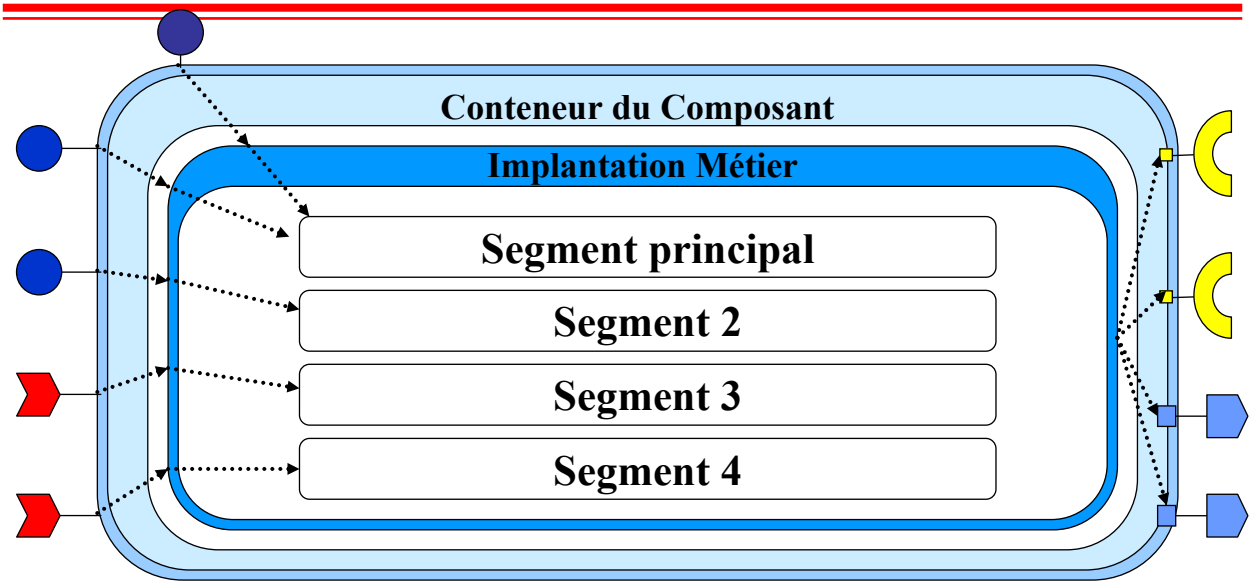


Conteneur et implantation d'un composant CORBA



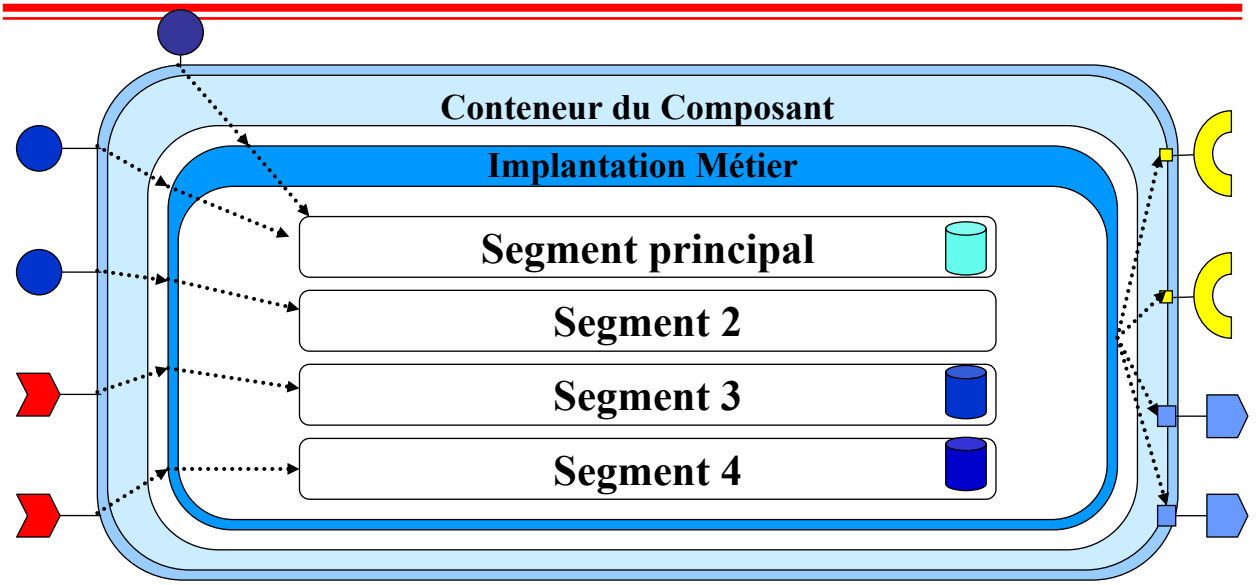
- **Par interposition, le conteneur gère**
  - Le cycle de vie, i.e. Service, Session, Process, Entity
  - Les ports, l'activation, la persistance, les transactions, la sécurité, les communications

Implantation segmentée d'un composant CORBA



- **Implantation de plusieurs ports offerts ayant la même interface**
  - Impossible si le composant est implanté par une unique classe
  - Equivalent aux "component parts " dans UML 2.0

Segmentation de la persistance d'un composant CORBA



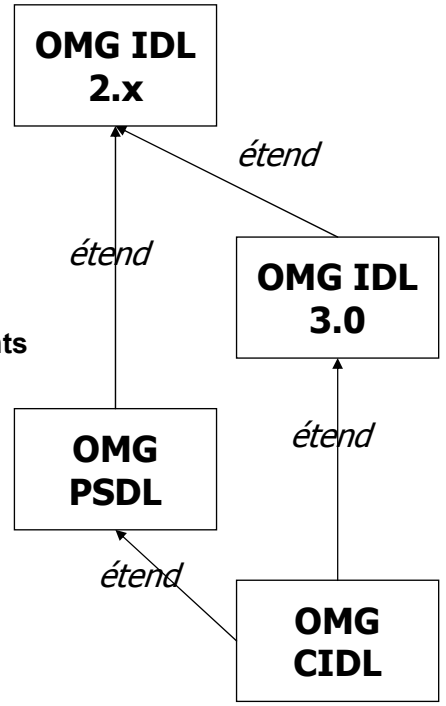
- **Contrôle fin de la persistance métier**
  - Chaque segment peut avoir son propre état persistant stocké sur un support de données distinct

CCM : un cadre global pour la construction d'applications

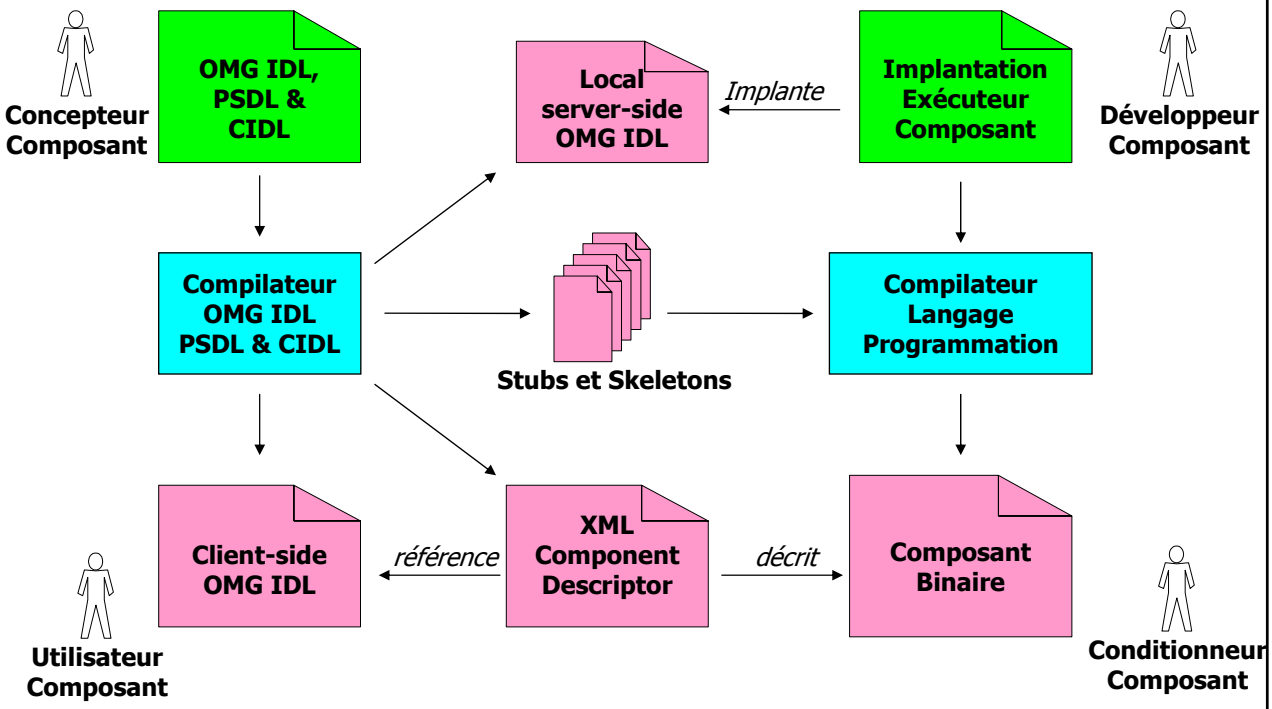
- **Spécification des composants**
  - Via OMG Interface Definition Language (OMG IDL)
    - Description des composants, de leurs ports et des maisons
- **Implantation des composants**
  - Via OMG Component Implementation Definition Language (OMG CIDL)
    - Description du cycle de vie, de la persistance et de la segmentation des composants
  - Via Component Implementation Framework (CIF)
    - Règles de programmation à respecter
    - Interfaces conteneur → composant & composant → conteneur
- **Conditionnement des composants**
  - Via archives ZIP incluant des descripteurs XML + code binaire
- **Assemblage des composants**
  - Via le XML Component Assembly Descriptor (CAD) et des archives ZIP
- **Déploiement des composants**
  - Via une infrastructure répartie d'interprétation des descripteurs XML
- **Exécution des composants**
  - Via des conteneurs hébergés par des serveurs d'applications

Les langages de définition du CCM

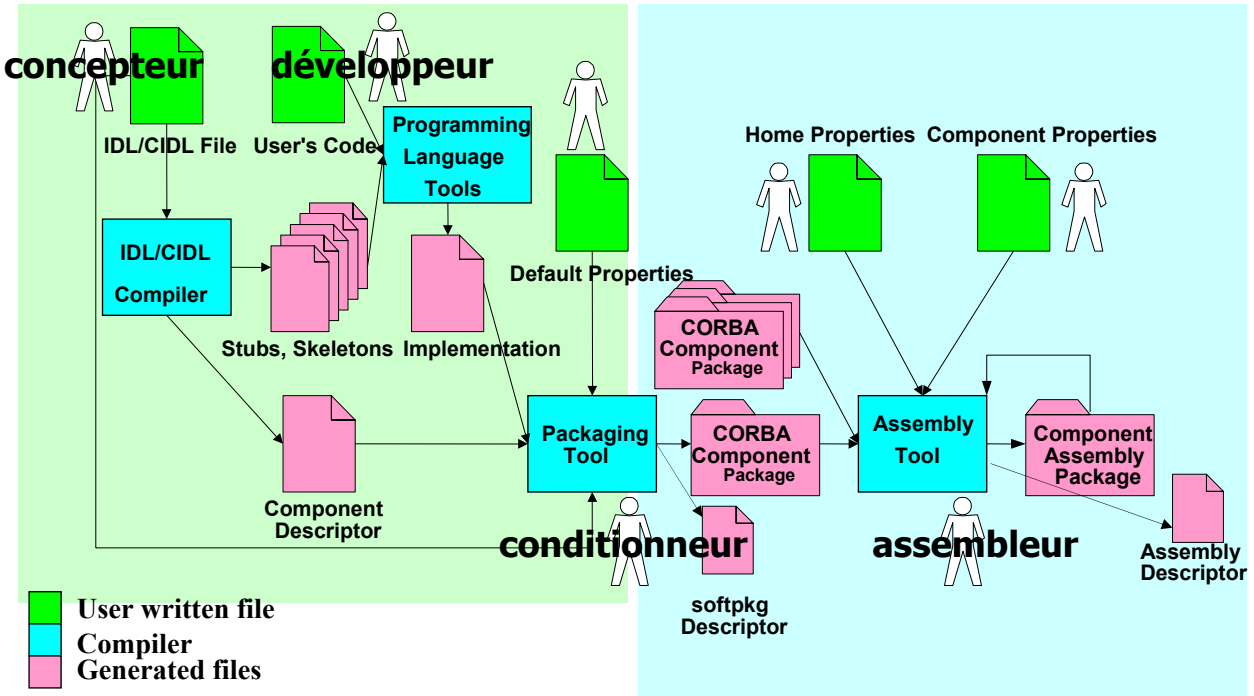
- **OMG IDL 2.x**
  - Modèle orienté objet
  - i.e. types de données, exceptions, interfaces et types valeurs
- **OMG IDL 3.0**
  - Modèle orienté composant
  - i.e. types de composants, maisons et événements
- **OMG PSDL**
  - Définition d'états persistents
  - i.e. types et maisons [abstraites] de stockage
- **OMG CIDL**
  - Description de l'implantation de composants
  - i.e. compositions et segments



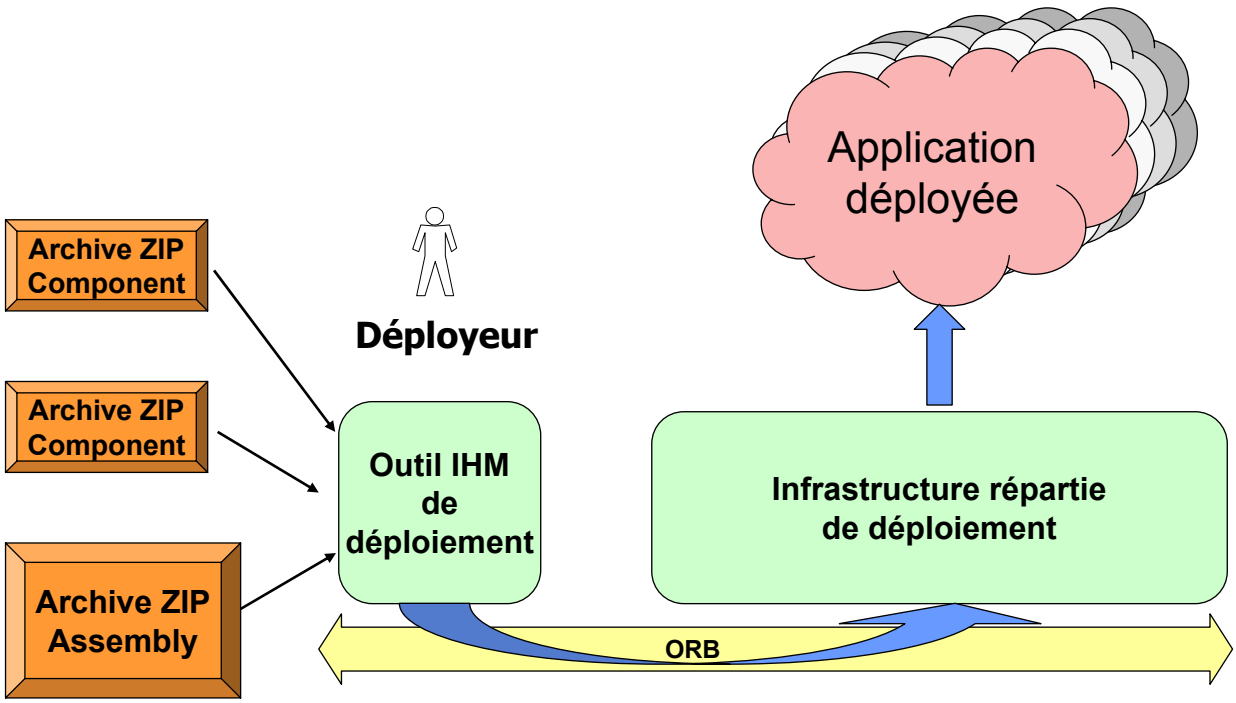
De la conception au conditionnement des composants CORBA



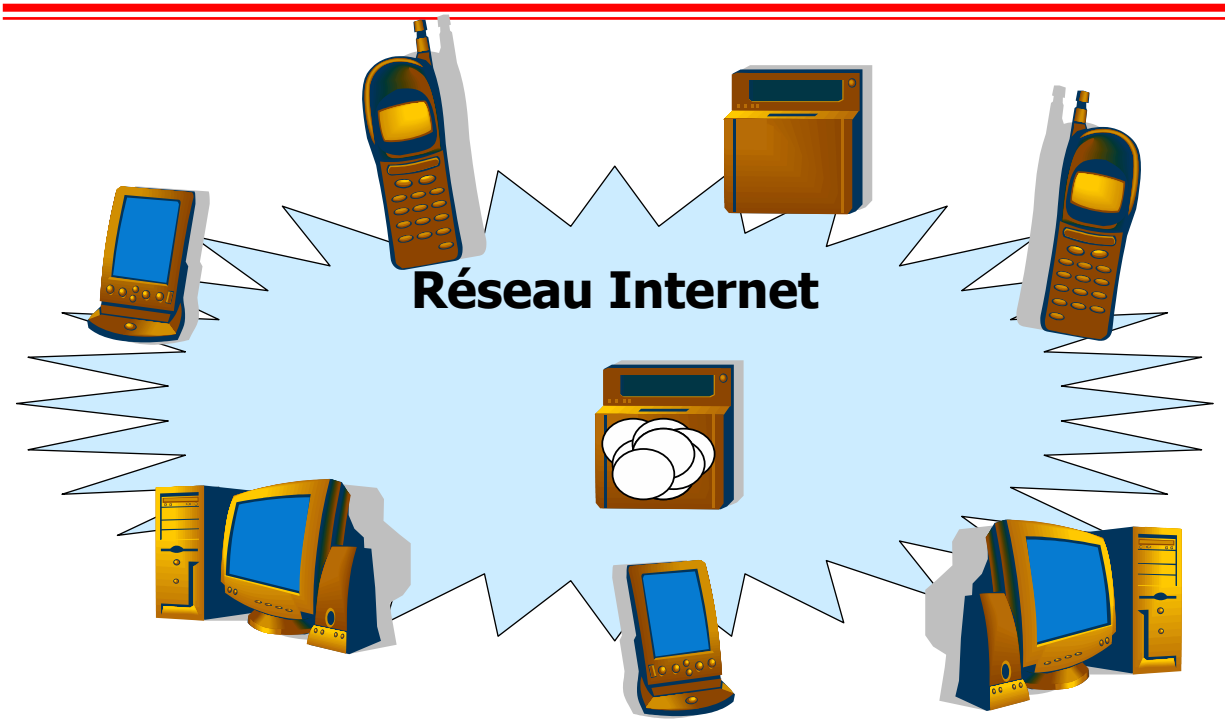
Conception, développement, conditionnement et assemblage



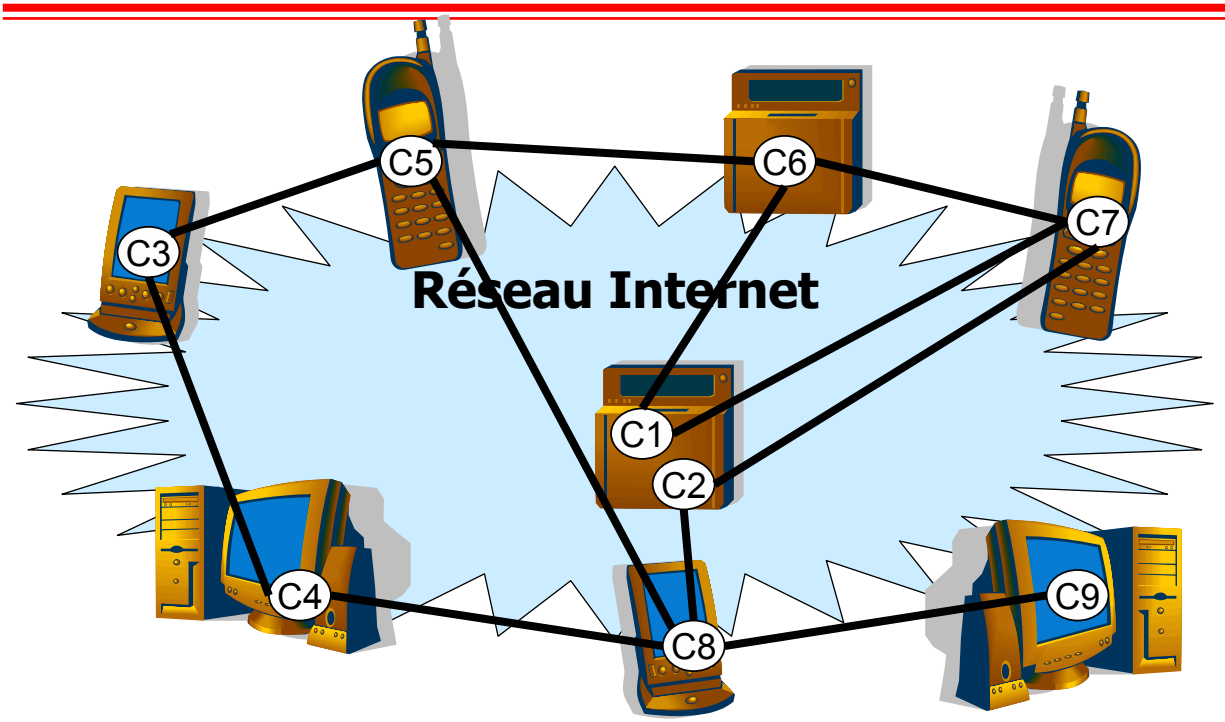
Le déploiement d'assemblages CCM



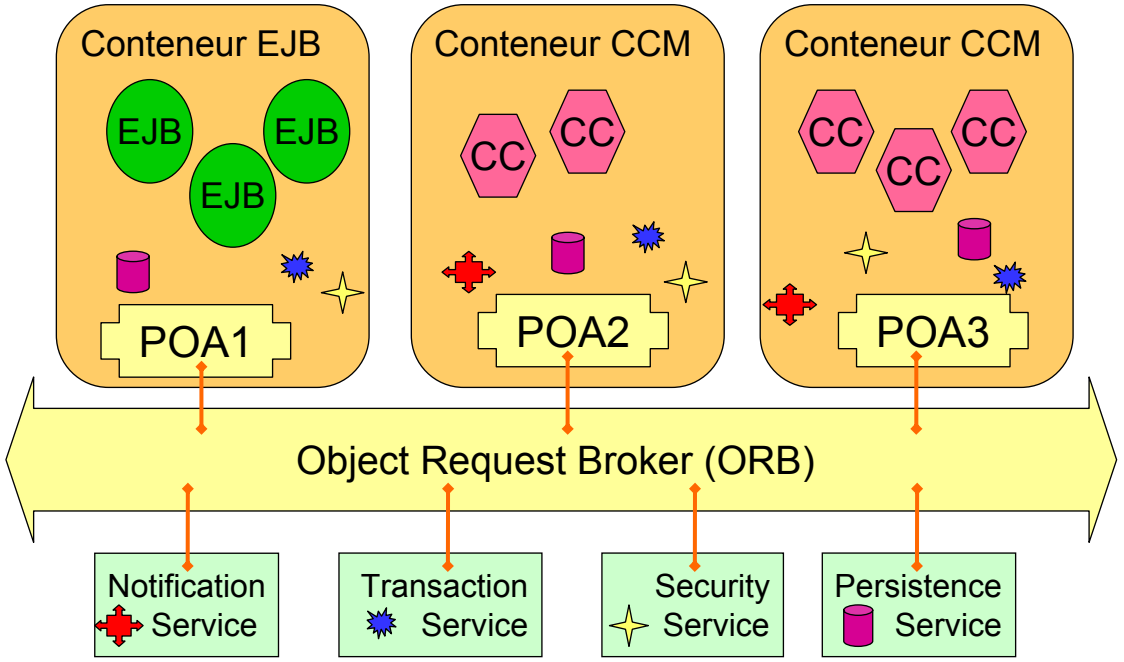
Le déploiement d'une application CCM



Le déploiement d'une application CCM



Conteneurs CORBA, ORB et services



La spécification CORBA Components

- Le modèle de composants CORBA
- Component Implementation Definition Language (CIDL)
- Component Implementation Framework (CIF)
- Component Container Programming Model
- Interopérabilité avec EJB 1.1
- Conditionnement et déploiement
- Un méta modèle MOF pour OMG IDL et CIDL

=> Spécification OMG disponible ~ 500 pages

ObjectWeb
Open Source Middleware

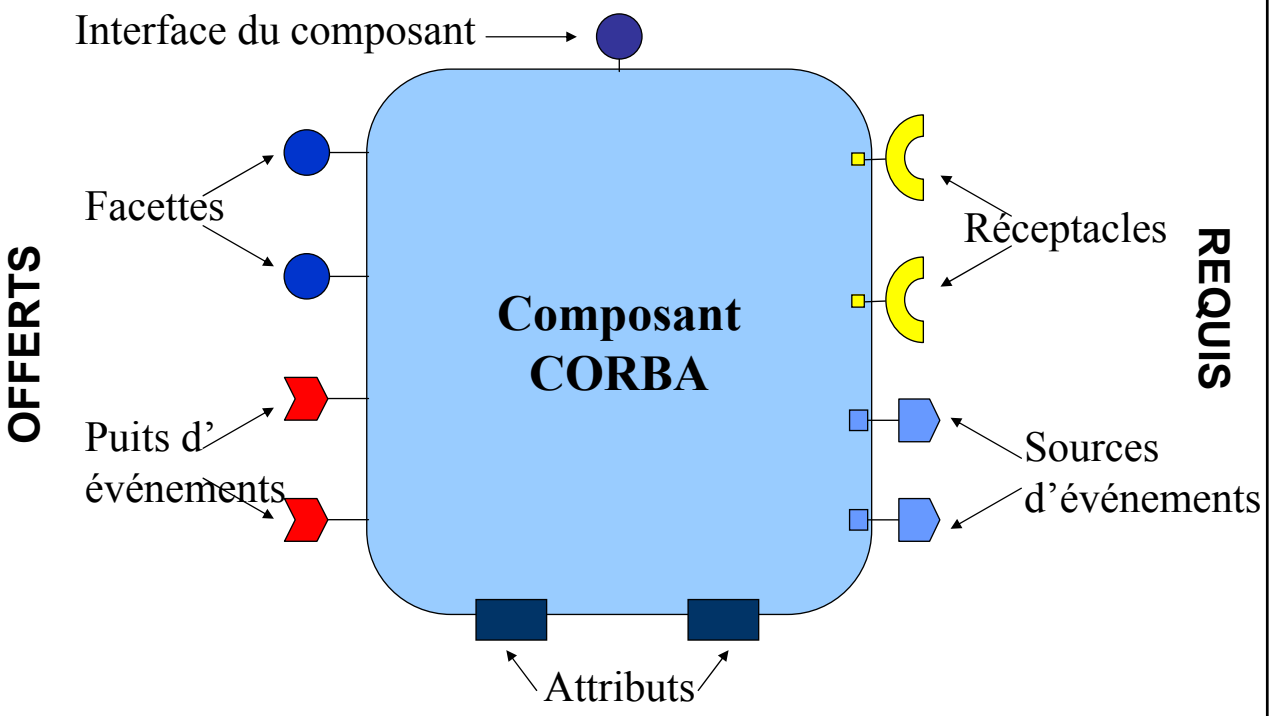
INRIA
RHÔNE-ALPES

Le modèle abstrait de composants CORBA

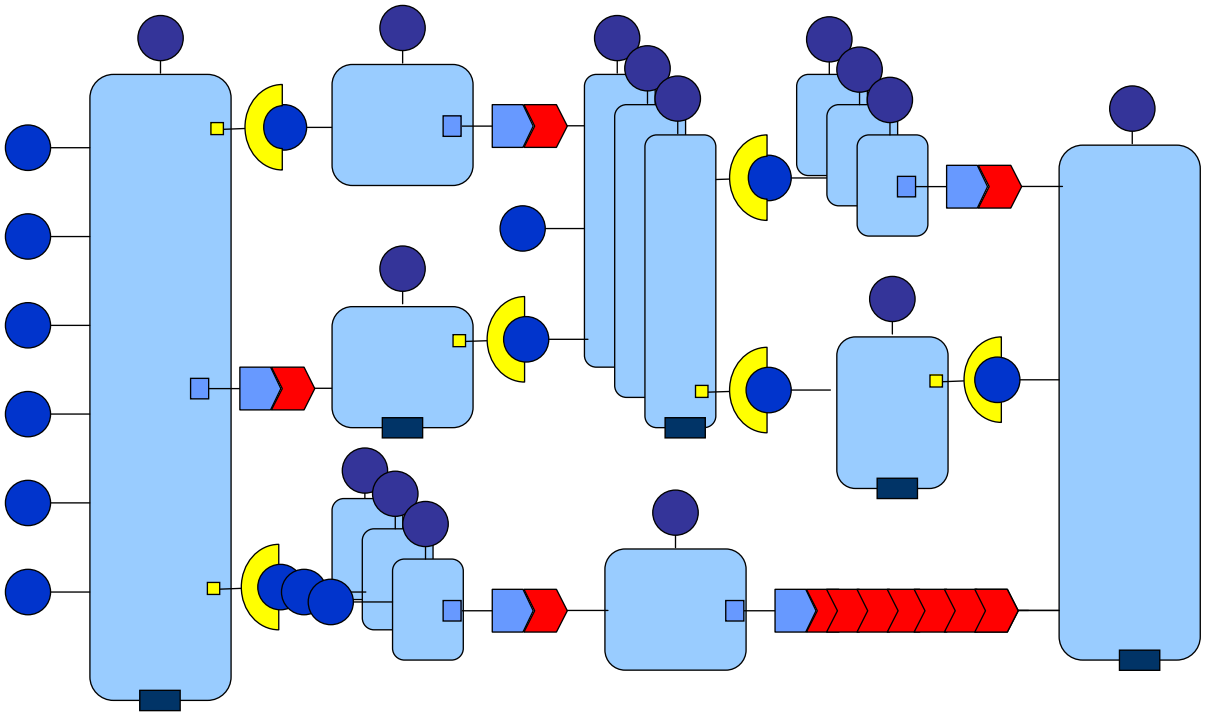
Le modèle abstrait de composants CORBA

- **Permet de capturer les fonctionnalités externes des composants CORBA**
  - Facettes = interfaces offertes
  - Réceptacles = interfaces requises
  - Puits = événements consommés
  - Sources = événements produits
  - Attributs = propriétés configurables
  - Opérations des maisons / gestionnaires de composants
- **Exprimé via le langage OMG IDL 3.0**
  - Nouvelles constructions syntaxiques masquant des canevas de conception
  - Traduites vers des interfaces OMG IDL compatibles CORBA 2.x

Un composant CORBA



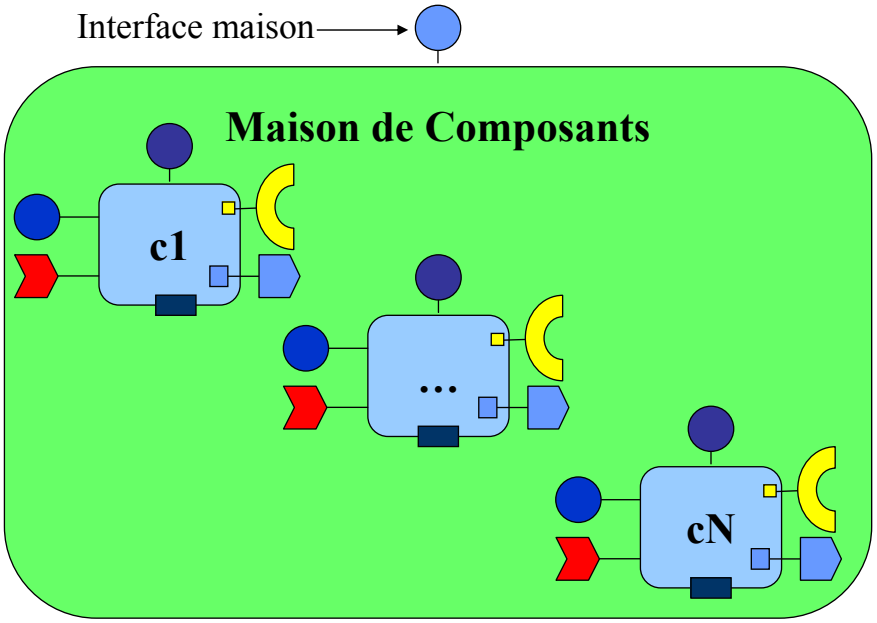
Construire une application CCM = Assembler des composants CORBA



Types et instances de composants CORBA

- **Un type de composants**
  - Nouveau méta-type et mot-clé OMG IDL *component*
  - Identification de la liste des ports via nouveaux mots-clés
    - *provides* pour les facettes
    - *[multiple] uses* pour les réceptacles simples ou multiples
    - *consumes* pour les puits d'événements
    - *emits / publishes* pour les sources d'événements (1:1 | 1:N)
    - *[readonly] attribute* pour les propriétés configurables
  - Héritage simple entre types de composants
  - Héritage multiple d'interfaces (mot-clé *supports*)
- **Une instance de composant**
  - 1 référence distincte pour l'interface de base, chaque facette et chaque puits
  - API générique pour la connexion, navigation et introspection des ports
  - Créée et gérée par une unique instance de maison

Une maison de composants CORBA



Types et instances de maisons de composants CORBA

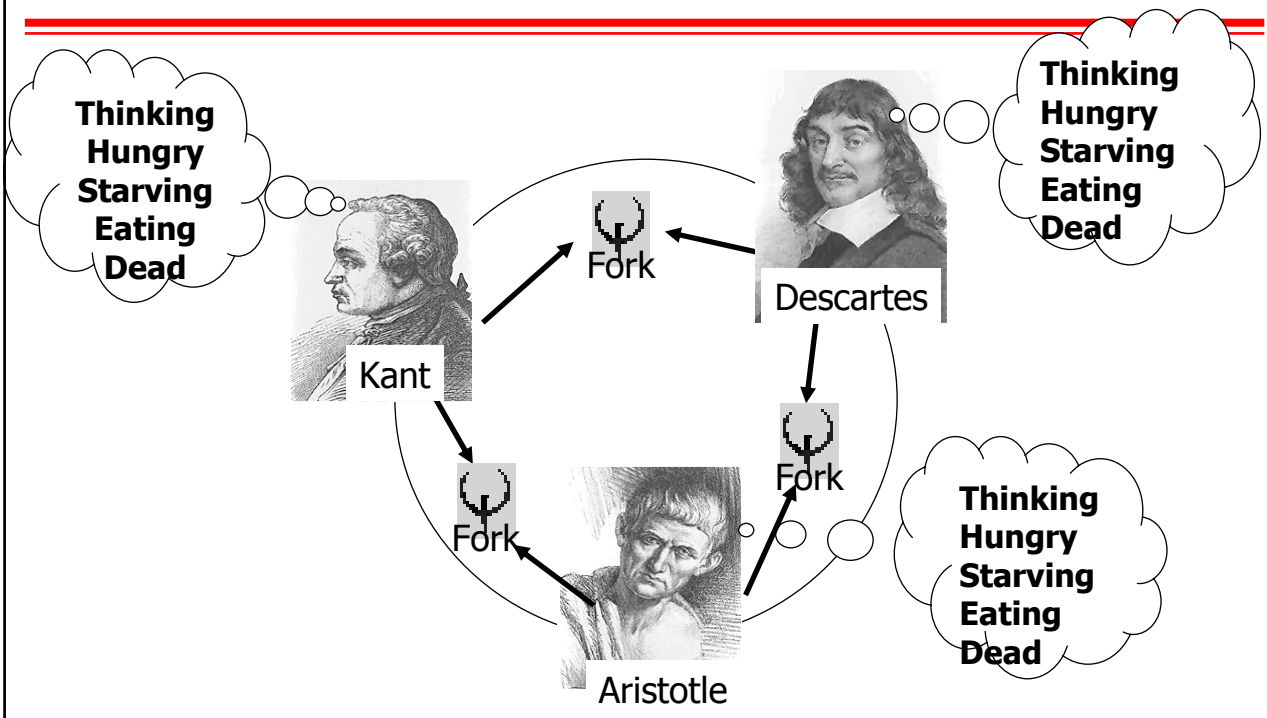
Un type de maisons

- Nouveau méta-type et mot-clé OMG IDL *home*
- Identification de l'unique type de composants gérés (mot-clé *manages*)
  - Plusieurs types de maisons pour le même type de composants possible
- Identification du type de l'identité / clé persistante des composants
  - mot-clé *primarykey* ; pour composants Entity
- Opérations `factory` et `finder`
- N'importe quelle opération métier
- Héritage simple entre types de maisons
- Héritage multiple d'interfaces (mot-clé *supports)*

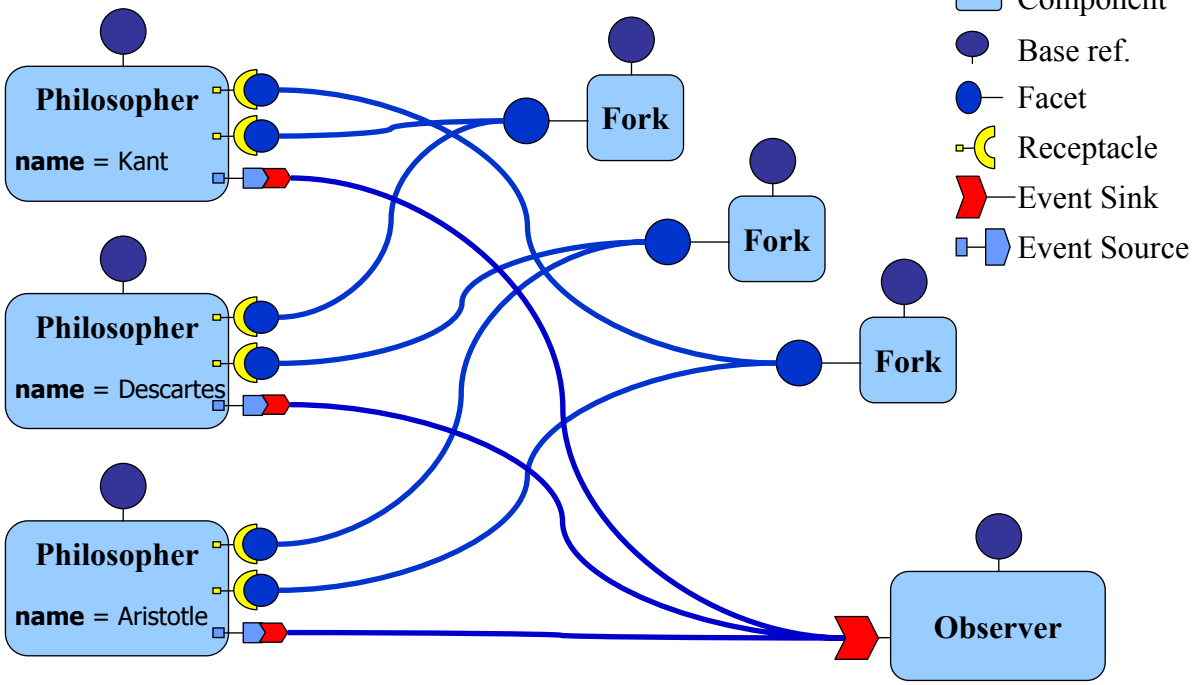
Une instance de maison

- 1 référence pour l'interface de base
- Instanciée lors du déploiement
- Co localisation des instances de composants gérés

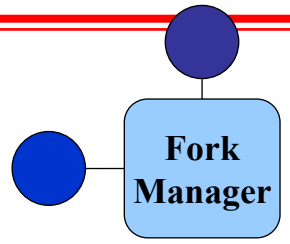
L'exemple du dîner des philosophes



Les composants CORBA du dîner des philosophes

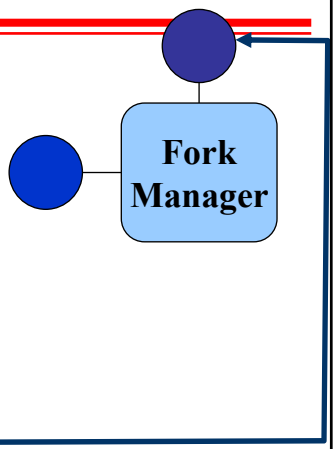


Le composant ForkManager




```
exception InUse {};

interface Fork
{
   void get() raises (InUse);
   void release();
};

// The fork component.
component ForkManager
{
   // The fork facet used by philosophers.
   provides Fork the_fork;
};

// Home for instantiating ForkManager components.
home ForkHome manages ForkManager {};
```

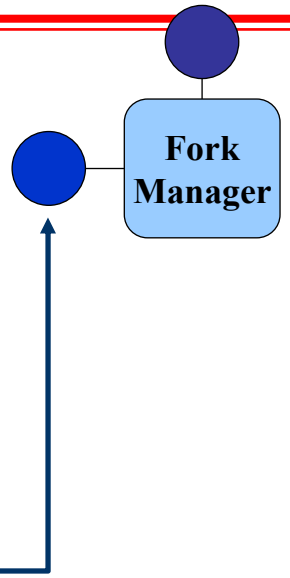
Le composant ForkManager



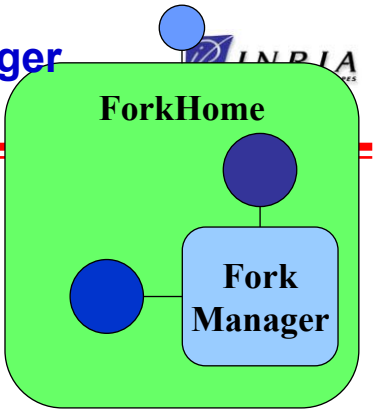

```
exception InUse {};

interface Fork
{
   void get() raises (InUse);
   void release();
};

// The fork component.
component ForkManager
{
   // The fork facet used by philosophers.
   provides Fork the_fork;
};

// Home for instantiating ForkManager components.
home ForkHome manages ForkManager {};
```

Le composant ForkManager

```
exception InUse {};

interface Fork
{
   void get() raises (InUse);
   void release();
};

// The fork component.
component ForkManager
{
   // The fork facet used by philosophers.
   provides Fork the_fork;
};

// Home for instantiating ForkManager components.
home ForkHome manages ForkManager {};
```



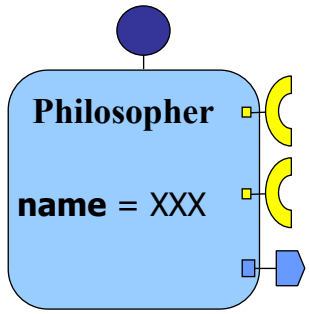
La maison pour ForkManager




```
exception InUse {};

interface Fork
{
   void get() raises (InUse);
   void release();
};

// The fork component.
component ForkManager
{
   // The fork facet used by philosophers.
   provides Fork the_fork;
};

// Home for instantiating ForkManager components.
home ForkHome manages ForkManager {};
```

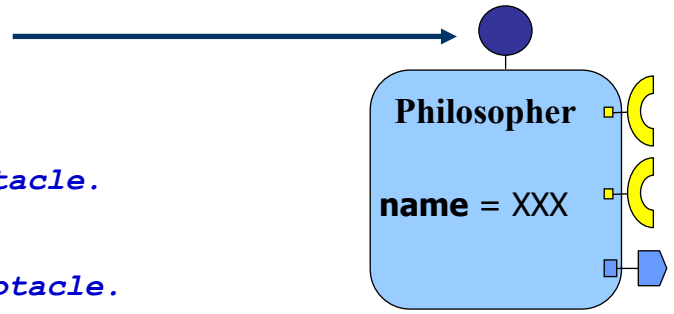
Le composant Philosopher

```
enum PhilosopherState
{
   EATING, THINKING, HUNGRY,
   STARVING, DEAD
};

eventtype StatusInfo
{
   public string name;
   public PhilosopherState state;
   public unsigned long ticks_since_last_meal;
   public boolean has_left_fork;
   public boolean has_right_fork;
};
```



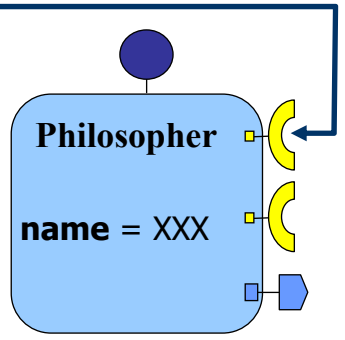
Le composant Philosopher

```
component Philosopher
{
   attribute string name;

   // The left fork receptacle.
   uses Fork left;

   // The right fork receptacle.
   uses Fork right;

   // The status info event source.
   publishes StatusInfo info;
};

home PhilosopherHome manages Philosopher {
    factory new(in string name);
};
```



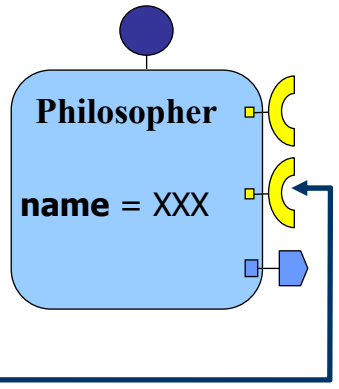
Le composant Philosopher

```
component Philosopher
{
   attribute string name;
   // The left fork receptacle.
   uses Fork left;
   // The right fork receptacle.
   uses Fork right;
   // The status info event source.
   publishes StatusInfo info;
};

home PhilosopherHome manages Philosopher {
    factory new(in string name);
};
```



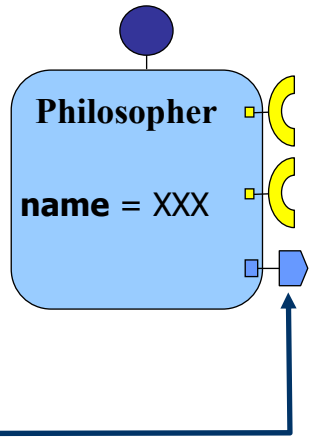
Le composant Philosopher

```
component Philosopher
{
   attribute string name;
   // The left fork receptacle.
   uses Fork left;
   // The right fork receptacle.
   uses Fork right;
   // The status info event source.
   publishes StatusInfo info;
};

home PhilosopherHome manages Philosopher {
    factory new(in string name);
};
```



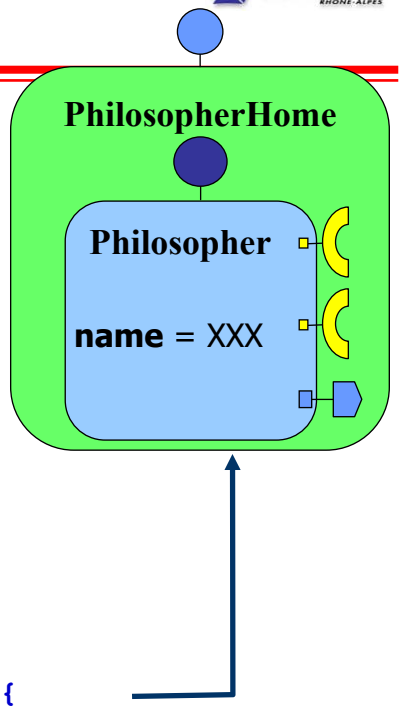
Le composant Philosopher

```
component Philosopher
{
   attribute string name;
   // The left fork receptacle.
   uses Fork left;
   // The right fork receptacle.
   uses Fork right;
   // The status info event source.
   publishes StatusInfo info;
};

home PhilosopherHome manages Philosopher {
    factory new(in string name);
};
```

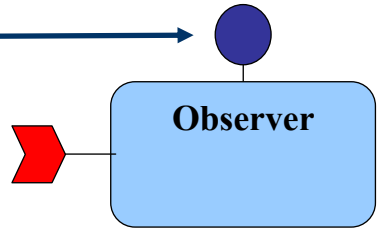


La maison de Philosopher

```
component Philosopher
{
   attribute string name;
   // The left fork receptacle.
   uses Fork left;
   // The right fork receptacle.
   uses Fork right;
   // The status info event source.
   publishes StatusInfo info;
};

home PhilosopherHome manages Philosopher {
    factory new(in string name);
};
```



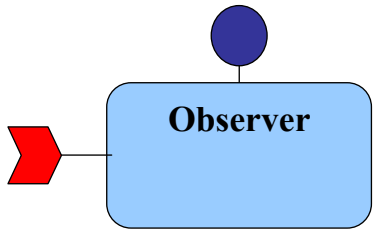
Le composant Observer



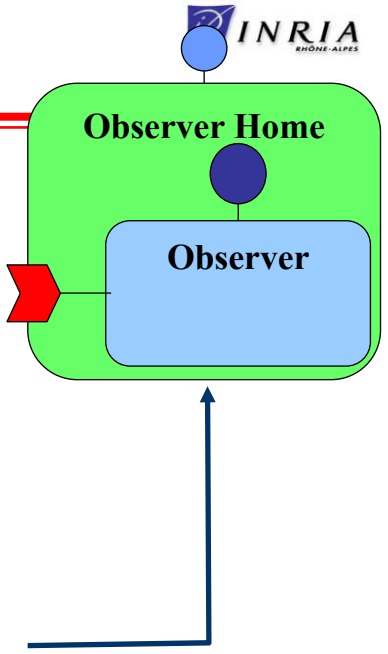

```
component Observer
{
   // The status info sink port.
   consumes StatusInfo info;
};

// Home for instantiating observers.
home ObserverHome manages Observer {};
```

Le composant Observer

```
component Observer
{
   // The status info sink port.
   consumes StatusInfo info;
};

// Home for instantiating observers.
home ObserverHome manages Observer {};
```

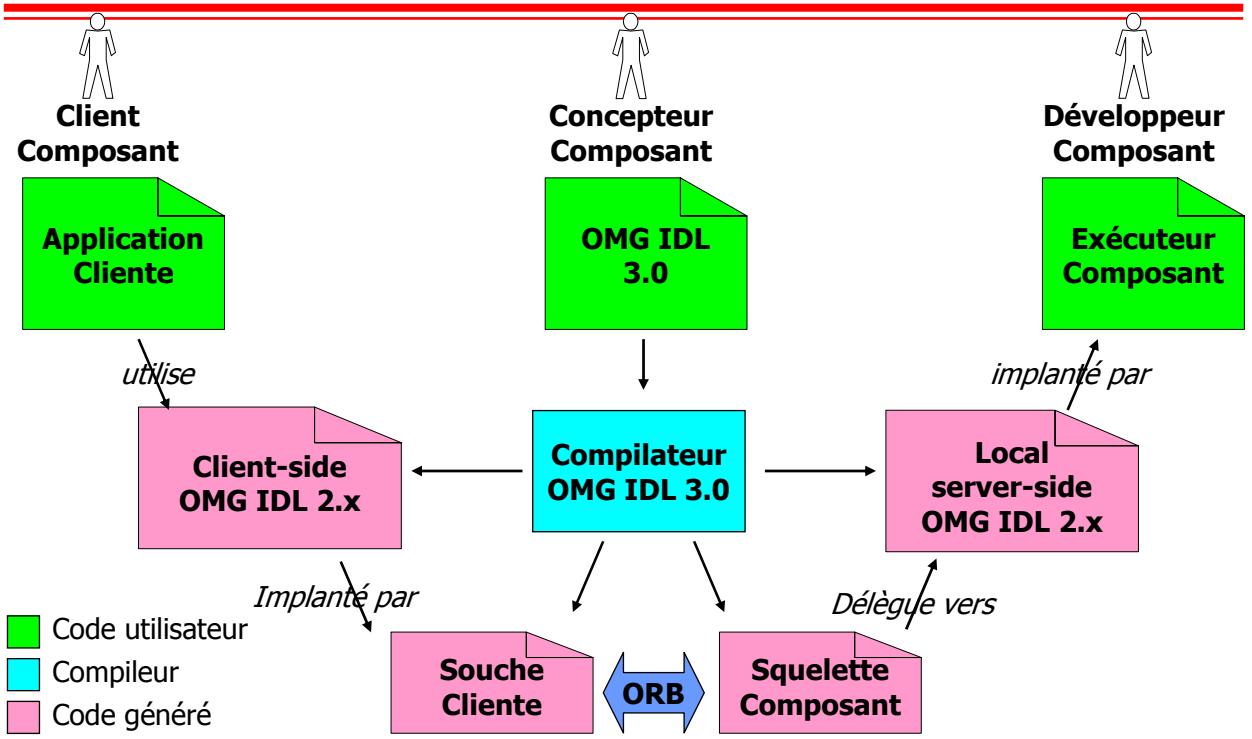


Le maison d'Observer

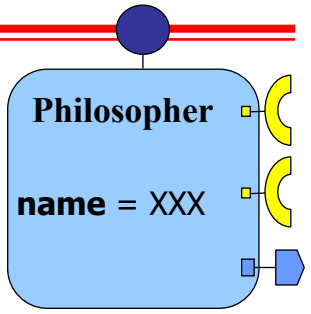



```
component Observer
{
   // The status info sink port.
   consumes StatusInfo info;
};

// Home for instantiating observers.
home ObserverHome manages Observer {};
```

La compilation OMG IDL 3.0



Projection cliente pour le composant Philosopher

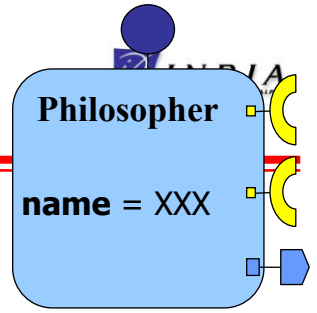



```
component Philosopher {
   attribute string name;
   uses Fork left;
   uses Fork right;
   publishes StatusInfo info;
};
```

Traduit en

```
interface Philosopher :
            ::Components::CCMObject {
  attribute string name;

   .../...
```

Projection cliente pour le composant Philosopher




```
    void connect_left(in Fork cnx) raises(…);
    Fork disconnect_left() raises(…);
    Fork get_connection_left();

    void connect_right(in Fork cnx) raises (…);
    Fork disconnect_right() raises (…);
    Fork get_connection_right();

    Components::Cookie subscribe_info(
              in StatusInfoConsumer consumer) raises(…);
    StatusInfoConsumer unsubscribe_info(
                  in Components::Cookie ck) raises(…);
};
```

Component Implementation Framework (CIF)

- **Définit le modèle de programmation de l'implantation des composants**
  - Concepts de composition, de segment et d'exécuteur
  - Projection composant OMG IDL en interfaces locales OMG IDL
  - Interfaces OMG IDL des conteneurs
    - `SessionContext` et `EntityContext`, …
  - Interfaces OMG IDL des exécuteurs de composant
    - `SessionComponent` et `EntityComponent`, …
  - Component Implementation Description Language (CIDL)

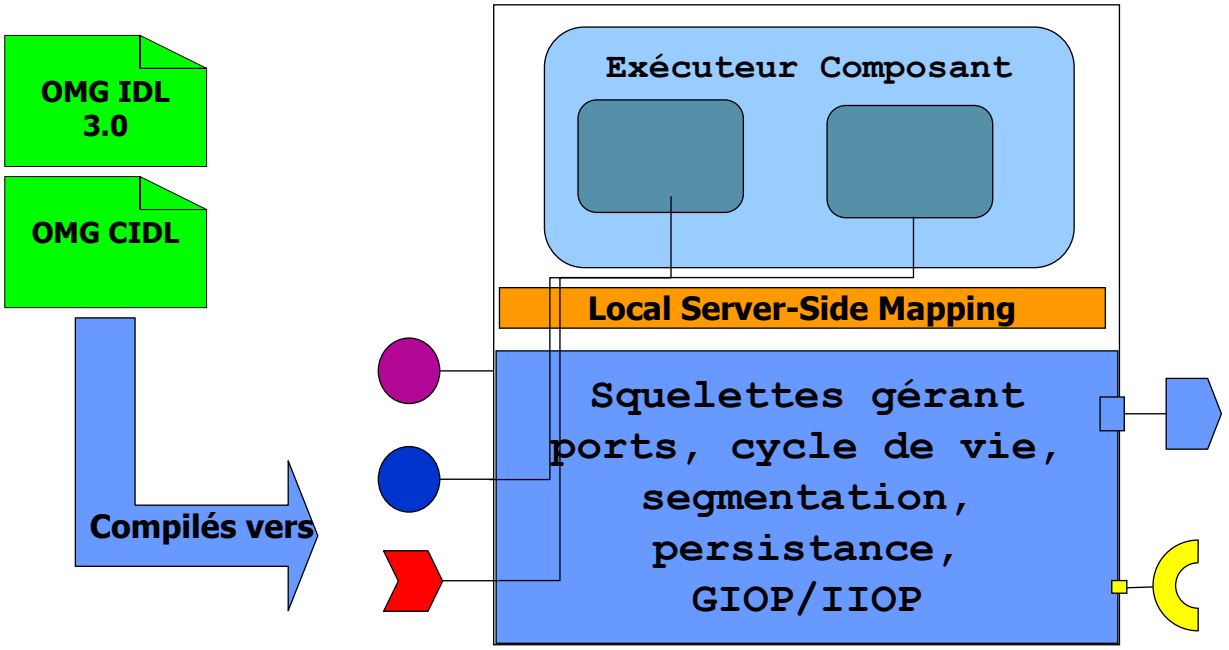
Component Implementation Definition Language (CIDL)

- **Description de la structure d'implantation des composants**
  - Composition d'une implantation de maison et d'une implantation de composant
  - Segmentation de l'implantation des composants
  - Association d'un état persistant à chaque segment
- **OMG Persistent State Definition Language (OMG PSS)**
  - Description des états persistants
- **Génération de squelettes prenant en charge**
  - Les ports des composants
  - Le cycle de vie des composants
  - La segmentation de l'implantation
  - La liaison avec les états persistants
  - L'implantation par défaut des opérations de retour
    - `configuration_complete`, `ccm_activate`, `ccm_passivate`, `ccm_load`, `ccm_store`, `ccm_remove`

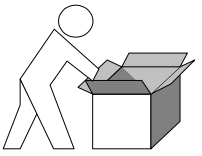
Composition CIDL pour le composant ForkManager

```
import DiningPhilosophers;

composition session ForkManagerComposition
{
  home executor ForkHomeSessionImpl
  {
    implements DiningPhilosophers::ForkHome;
    manages ForkManagerSessionImpl {
    segment Seg {
       provides facet the_fork; }
    };
  };
};
```

La compilation CIDL



Le conditionnement, l'assemblage et le déploiement CCM



Conditionnement, assemblage et déploiement via le CCM

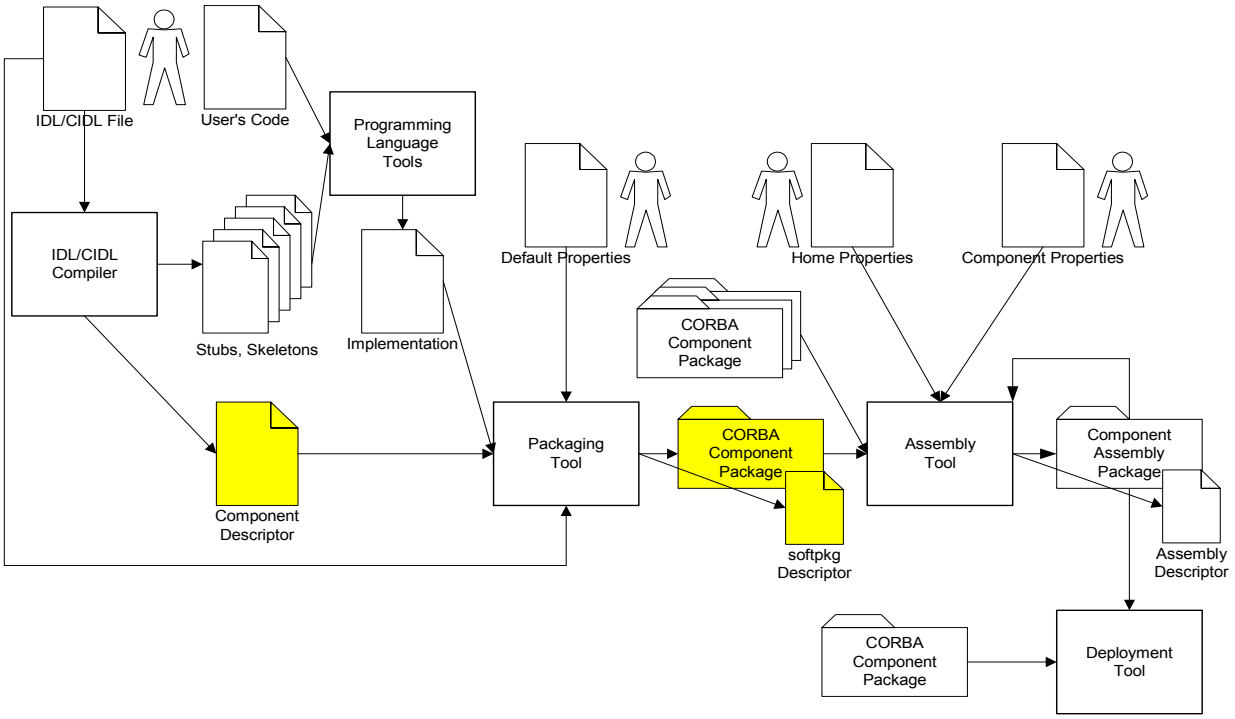
- **CORBA 2.x : aucun moyen standard pour configurer, conditionner, diffuser et déployer des applications réparties**

- **CCM fournit une technologie de conditionnement, d'assemblage et de déploiement d'applications à base de composants hétérogènes et distribués**
  - Composants et assemblages conditionnés dans des archives ZIP
  - Auto-description via divers descripteurs XML
  - API de l'infrastructure de déploiement répartie
  - Déploiement totalement automatisé

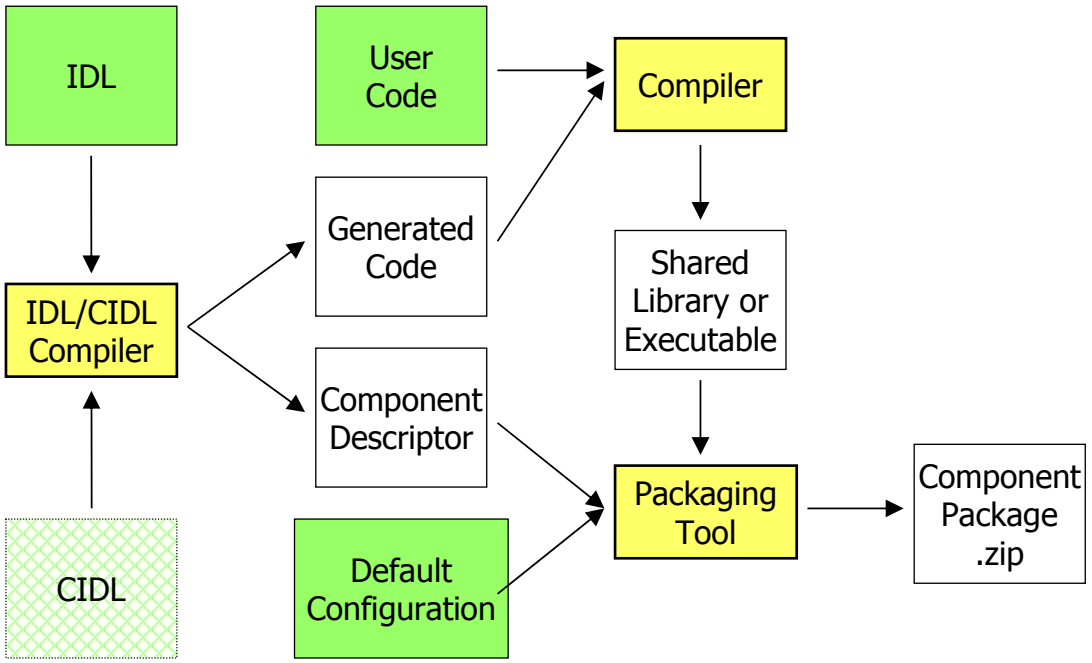
Archive de composant

- **Unité de conditionnement réutilisable dans des assemblages**
  - Auto descriptive et auto suffisante
- **Fichier ZIP contenant**
  - Un Software Package Descriptor (.csd) décrivant le contenu de l'archive
  - Le fichier OMG IDL du composant, des interfaces des ports et de la maison
  - Des Property File Descriptor (.cpf)
    - Les valeurs par défauts des attributs
  - Une ou plusieurs implantations binaires
    - E.g. pour différents OSs, ORBs, processeurs, QoS, ...
  - Des CORBA Component Descriptor (.ccd)
    - Configuration des politiques techniques gérées par les conteneurs

Artefacts de conditionnement des composants



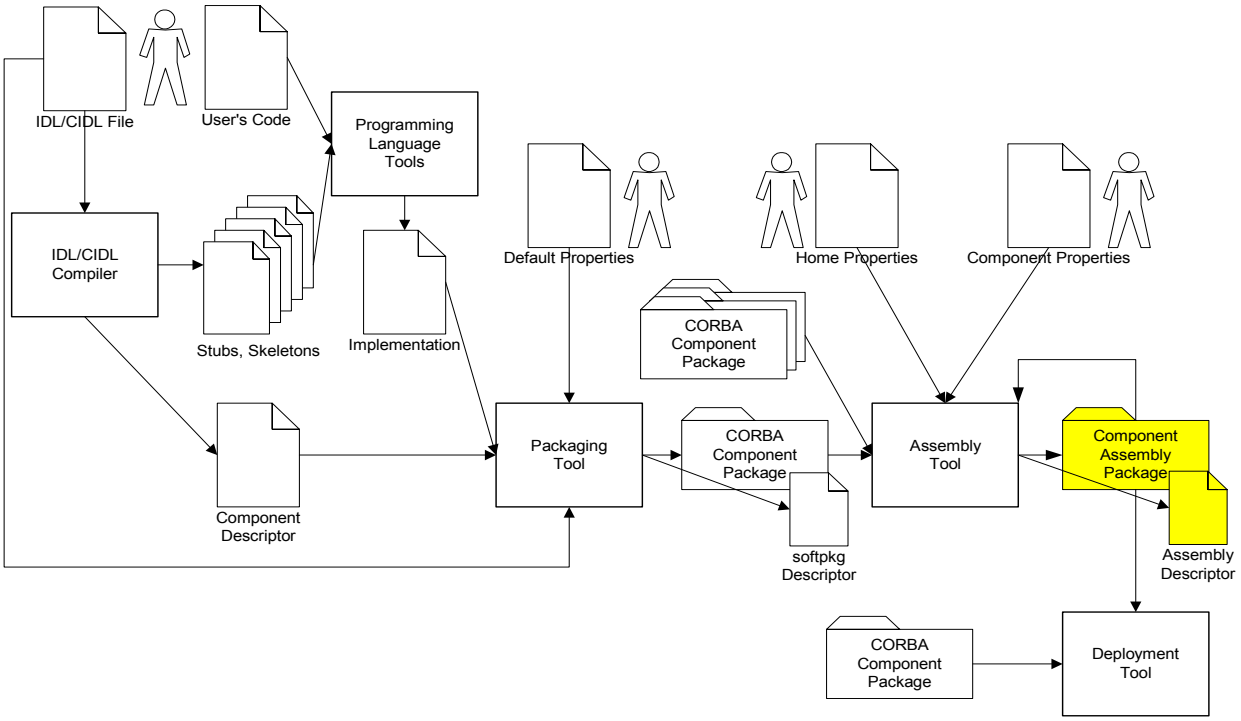
Conditionnement de composants



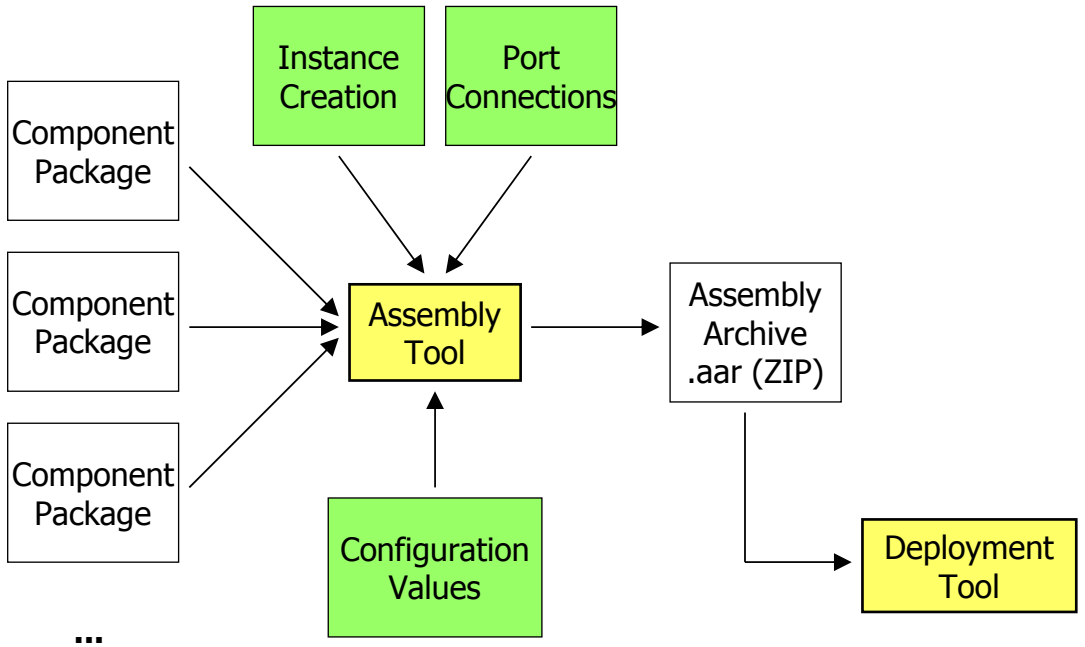
Archive d'assemblage

- **Unité de conditionnement réutilisable pour le déploiement automatique d'applications réparties**
  - Auto descriptive et auto suffisante
- **Fichier ZIP contenant**
  - Un Component Assembly Descriptor (.cad)
    - Le placement des maisons à créer
    - Les instances de composants à créer
    - Les connexions entre les ports
  - Des Property File Descriptors (.cpf)
    - Les valeurs initiales des attributs
  - Une ou plusieurs archives de composants
    - Soit inclusion du contenu des archives
    - Soit inclusion des archives
    - Soit références URL vers des archives externes

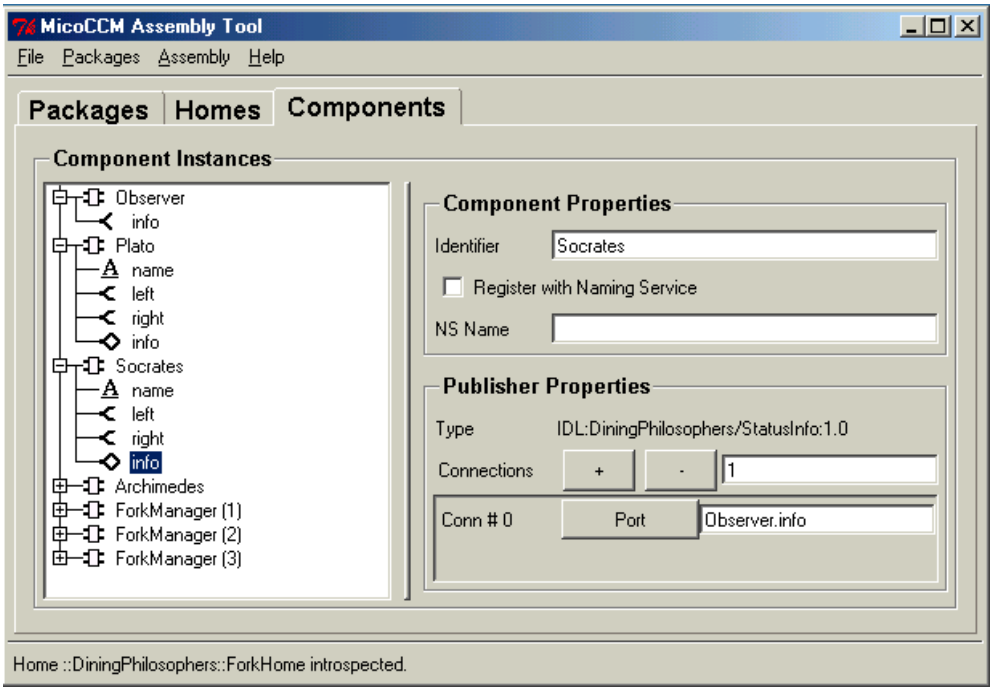
Artefacts d'assemblage des composants



Assemblage de composants



L'outil d'assemblage de MicoCCM



Les descripteurs XML du CCM

- **Software Package Descriptor (.csd)**
  - Description du contenu d'une archive de composant logiciel
  - Identification d'une ou plusieurs implantations binaires
- **CORBA Component Descriptor (.ccd)**
  - Informations techniques générées depuis définitions CIDL
  - Paramétrage des politiques gérées par les conteneurs
- **Component Property File Descriptor (.cpf)**
  - Configuration des attributs des maisons et des composants
- **Component Assembly Descriptor (.cad)**
  - Identification des archives de composants utilisées
  - Description d'un assemblage de composants

Software Package Descriptor (.csd)

- **Informations générales**
  - Titre, description, auteurs, compagnies, liens Web, licences
- **Lien sur le descripteur XML de propriétés par défaut (.cpf)**
- **Lien sur le fichier OMG IDL du composant**
- **Pour chaque implantation**
  - Informations techniques
    - Systèmes d'exploitation, processeurs, langages, compilateurs et ORBs supportés
    - Dépendances vers des bibliothèques externes
    - Pré requis de déploiement
  - Lien sur descripteurs .cpf et .ccd spécifiques
  - Lien sur le fichier d'implantation
    - Bibliothèque partagée, classe ou archive Java, exécutable, …
  - Point d'entrée de l'implantation de la maison, i.e. fonction statique

Exemple de Software Package Descriptor

```
<?xml version='1.0'?>
<!DOCTYPE softpkg>
<softpkg name="PhilosopherHome">
  <idl id="IDL:DiningPhilosophers/PhilosopherHome:1.0">
    <fileinarchive name="philo.idl"/>
  </idl>
  <implementation id="*">
    <code type="DLL">
      <fileinarchive name="philo.dll"/>
      <entrypoint>create_DiningPhilosophers_PhilosopherHome</entrypoint>
    </code>
  </implementation>
</softpkg>
```

Software Package Descriptor pour Composant Observer

```
<?xml version="1.0"?>
<!DOCTYPE softpkg SYSTEM "softpkg.dtd">

<softpkg name="Observer" version="1,0,0,0">
  <pkgtype>CORBA Component</pkgtype>
  <title>Observer</title>
  <author>
    <name>Philippe Merle</name>
    <company>INRIA</company>
    <webpage href="http://www.inria.fr"/>
  </author>
  <description>The CCM dining philosophers example</description>
```

Software Package Descriptor pour Composant Observer

```
<license href= "http://www.objectweb.org/license.html"/>

<idl id="IDL:DiningPhilosophers/Observer:1.0">
  <link href="http://www.objectweb.org/philo.idl"/>
</idl>

<descriptor type="CORBA Component">
  <fileinarchive name="observer.ccd"/>
</descriptor>

<propertyfile>
  <fileinarchive name="observer.cpf"/>
</propertyfile>

<implementation> . . .  </implementation>
</softpkg>
```

Software Package Descriptor pour Composant Observer

```
<implementation id="Observer_impl">
   <os name="WinNT" version="4,0,0,0"/>
   <os name="Linux" version="2,2,17,0"/>
   <processor name="x86"/>
   <compiler name="JDK"/>
   <programminglanguage name="Java"/>
   <code type="Java class">
       <fileinarchive name="ObserverHomeImpl.class"/>
       <entrypoint>ObserverHomeImpl.create_home</entrypoint>
   </code>
   <runtime name="Java VM" version="1,2,2,0"/>
   <runtime name="Java VM" version="1,3,0,0"/>
   <dependency>...</dependency>
</implementation>
```

Software Package Descriptor pour Composant Observer

```
<dependency type="ORB" action="assert">
      <name>OpenORB</name>
</dependency>

<dependency type="Java Class" action="install">
   <valuetypefactory
     repid="IDL:DiningPhilosophers/StatusInfo:1.0"
     valueentrypoint="DiningPhilosophers.StatusInfoDefaultFactory.create"
     factoryentrypoint="DiningPhilosophers.StatusInfoDefaultFactory">
     <fileinarchive
     name="DiningPhilosophers/StatusInfoDefaultFactory.class"/>
  </valuetypefactory>
</dependency>
```

Software Package Descriptor pour Composant Observer

```
<implementation id="observer_0x1">
  <os name="Win2000" />
  <processor name="x86" />
  <compiler name="VC++" />
  <programminglanguage name="C++" />
  <dependency type="DLL"><localfile name="jtc.dll"/></dependency>
  <dependency type="DLL"><localfile name="ob.dll"/></dependency>
  <descriptor type="CORBA Component">
    <fileinarchive name="observer.ccd" />
  </descriptor>
  <code type="DLL">
    <fileinarchive name="PhilosophersExecutors.dll"/>
    <entrypoint>create_ObserverHome</entrypoint>
  </code>
</implementation>
```

CORBA Component Descriptor (.ccd)

- **Informations techniques générées depuis CIDL**
  - Fonctionnalités des types de composants et de maisons
  - Ports et interfaces supportées
  - Catégorie du composant et segments
- **Politiques du conteneur à compléter**
  - Threading
  - Cycle de vie des servants POA
  - Transactions
  - Sécurité
  - Events
  - Persistance
  - Politiques POA étendues
- **Lien vers fichiers de propriétés pour la maison et ses instances de composants**

Exemple de CORBA Component Descriptor

```
<corbacomponent>
  <corbaversion>3.0</corbaversion>
  <componentrepid>IDL:DiningPhilosophers/Philosopher:1.0</componentrepid>
  <homerepid>IDL:DiningPhilosophers/PhilosopherHome:1.0</homerepid>
  <componentkind><session><servant lifetime="component"/></session></componentkind>
  <threading policy="multithread"/>
  <configurationcomplete set="true"/>
  <homefeatures name="PhilosopherHome" repid="IDL:...PhilosopherHome:1.0"/>
  <componentfeatures name="Philosopher" repid="IDL:...Philosopher:1.0">
    <ports>
      <publishes publishesname="info" eventtype="IDL:DiningPhilosophers/StatusInfo:1.0">
        <eventpolicy/>
      </publishes>
      <uses usesname="left" repid="IDL:DiningPhilosophers/Fork:1.0"/>
      <uses usesname="right" repid="IDL:DiningPhilosophers/Fork:1.0"/>
    </ports>
  </componentfeatures>
</corbacomponent>
```

CORBA Component Descriptor pour Composant Philosopher

```
<?xml version="1.0"?>
<!DOCTYPE corbacomponent SYSTEM "corbacomponent.dtd">

<corbacomponent>
  <corbaversion>3.0</corbaversion>
  <componentrepid repid=
        "IDL:DiningPhilosophers/Philosopher:1.0"/>
  <homerepid repid=
        "IDL:DiningPhilosophers/PhilosopherHome:1.0"/>
  <componentkind>
    <process><servant lifetime="container" /></process>
   </componentkind>
  <security rightsfamily="CORBA"
              rightscombinator="secanyrights" />
  <threading policy="multithread" />
  <configurationcomplete set="true" />
```

CORBA Component Descriptor pour Composant Philosopher

```
<homefeatures name="PhilosopherHome"
              repid="IDL:DiningPhilosophers/PhilosopherHome:1.0"/>
<componentfeatures name="Philosopher"
              repid="IDL:DiningPhilosophers/Philosopher:1.0">
  <ports>
    <uses usesname="right"
          repid="IDL:DiningPhilosophers/Fork:1.0" />
    <uses usesname="left"
          repid="IDL:DiningPhilosophers/Fork:1.0" />
    <publishes emitsname="info"
               eventtype="StatusInfo">
      <eventpolicy policy="normal" />
    </publishes>
  </ports>
</componentfeatures>
<interface name="Fork" repid="IDL:DiningPhilosophers/Fork:1.0"/>
```

CORBA Component Descriptor pour Composant Philosopher

```
<segment name="philosopherseg" segmenttag="1">
  <segmentmember facettag="1" />
  <containermanagedpersistence>
    <storagehome id="PSDL:PersonHome:1.0"/>
    <pssimplementation id="OpenORB-PSS" />
    <accessmode mode="READ_WRITE" />
    <psstransaction policy="TRANSACTIONAL" >
        <psstransactionisolationlevel level="SERIALIZABLE" />
    </psstransaction>
    <params>
      <param name="x" value="1" />
    </params>
  </containermanagedpersistence>
</segment>
</corbacomponent>
```

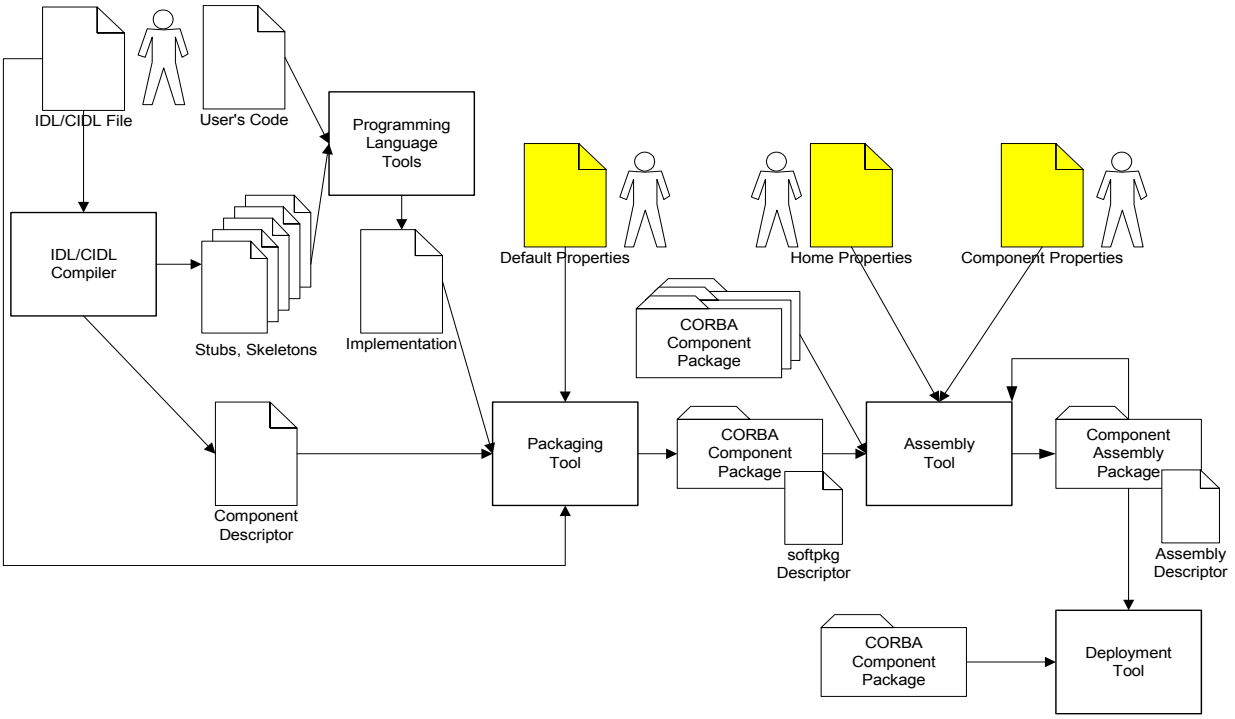
Property File Descriptor (.cpf)

- Permet de fixer les propriétés des instances de maisons et de composants
- Contient un couple (nom, valeur) pour chaque attribut à configurer
- Descripteurs référencés depuis
  - Software Package Descriptors
    - valeurs par défaut des attributs des composants
  - CORBA Component Descriptors
    - valeurs par défaut des attributs des composants et/ou des maisons
  - Component Assembly Descriptors
    - Valeurs initiales des instances de maisons et de composants

Property File pour Philosopher Kant

```
<?xml version="1.0"?>
<!DOCTYPE properties SYSTEM "properties.dtd">

<properties>
  <simple name="name" type="string">
    <description>Philosopher name</description>
    <value>Kant</value>
    <defaultvalue>Unknown</defaultvalue>
  </simple>
</properties>
```

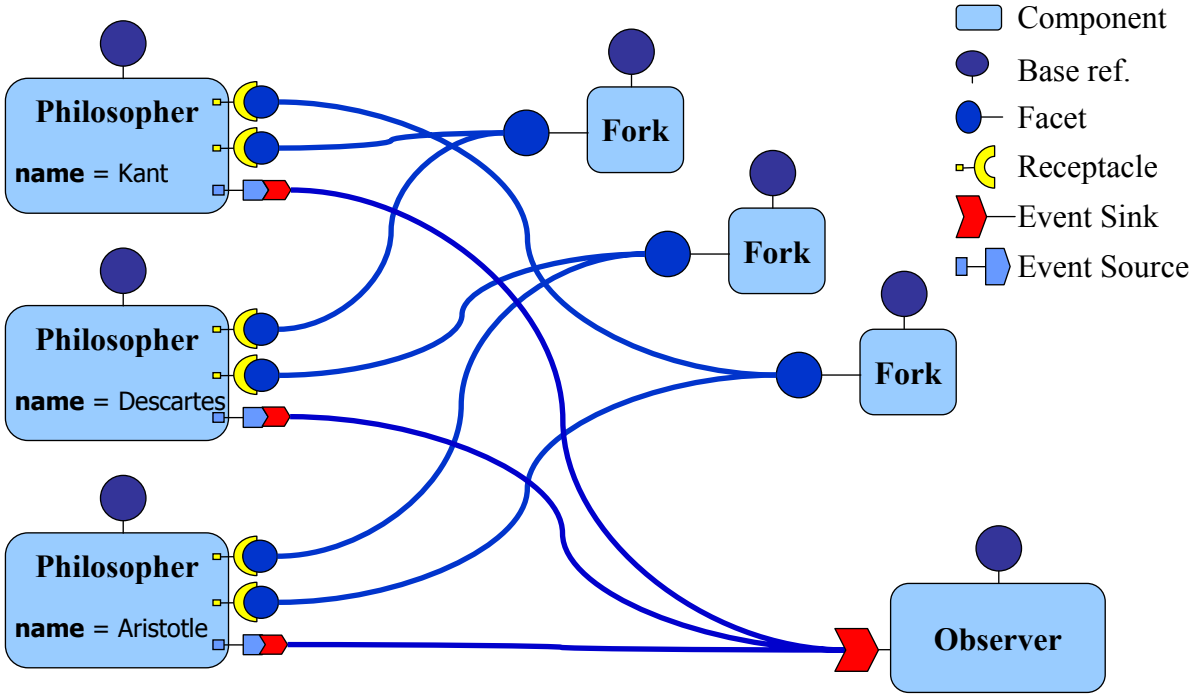
Les fichiers XML de propriétés CCM



Component Assembly Descriptor (.cad)

- **Implantations de composants à utiliser**
  - Références vers 1 ou plusieurs Component Software Descriptors
- **Instances de maisons à créer**
  - Placement, co-localisation et cardinalité
- **Instances de composants à créer**
- **Connexions entre les instances de composants**
  - Réceptacles → facettes et sources → puits
- **Valeurs initiales des attributs des maisons et composants**
- **Enregistrement des maisons et des composants**
  - Services de Nommage, Courtage et ComponentHomeFinder

Exemple du dîner des philosophes



Component Assembly Descriptor pour Dîner des Philosophes

```
<?xml version="1.0"?>
<!DOCTYPE componentassembly SYSTEM "componentassembly.dtd">
<componentassembly id="demophilo">
  <description>Dinner assembly descriptor</description>
  <componentfiles>
    <componentfile id="PhilosopherComponent">
      <fileinarchive name="philosopher.csd"/>
     </componentfile>
    <componentfile id="ObserverComponent">
      <fileinarchive name="observer.csd"/>
     </componentfile>
    <componentfile id="ForkManagerComponent">
      <fileinarchive name="forkmanager.csd"/>
    </componentfile>
  </componentfiles>
```

Component Assembly Descriptor pour Dîner des Philosophes

```
<partitioning>
  <homeplacement id="ObserverHome">
    <componentfileref idref="ObserverComponent"/>
    <registerwithnaming name="Dinner/ObserverHome"/>
  </homeplacement>
  <homeplacement id="PhilosopherHome">
    <componentfileref idref="PhilosopherComponent"/>
    <registerwithnaming name="Dinner/PhilosopherHome"/>
  </homeplacement>
  <homeplacement id="ForkHome">
    <componentfileref idref="ForkComponent"/>
    <registerwithnaming name="Dinner/ForkHome"/>
  </homeplacement>
 </partitioning><connections/></componentassembly>
```

Component Assembly Descriptor pour Dîner des Philosophes

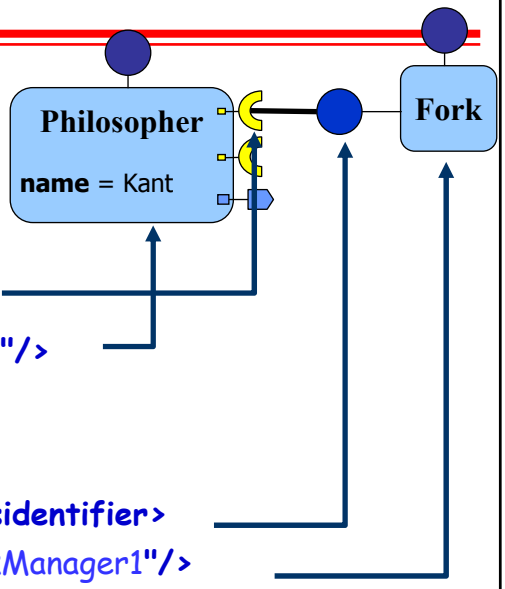
```
<partitioning>
  <homeplacement id="ObserverHome">
    <componentfileref idref="ObserverComponent"/>
    <componentinstantiation id="Freud"/>
    <registerwithnaming name="Dinner/ObserverComponent"/>
  </homeplacement>

  <homeplacement id="ForkHome">
    <componentfileref idref="ForkManagerComponent"/>
    <componentinstantiation id="ForkManager1"/>
    <componentinstantiation id="ForkManager2"/>
    <componentinstantiation id="ForkManager3"/>
    <registerwithhomefinder name="ForkHome"/>
  </homeplacement>
```

Component Assembly Descriptor pour Dîner des Philosophes

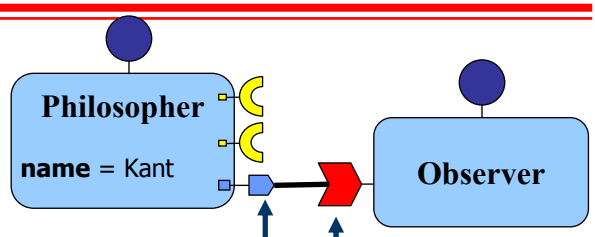
```
<homeplacement id="PhilosopherHome">
  <componentfileref idref="PhilosopherComponent"/>

  <componentinstantiation id="Kant">
      <componentproperties><fileinarchive name="Kant.cpf"/>
   </componentproperties></componentinstantiation>

  <componentinstantiation id="Descartes">
      <componentproperties><fileinarchive name="Descartes.cpf"/>
   </componentproperties></componentinstantiation>

  <componentinstantiation id="Aristotle">
      <componentproperties><fileinarchive name="Aristotle.cpf"/>
   </componentproperties></componentinstantiation>

 </homeplacement>
</partitioning>
```

Component Assembly Descriptor pour Dîner des Philosophes

```
<connections>
  <connectinterface>
    <usesport>
      <usesidentifier>left</usesidentifier>
      <componentinstantiationref idref="Kant"/>
    </usesport>
    <providesport>
      <providesidentifier>the_fork</providesidentifier>
      <componentinstantiationref idref="ForkManager1"/>
    </providesport>
  </connectinterface>
```

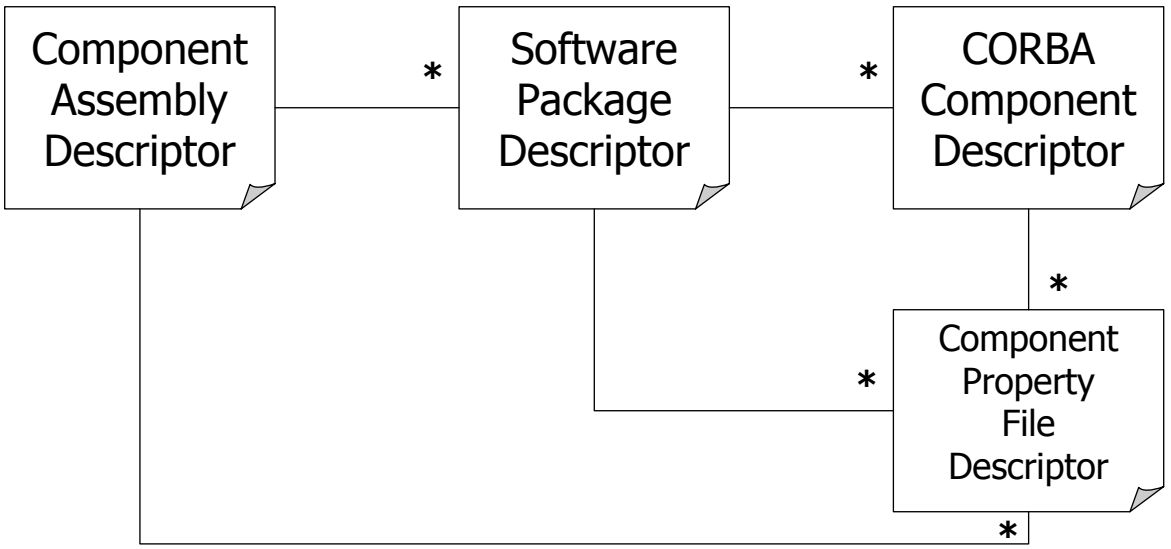


Component Assembly Descriptor pour Dîner des Philosophes

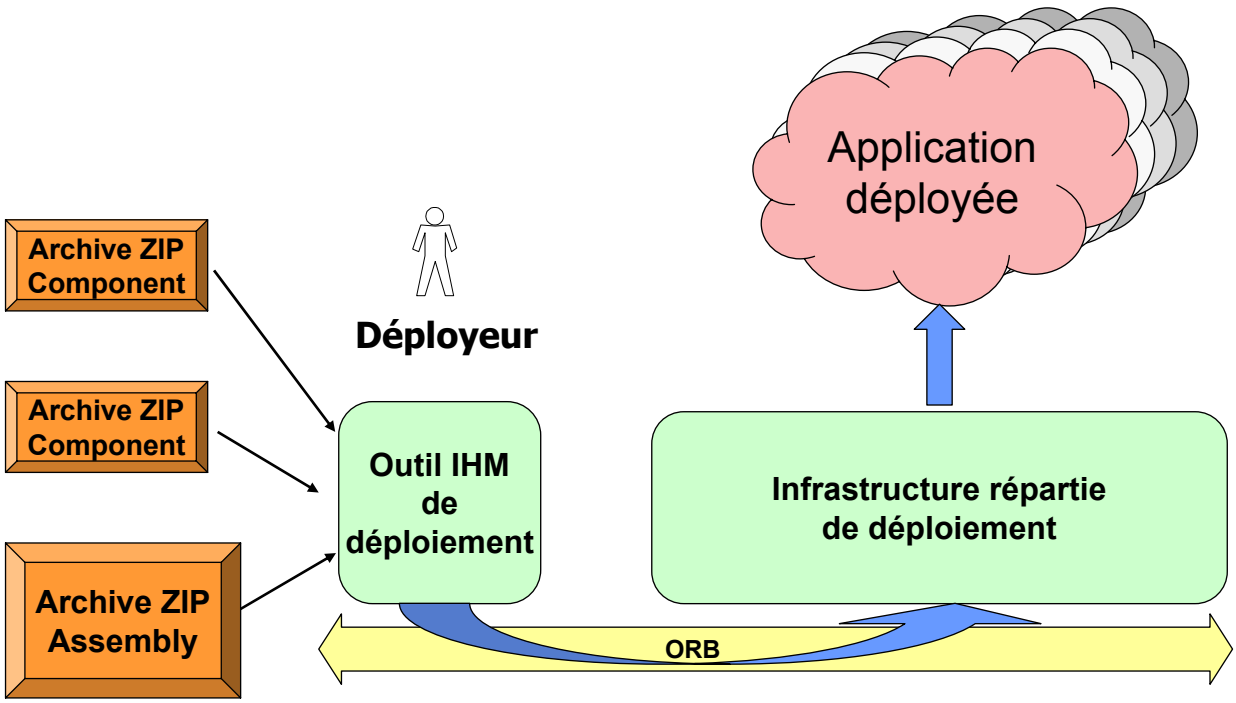



```
<connectevent>
  <publishesport>
    <publishesidentifier>info</publishesidentifier>
    <componentinstantiationref idref="Kant"/>
  </publishesport>
  <consumesport>
    <consumesidentifier>info</consumesidentifier>
    <componentinstantiationref idref="Freud"/>
  </consumesport>
</connectevent>
```

Référencement entre les descripteurs XML du CCM



Le déploiement d'assemblages CCM



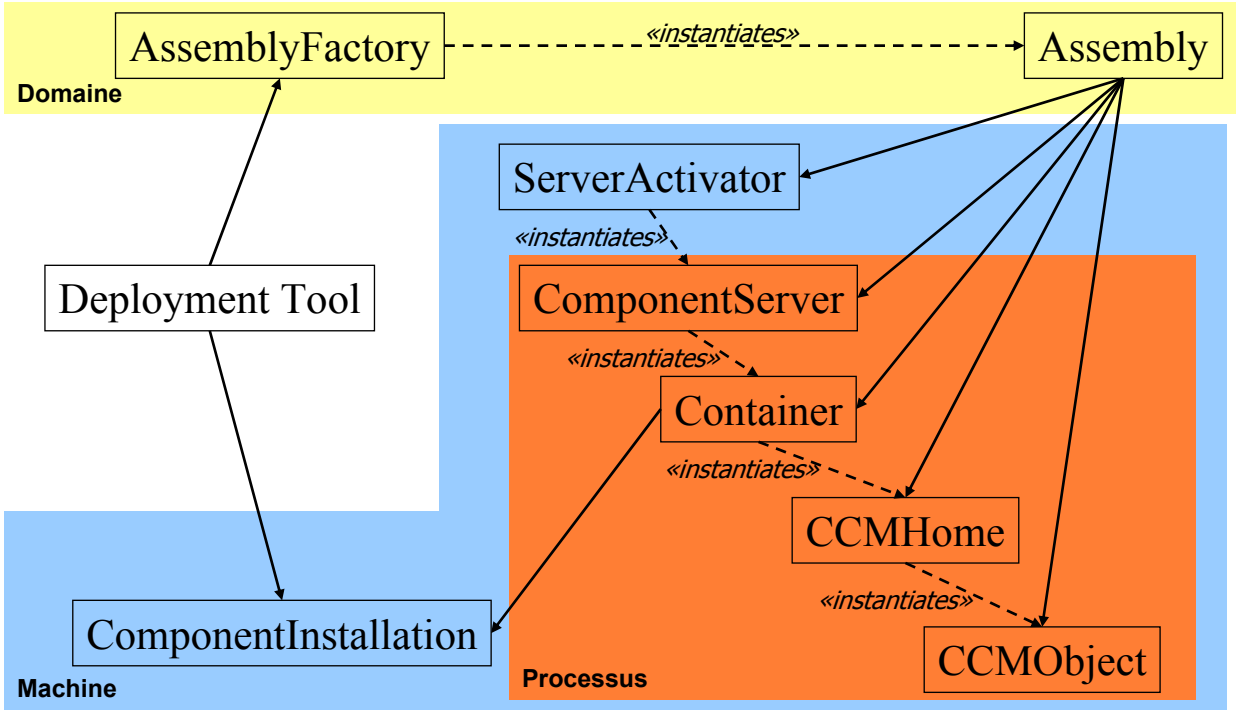
Le déploiement CCM

- **Déploiement automatique et réparti d'assemblages de composants CORBA**
  - Installation du code binaire des composants sur leur site d'exécution
    - Demande aux sites de télécharger le code accessible via Internet
  - Démarrage des serveurs d'applications nécessaires sur chaque site d'exécution
  - Création des conteneurs dans chaque serveur démarré
  - Installation des maisons dans les conteneurs
    - Chargement du code maison / composant en mémoire
  - Instanciation des composants à partir des maisons
  - Configuration des attributs des instances de composants
  - Interconnexion des composants via leurs ports
  - Démarrage effectif de tous les composants

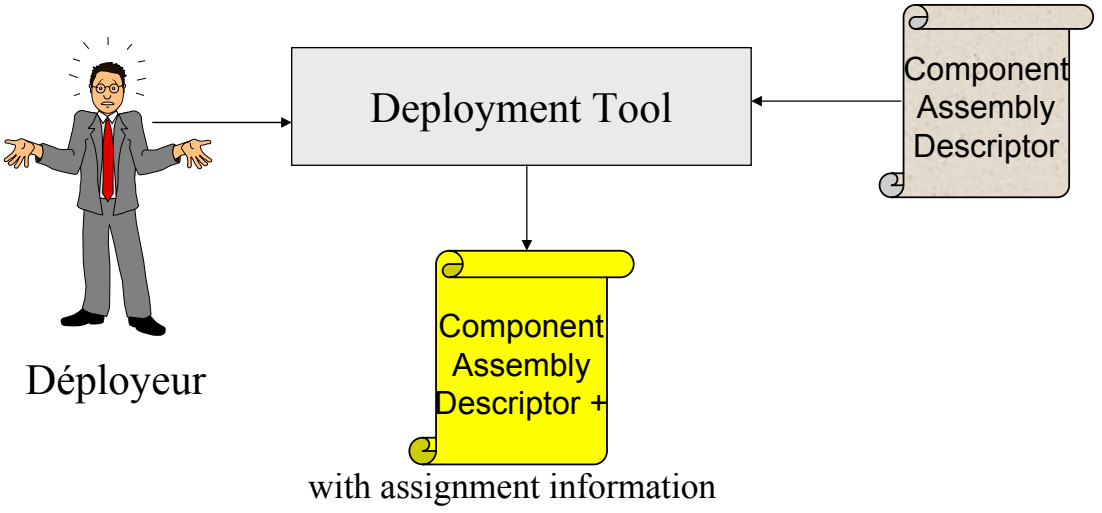
Interfaces de l'infrastructure de déploiement

- `ComponentInstallation`
  - Installation des implantations binaires de composants
  - 1 instance par machine
- `AssemblyFactory`
  - Fabrique des objets `Assembly`
  - 1 instance par domaine, e.g. un réseau, un ensemble de machines, …
- `Assembly`
  - Contrôleur du déploiement d'un assemblage
  - Interpréteur des Component Assembly Descriptor (CAD)
- `ServerActivator`
  - Fabrique d'objets `ComponentServer`
  - Une instance par machine
- `ComponentServer`
  - Fabrique d'objets `Container`
- `Container`
  - Installation et instanciation des maisons de composants

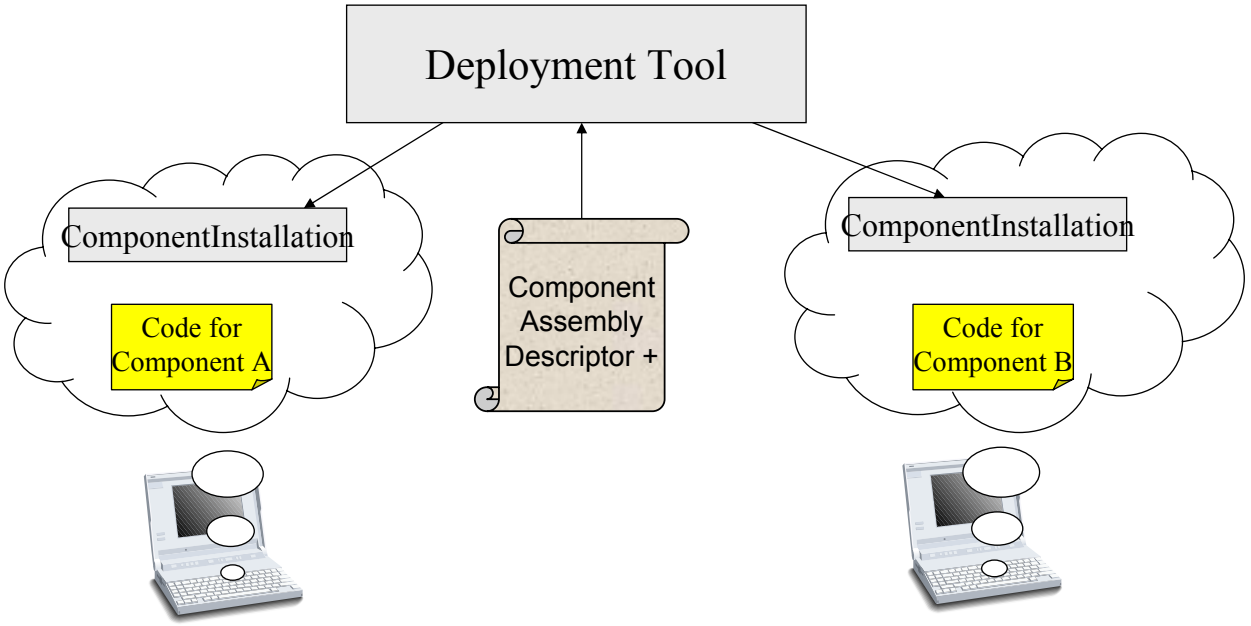
Le processus de déploiement du CCM



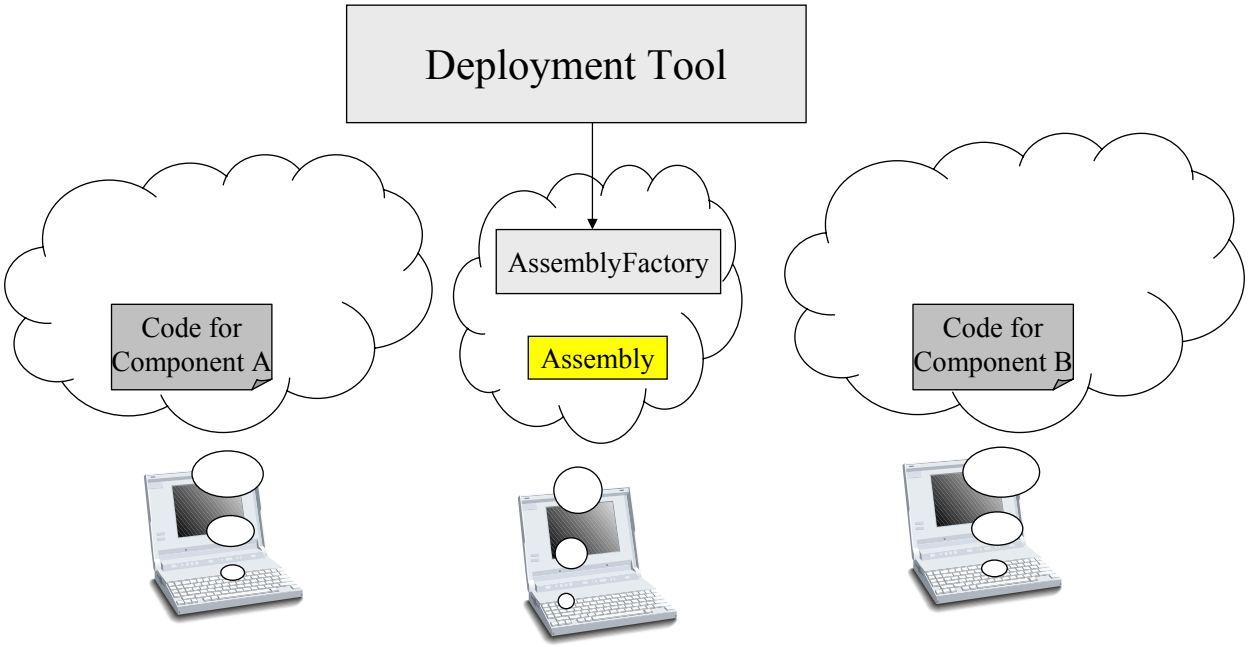
Scénario de déploiement



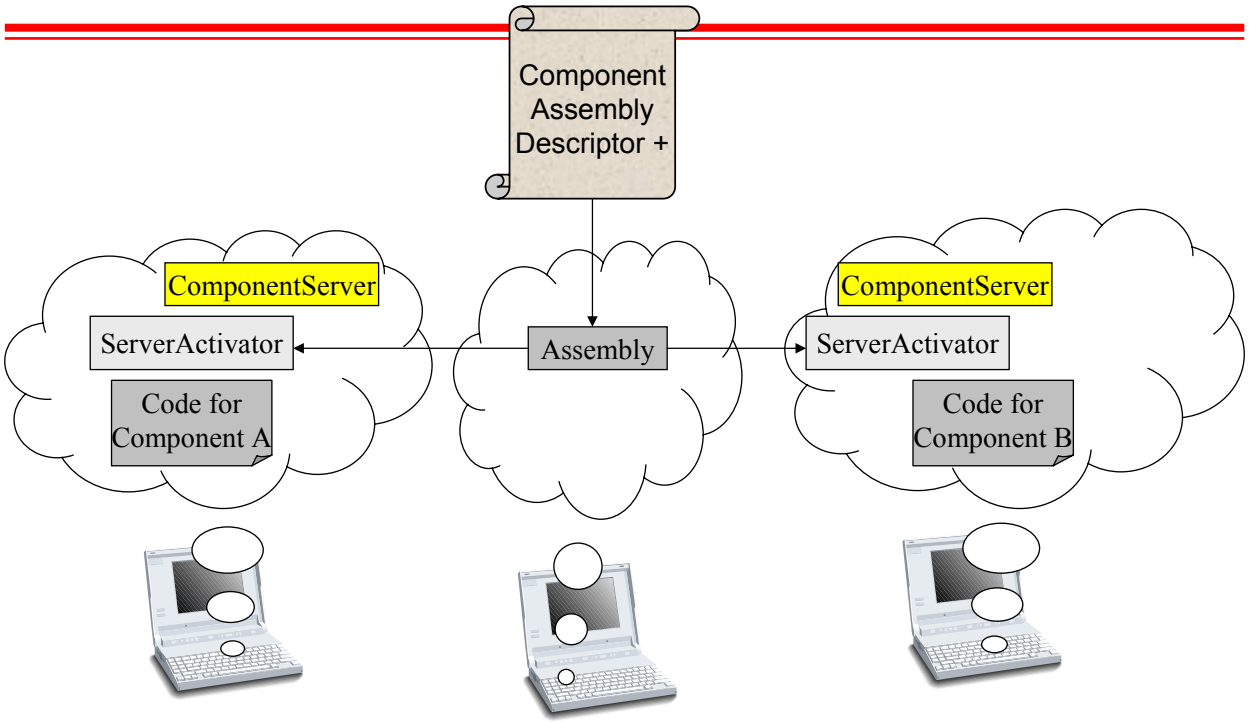
Scénario de déploiement : chargement des implantations



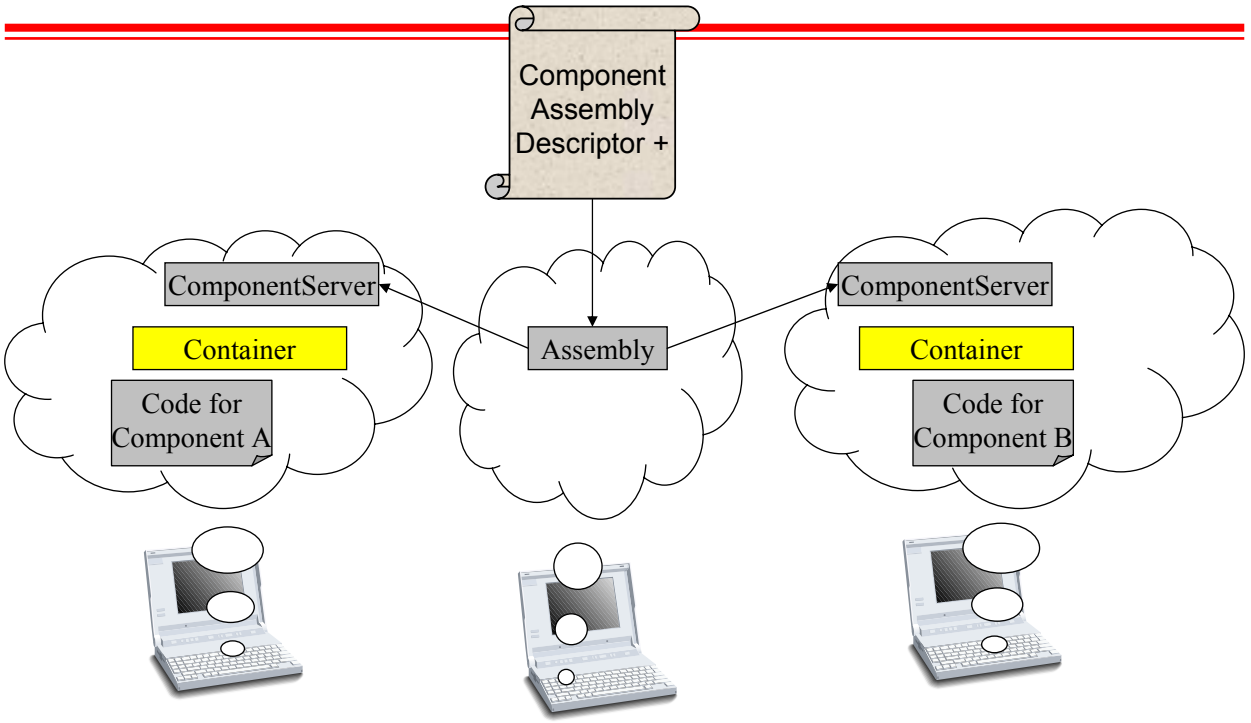
Scénario de déploiement : création de l'objet Assembly



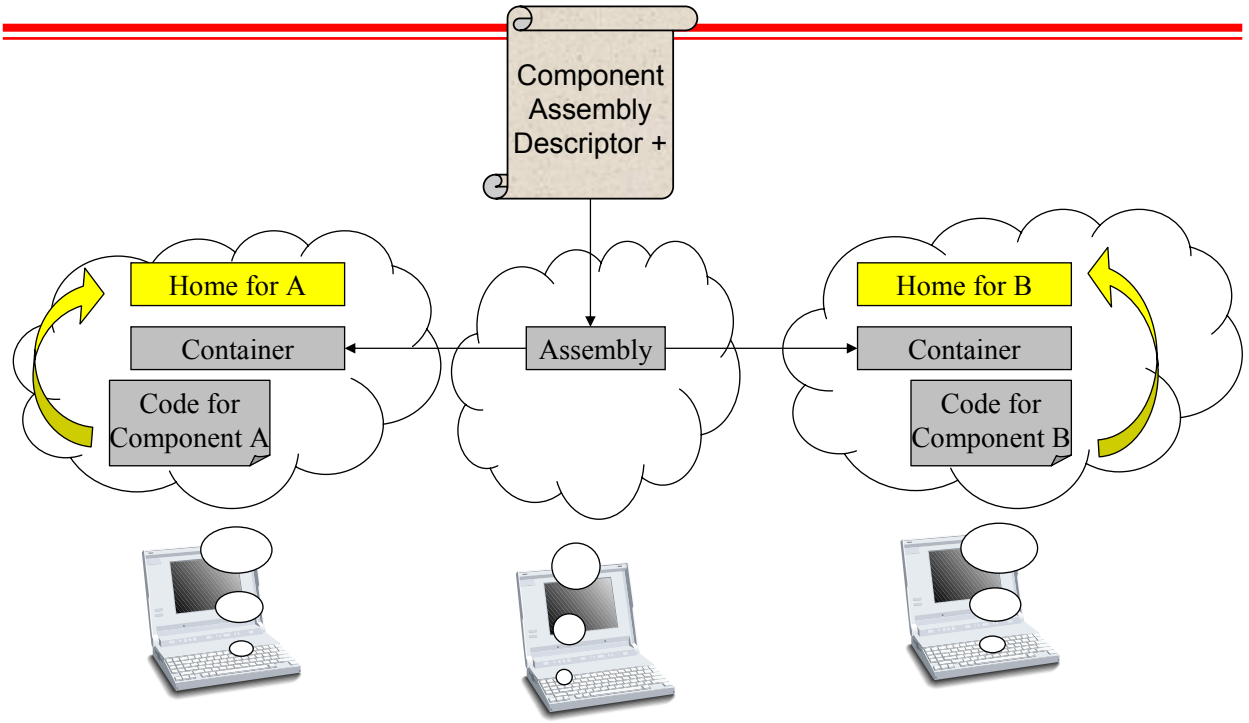
Scénario de déploiement : instanciation des serveurs de composants



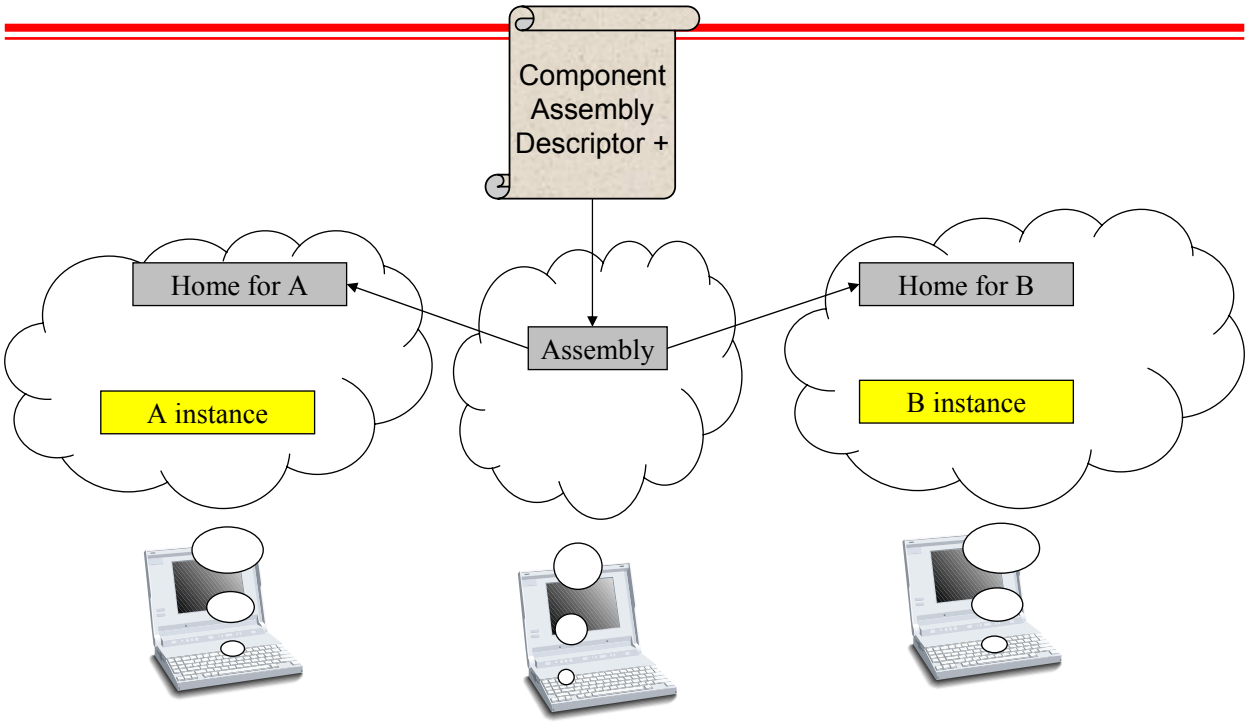
Scénario de déploiement : instanciation des conteneurs



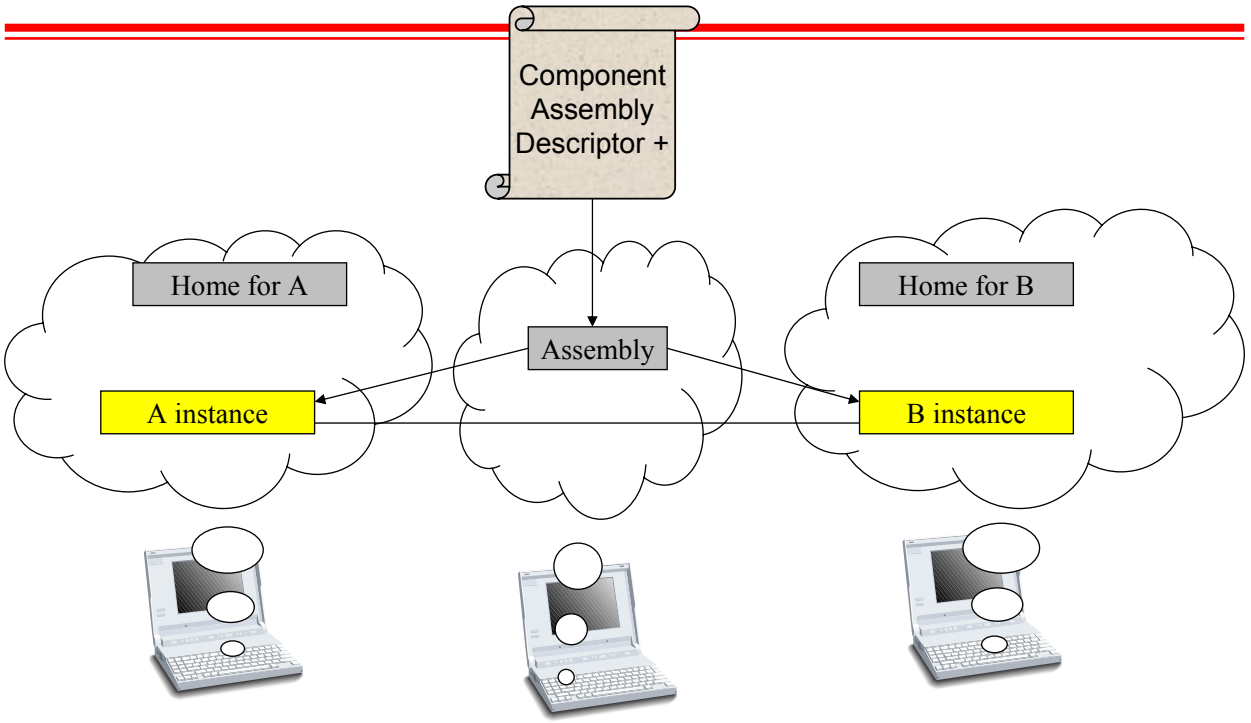
Scénario de déploiement : installation des maisons



Scénario de déploiement : instanciation des composants



Scénario de déploiement : configuration des composants



Le déploiement de l'application Dîner des Philosophes

- **Démarrer l'outil de déploiement**
  - Utilise le ComponentInstallation de chaque site d'exécution pour télécharger les implantations binaires nécessaires
  - Utilise l'AssemblyFactory pour créer un Assembly
  - Invoque l'opération build() de l'instance Assembly
    - Création des serveurs de composants nécessaires
    - Création des conteneurs
    - Installation des maisons
    - Création des instances de composants
    - Interconnexion des ports des composants
    - Invocation configuration_complete() sur chaque composant
- **Déploiement totalement distribué et automatisé**

ObjectWeb
Open Source Middleware

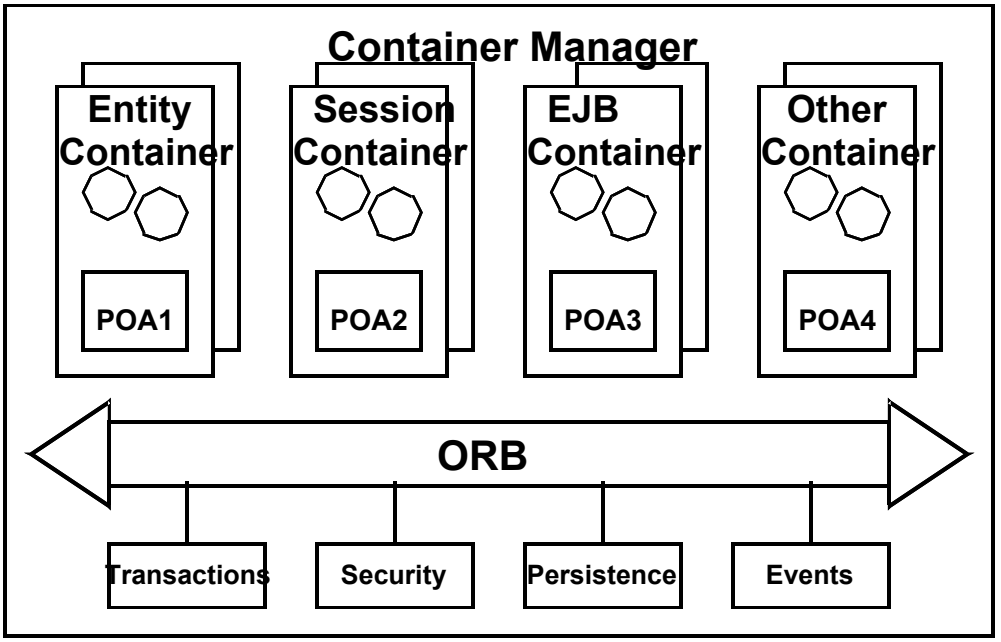
INRIA
RHÔNE-ALPES

Les conteneurs CORBA

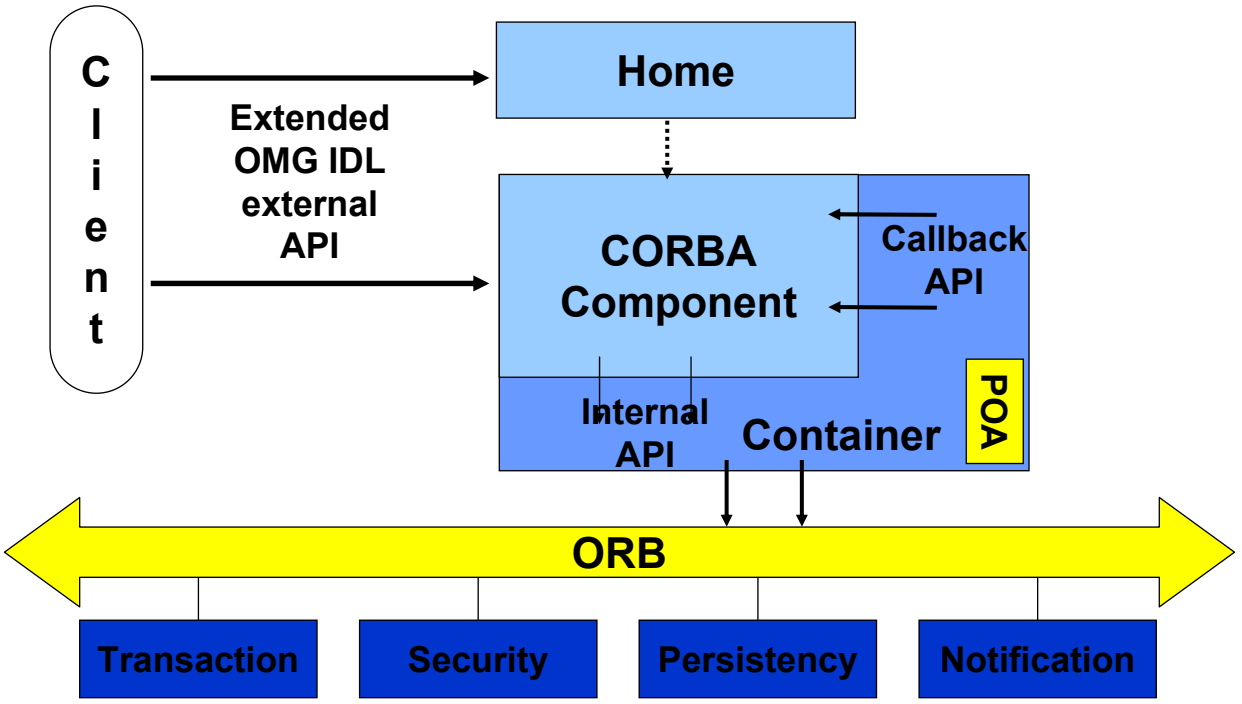
Les conteneurs CCM

- **Un conteneur gère une catégorie de composants**
  - `entity` : persistant, clé primaire et destruction explicite
  - `process` : persistant, pas de clé et destruction explicite
  - `session` : existe durant une session avec le client
  - `service` : existe durant une invocation
  - `EJBsession`, `EJBentity` : support pour EJBs
  - `empty` : autres politiques spécifiques
- **Encapsule un ou des Portable Object Adaptors**
  - (Dé)activation automatique
  - Optimisation des ressources
- **Fournit des interfaces simplifiées pour les services CORBA**
  - Sécurité, transactions et persistance
- **Utilise des appels retour pour la gestion des instances**

L'architecture d'un serveur de conteneurs



L'architecture d'un conteneur



Politiques gérées par les conteneurs

- Décrites via CORBA Component Descriptors (.ccd)
- Implantées par le conteneur, pas par le composant
- Politiques définies pour :
  - Cycle de vie des servants
  - Transaction
  - Sécurité
  - Evénements
  - Persistance

Politiques de cycle de vie des servants

- `method` – valide pour toutes les catégories
  - Activation avant chaque invocation
  - Passivation après chaque invocation
- `transaction` – valide pour toutes sauf `service`
  - Activation avant la 1ière invocation d'une nouvelle transaction
  - Passivation après la dernière invocation de la transaction
- `component` – valide pour toutes sauf `service`
  - Activation avant la 1ière invocation
  - Passivation explicite par le composant
- `container` – valide pour toutes sauf `service`
  - Activation avant la 1ière invocation
  - Passivation quand le conteneur requière de la mémoire

Politiques transactionnelles

- **Définies pour chaque opération si gérées par le conteneur**
  - NOT_SUPPORTED
  - REQUIRED
  - SUPPORTS
  - REQUIRES_NEW
  - MANDATORY
  - NEVER
- **Gérées par le composant via l'API `Components::Transaction::UserTransaction`**
  - Simplification de l'API CosTransactions (OMG OTS)

Politiques de sécurité

- **Définies pour chaque opération si gérées par le conteneur (élément *security* des descripteurs de composants)**
  - **CLIENT_IDENTITY**
  - **SYSTEM_IDENTITY**
  - **SPECIFIED_IDENTITY (=userid)**
- **Les conteneurs contrôlent à l'exécution l'identité de l'appelant et ses droits**
- **Construit au dessus de CORBA Security V2**

ObjectWeb Open Source Middleware

Politiques événementielles

INRIA Rhône-Alpes

- **Comportement transactionnelle de la notification des événements défini pour chaque source**
  - non-transactional
  - default
  - transactional
- **Le conteneur peut utiliser le service de notification CORBA**
  - Modèle push uniquement
  - Projection des eventtypes en Structured Events
  - Création des channels

ObjectWeb Open Source Middleware

Politiques de persistance

INRIA RHÔNE-ALPES

- Uniquement pour composants Process et Entity
- Self managed ou Container managed
- Utilise le service de persistance CORBA (PSS) ou un mécanisme de persistance propriétaire

ObjectWeb
Open Source Middleware

INRIA
RHÔNE-ALPES

Conclusions

Le modèle de composants CORBA

- 1ier standard industriel pour les composants distribués
  - Ouvert, hétérogénéité, portabilité et interopérabilité
  - Un processus d'ingénierie de logiciel à base de composants
  - Un modèle abstrait de composants riche
    - Très proche du modèle de composants UML 2.0
  - Conditionnement, assemblage et déploiement réparti
  - Un canevas de conteneurs
  - Interopérabilité avec EJB
  - Méta modèles prêts pour l'approche Model Driven Architecture (MDA)
- Au coeur de CORBA 3.0
  - Spécification libre ~ 500 pages
- Mais peu d'intérêts de la part des fournisseurs CORBA
  - Ils surfent sur la vague des Web Services

Implantations CCM

- **OpenCCM - ObjectWeb / INRIA & LIFL**
  - Open source Java sur ORBacus 4.1 & OpenORB 1.x & BES 5.x
  - http://openccm.objectweb.org/
- **MicoCCM - FPX & Alcatel**
  - Open source C++ sur MICO
  - http://www.fpx.de/MicoCCM/
- **Qedo - IST COACH**
  - Fraunhofer FOKUS & Humboldt University
  - Open source C++ sur MICO & ORBacus 4.1 (& TAO)
  - http://qedo.berlios.de
- **EJCCM - CPI Inc.**
  - Semi open source Java sur OpenORB 1.x
  - http://www.ejccm.org
- **K2 - ICMG**
  - Produit commercial C++ sur divers ORBs
  - http://www.icmgworld.com
- **Quelques autres implantations moins connues**

Activités OMG autour du CCM

- **Components 1.2 Revision Task Force (RTF)**
  - Révision de la spécification CORBA Components
- **Deployment and Configuration FTF**
  - Modèle de conditionnement, d'assemblage et de déploiement plus sophistiqué
  - OMG TC Document ptc/03-07_02 & 08
- **Soumission Fraunhofer FOKUS/IK++ pour MOF 2.0 IDL RFP**
  - Référentiels MOF construits en composants CORBA
  - RFP = OMG TC Document ad/01-11-07
  - Subm. = OMG TC Document ad/02-12-05
- **UML Profile for CORBA Components RFP**
  - Extension du profil UML/CORBA pour composants CORBA
  - RFP = OMG TC Document ab/02-10-01
  - Subm. = OMG TC Document mars/03-05-09
- **Lightweight CCM RFP**
  - Simplication du CCM pour systèmes embarqués
  - RFP = OMG TC Document realtime/02-11-27
  - Subm. = OMG TC Document realtime/03-05-05
- **Streams for CORBA Components RFP**
  - Nouveaux types de ports pour communication par flux
  - RFP = OMG TC Document mars/03-06-11
- **QoS for CORBA Components RFP**
  - Prise en compte de qualités de services dans le CCM
  - RFP = OMG TC Document mars/03-06-12

Références sur le modèle de composants CORBA

- **CORBA 3 Fundamentals and Programming**
  - Dr. John Siegel, publié chez John Wiley and Sons
- **CORBA/IIOP Specification version 3.0.2**
  - OMG TC Document formal/2002-12-06
- **CORBA Components Specification**
  - OMG TC Document formal/2002-06-65 & ptc/2002-08-03
- **CORBA Component Model Tutorial**
  - OMG TC Document ccm/2002-06-01
- **"The CCM Page", Diego Sevilla Ruiz**
  - *http://www.ditec.um.es/~dsevilla/ccm/*
- **IST COACH Project**
  - *http://www.ist-coach.org*

ObjectWeb
Open Source Middleware

Pas d'intergiciel universel
pour composants !

INRIA
RHÔNE-ALPES

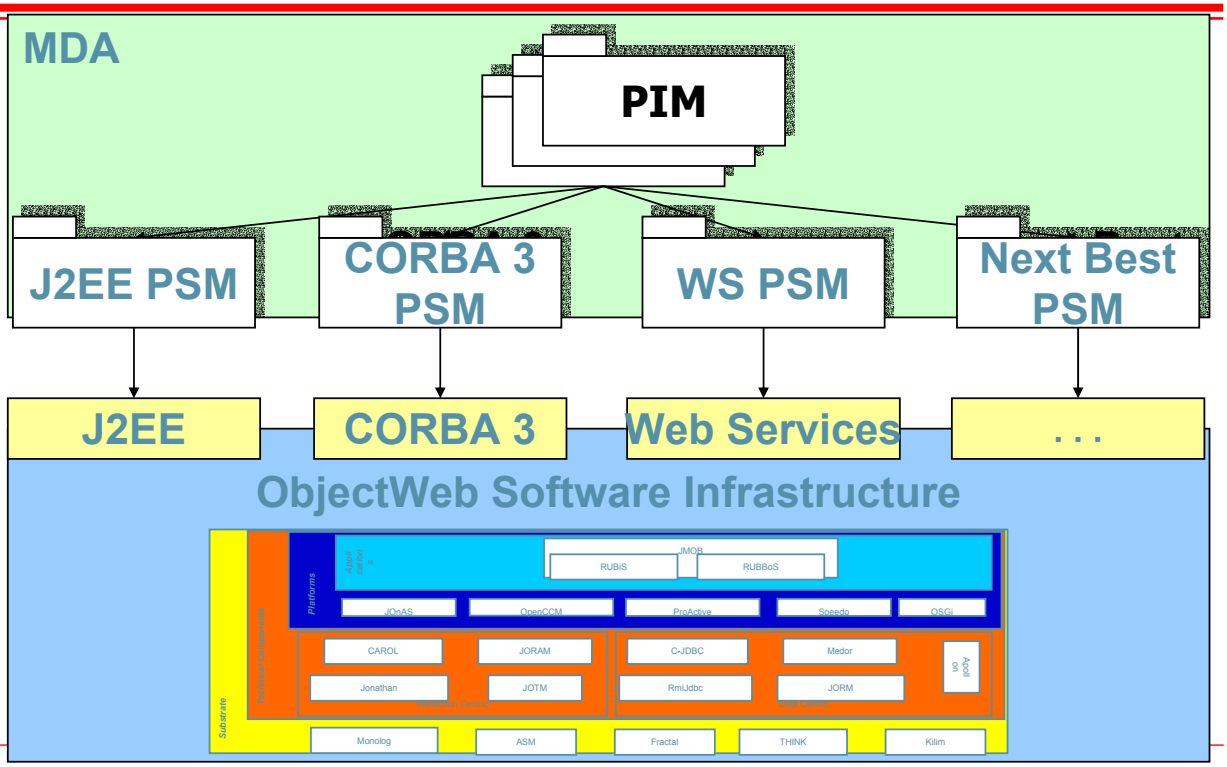
Composants Distribués

OMG
MDA
?

Enjeu majeur !

CORBA	EJB	.NET	. . .

Pas d'intergiciel universel pour composants !



Merci