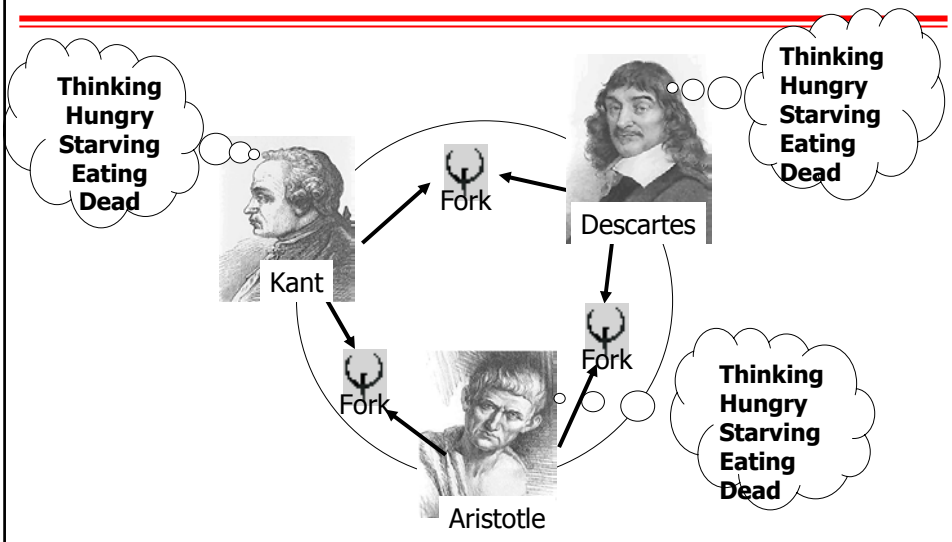
ICAR'03

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

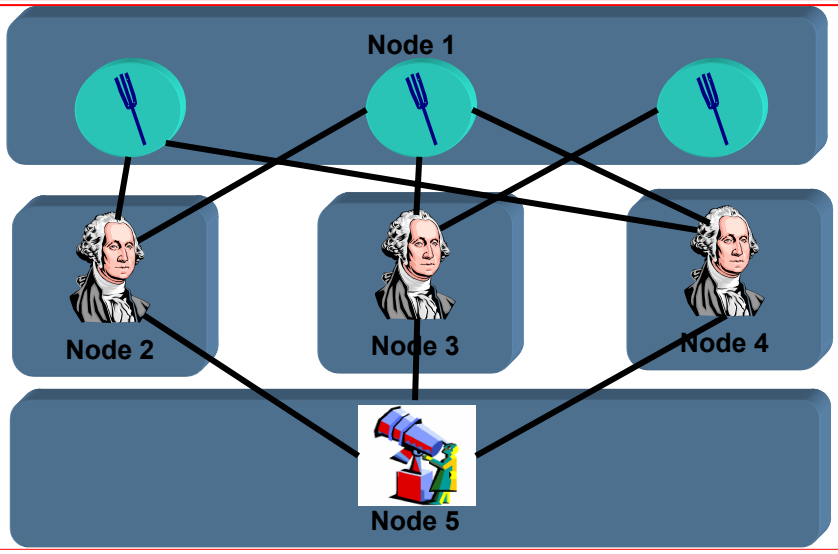
Etude de cas d'une application construite avec CCM

Philippe Merle
Projet Jacquard (INRIA et LIFL)
http://www.lifl.fr/~merle

L'application du dîner des philosophes



L'application à l'exécution



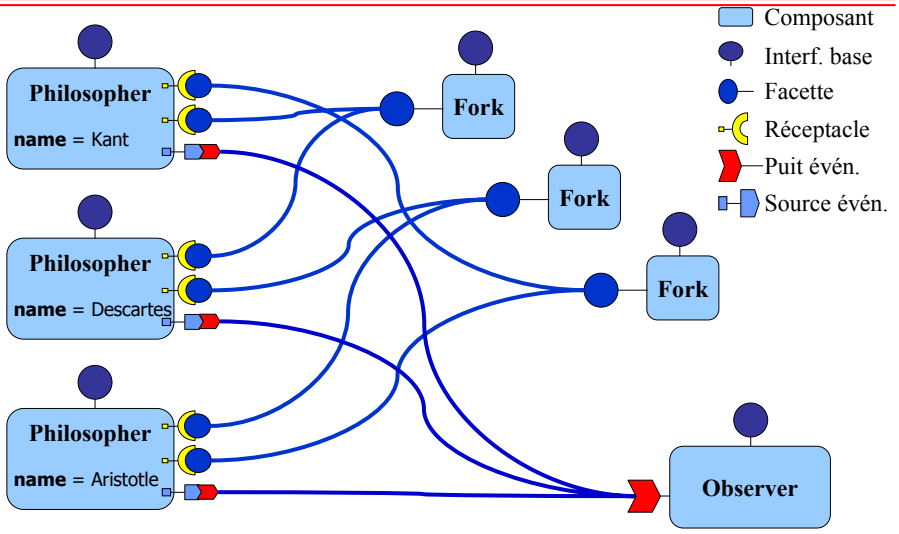
Plan

- La conception des composants CORBA
  - Notations : graphique, profile UML 1.x pour CCM et OMG IDL 3.0
- L'utilisation des composants CORBA
  - Projection OMG IDL 3.0 vers OMG IDL ; code client
- Le développement des composants CORBA
  - Notation OMG CIDL ; classes Java d'implantation
- Le conditionnement et l'assemblage des composants CORBA
  - Notation XML CCM ; exemples d'outils
- Le déploiement, l'exécution et l'administration de l'application
  - Démonstration sur la plate-forme OpenCCM
- Conclusion

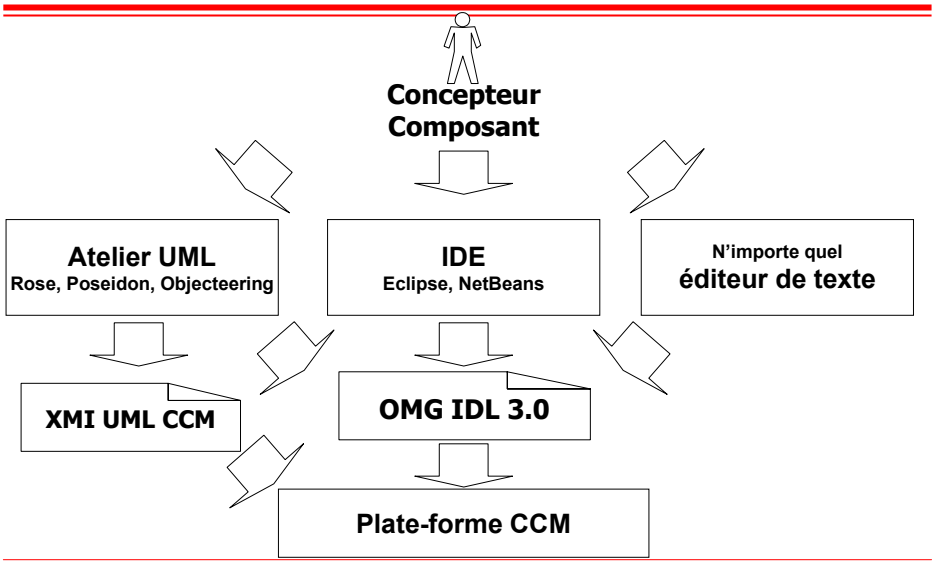
La conception des composants CORBA

- **Concevoir des composants CORBA = identifier les**
  - Types de composants et leurs ports
  - Interfaces et événements d'interaction (~ typage des ports)
  - Types de maisons de composants et leurs opérations
  - Types de données et exceptions utilitaires
- **Via une notation**
  - Graphique informelle
  - Profil UML 2.0 pour CCM (non existant)
  - Profil UML 1.x pour CCM (en cours de standardisation OMG)
  - OMG IDL 3.0 défini dans CORBA 3.0
- **Cependant une plate-forme CCM « classique » ne comprend que la notation OMG IDL 3.0**
  - Toutefois OpenCCM compile aussi des fichiers XMI UML CCM

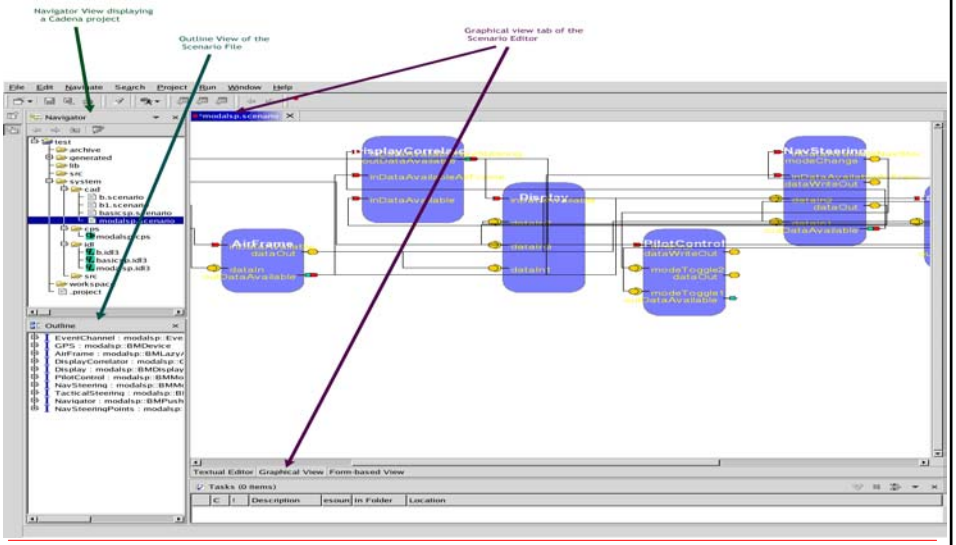
Le dîner des philosophes notation graphique informelle



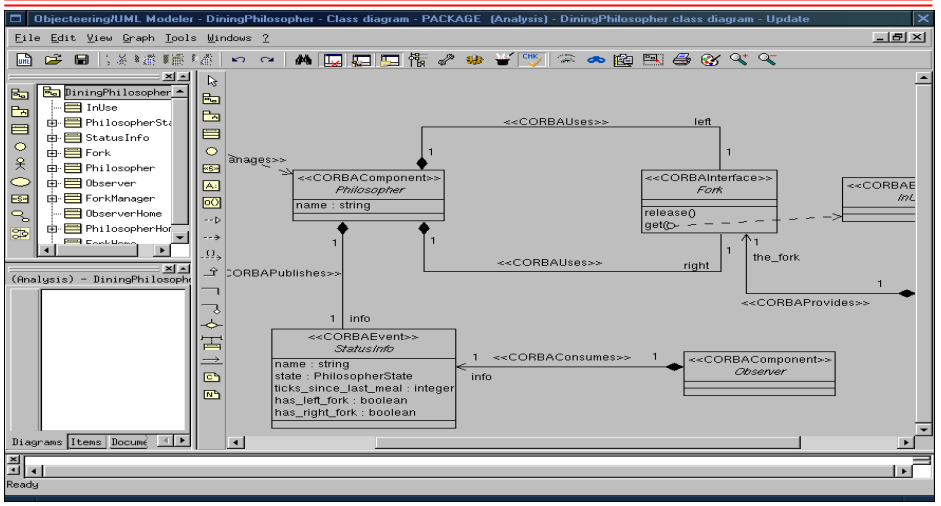
Des outils pour la conception des composants CORBA



L'atelier Cadena
Plug-in Eclipse + OpenCCM

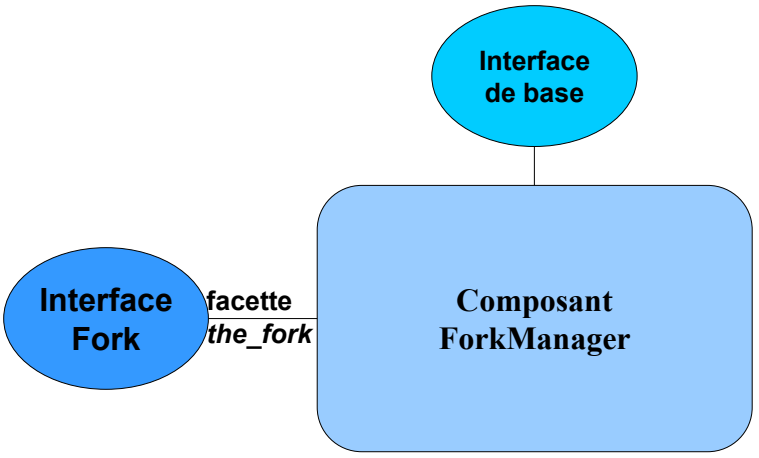


L'atelier Objecteering
avec profil UML pour CCM

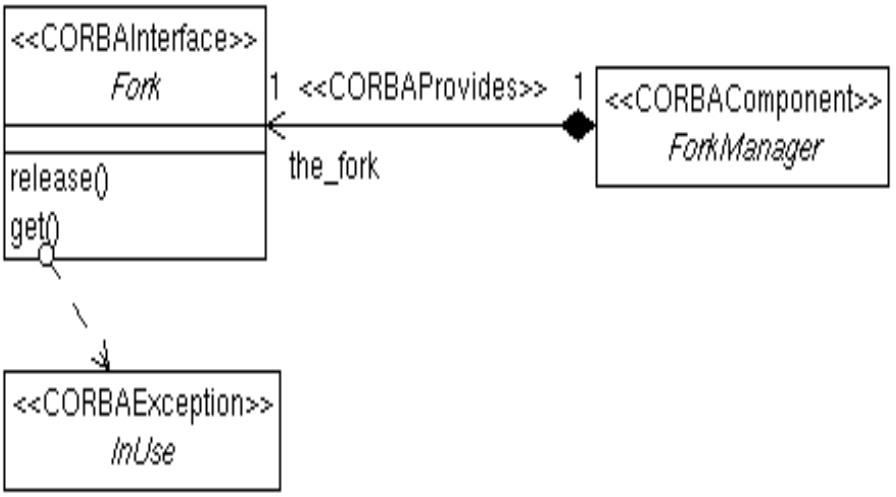


- **Développement LIFL dans projet RNTL ACCORD**

Le composant ForkManager
notation graphique informelle

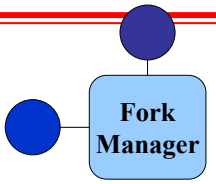


Le composant ForkManager
notation profil UML 1.x pour CCM



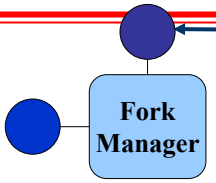
Le composant ForkManager notation OMG IDL 3.0

```
exception InUse {};
interface Fork
{
   void get() raises (InUse);
   void release();
};
// Le composant fourchette.
component ForkManager
{
   // La facette fourchette utilisée par les philosophes.
   provides Fork the_fork;
};
```



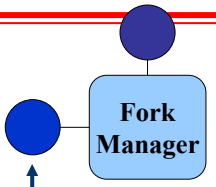
Le composant ForkManager notation OMG IDL 3.0

```
exception InUse {};
interface Fork
{
   void get() raises (InUse);
   void release();
};
// Le composant fourchette.
component ForkManager
{
   // La facette fourchette utilisée par les philosophes.
   provides Fork the_fork;
};
```

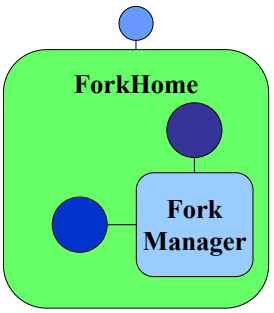


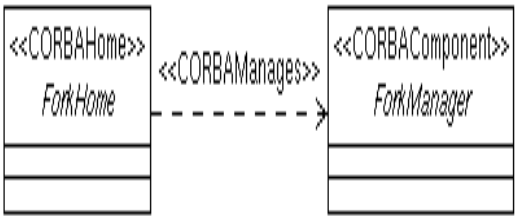
Le composant ForkManager notation OMG IDL 3.0

```
exception InUse {};
interface Fork
{
   void get() raises (InUse);
   void release();
};
// Le composant fourchette.
component ForkManager
{
   // La facette fourchette utilisée par les philosophes.
   provides Fork the_fork;
};
```



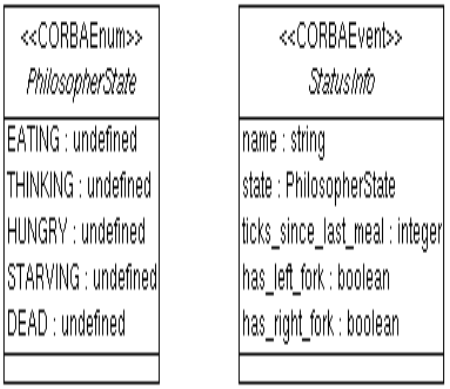
La maison de ForkManager selon les 3 notations






```
// OMG IDL 3.0
home ForkHome manages ForkManager {};
```

L'événement StatusInfo notations UML / CCM et OMG IDL 3.0

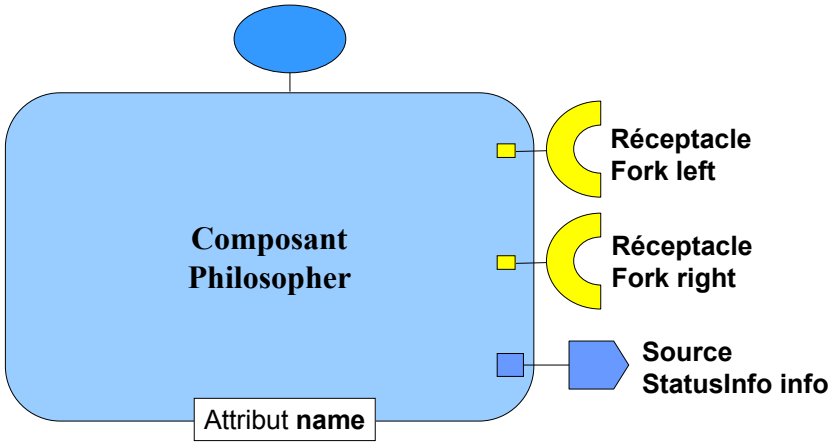


profil UML pour CCM

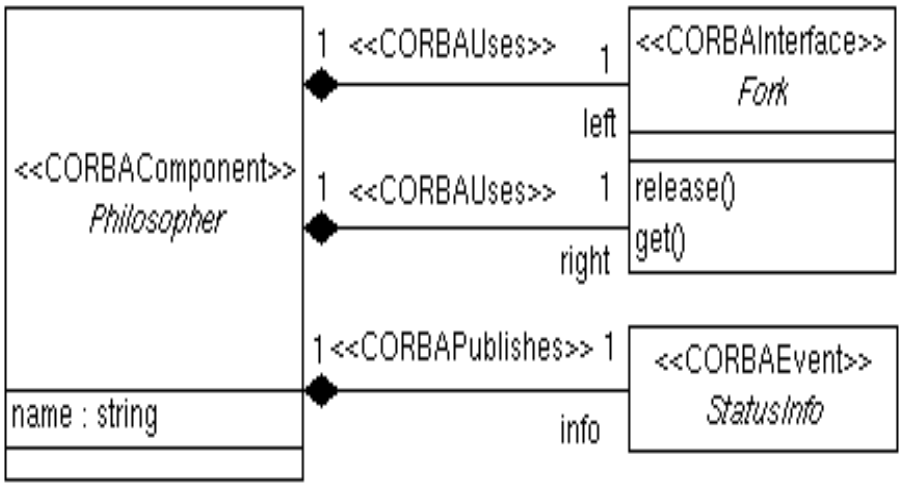
```
enum PhilosopherState {
  EATING, THINKING,
  HUNGRY, STARVING, DEAD
};

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;
};
```

OMG IDL 3.0

Le composant Philosopher notation graphique informelle

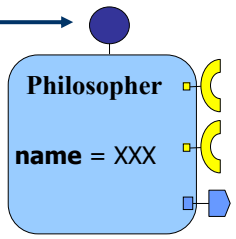


Le composant Philosopher
notation profil UML 1.x pour CCM



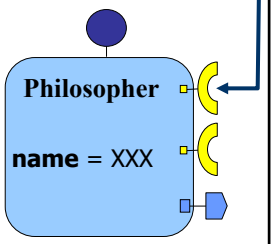
Le composant Philosopher
notation OMG IDL 3.0

```
component Philosopher
{
   attribute string name;

   // La fourchette à gauche.
   uses Fork left;

   // La fouchette à droite.
   uses Fork right;

   // La source d'événements StatusInfo.
   publishes StatusInfo info;
};
```



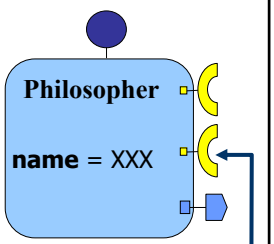
Le composant Philosopher notation OMG IDL 3.0

```
component Philosopher
{
   attribute string name;

   // La fourchette à gauche.
   uses Fork left;

   // La fouchette à droite.
   uses Fork right;

   // La source d'événements StatusInfo.
   publishes StatusInfo info;
};
```



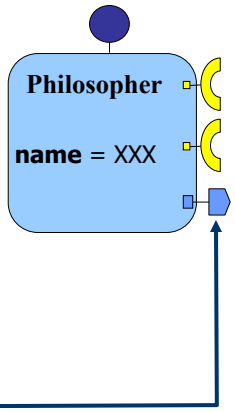
Le composant Philosopher notation OMG IDL 3.0

```
component Philosopher
{
   attribute string name;

   // La fourchette à gauche.
   uses Fork left;

   // La fouchette à droite.
   uses Fork right;

   // La source d'événements StatusInfo.
   publishes StatusInfo info;
};
```

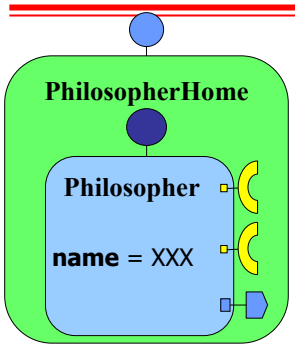


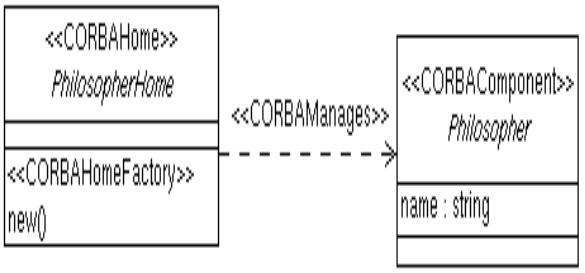
Le composant Philosopher notation OMG IDL 3.0

```
component Philosopher
{
   attribute string name;

   // La fourchette à gauche.
   uses Fork left;

   // La fouchette à droite.
   uses Fork right;

   // La source d'événements StatusInfo.
   publishes StatusInfo info;
};
```



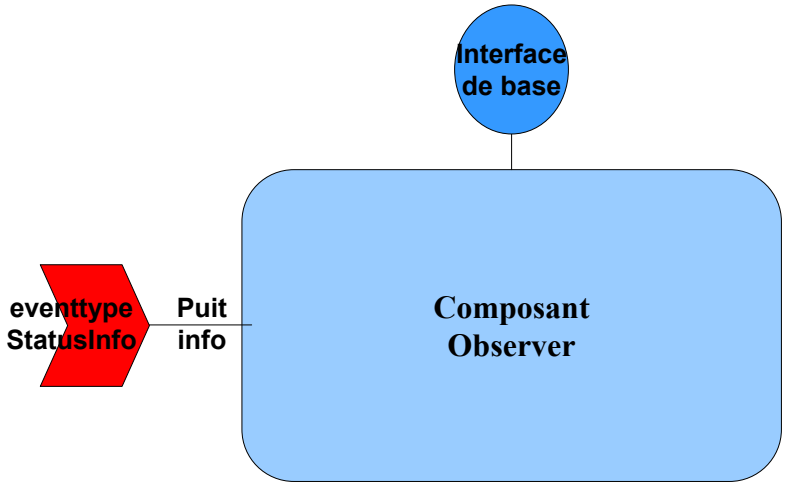
La maison de Philosopher selon les 3 notations



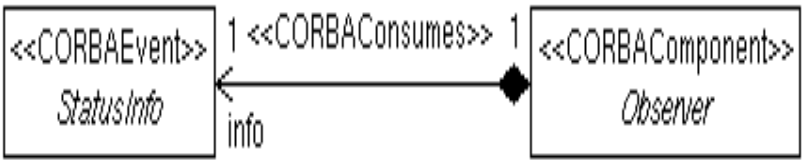



```
// OMG IDL
home PhilosopherHome manages Philosopher {
   factory new(in string name);
};
```

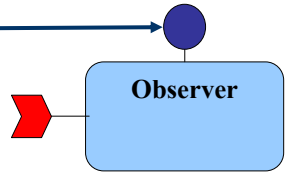
Le composant Observer
notation graphique informelle



Le composant Observer
notation profil UML 1.x pour CCM

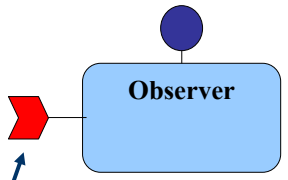


Le composant Observer notation OMG IDL 3.0



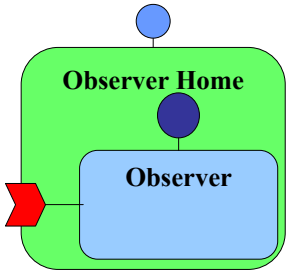

```
component Observer
{
  // Le puit de réception de StatusInfo.
  consumes StatusInfo info;
};
```


Le composant Observer notation OMG IDL 3.0

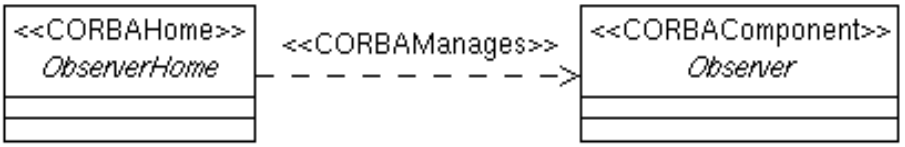



```
component Observer
{
  // Le puit de réception de StatusInfo.
  consumes StatusInfo info;
};
```

La maison de Observer selon les 3 notations




```
// OMG IDL
home ObserverHome
manages Observer {};
```

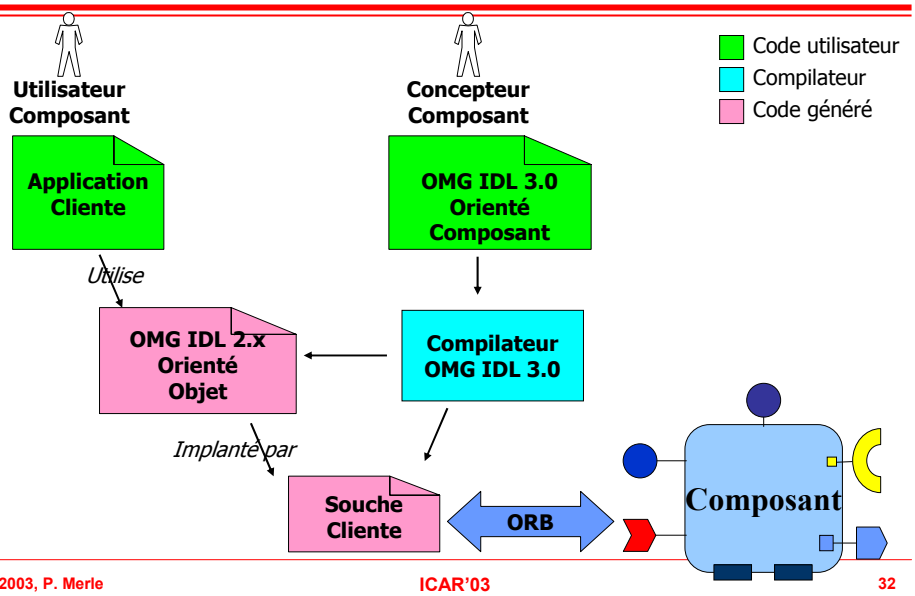


L'utilisation des composants CORBA

L'utilisation des composants CORBA

- « OMG IDL orienté composant » uniquement sucre syntaxique pour décrire les composants CORBA, projeté en OMG IDL 2.x
  - Chaque construction OMG IDL 3.0 a une équivalence en OMG IDL 2.x
  - Une seule projection spécifiée au lieu d'une par langages de programmation
  - Préservation des standards de projection OMG IDL 2.x → langages
- Les composants et maisons sont utilisés par les développeurs comme des objets CORBA étendus
  - Composants et maisons invoqués comme des objets CORBA
  - Applications clientes uniquement CORBA 2.x possible
  - Préservation des compétences des développeurs CORBA 2.x
  - Réutilisation des outils CORBA 2.x, e.g. générateurs de souches/squelettes

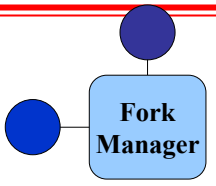
Le projection OMG IDL 3.0 vers OMG IDL 2.x



Les principales règles de projection OMG IDL 3.0 vers OMG IDL

- **1 type de composants → 1 interface**
  - **héritant de** `Components::CCMObject`
  - **1 propriété → 1 attribut**
  - **1 facette ou puits → 1 opération fournissant la référence du port**
  - **1 réceptacle → opérations de connexion, déconnexion et d'acquisition de la (ou des) référence(s) associée(s)**
  - **1 source → opérations de (dé)souscription aux événements produits**
- **1 type de maisons →3 interfaces**
  - **1 pour opérations explicites du concepteur**
    - **+ héritage de** `Components::CCMHome`
  - **1 pour opérations implicites générées**
  - **1 héritant des 2 interfaces précédentes**
- **1 type d'événements → 1 type de valeurs + 1 interface Consumer**
  - **héritant de** `Components::EventBase`
  - **héritant de** `Components::EventConsumerBase`

La projection OMG IDL du composant ForkManager

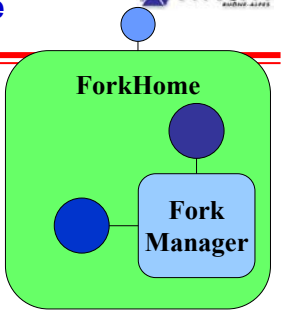



```
component ForkManager
{
  provides Fork the_fork;
};
```

Traduit en

```
interface ForkManager :
          ::Components::CCMObject
{
    Fork provide_the_fork();
};
```

La projection OMG IDL de la maison ForkHome




```
home ForkHome
     manages ForkManager {};
```

Traduit en

```
interface ForkHomeExplicit :
          ::Components::CCMHome {};
interface ForkHomeImplicit :
          ::Components::KeylessCCMHome {
  ForkManager create();
};
interface ForkHome :
          ForkHomeExplicit,
          ForkHomeImplicit {};
```


La projection OMG IDL de l'événement StatusInfo

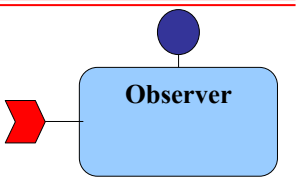
```
eventtype StatusInfo { . . . };
```

Traduit en

```
valuetype StatusInfo :
          ::Components::EventBase { . . . };

interface StatusInfoConsumer :
          ::Components::EventConsumerBase {
  void push_StatusInfo(in StatusInfo
                       the_StatusInfo);
};
```

La projection OMG IDL du composant Observer

```
component Observer {
   consumes StatusInfo info;
};
```

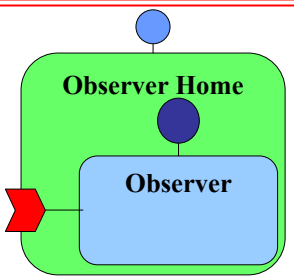


Traduit en

```
interface Observer :
          ::Components::CCMObject {
  StatusInfoConsumer get_consumer_info();
};
```


La projection OMG IDL de la maison ObserverHome

```
home ObserverHome
     manages Observer {};
```

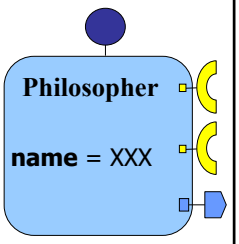


Traduit en

```
interface ObserverHomeExplicit :
          ::Components::CCMHome {};
interface ObserverHomeImplicit :
          ::Components::KeylessCCMHome {
  Observer create();
};
interface ObserverHome :
           ObserverHomeExplicit,
           ObserverHomeImplicit {};
```

La projection OMG IDL du composant Philosopher

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

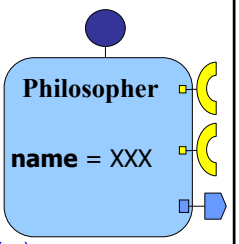


Traduit en

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

   .../...
```


La projection OMG IDL du composant Philosopher

Traduit en



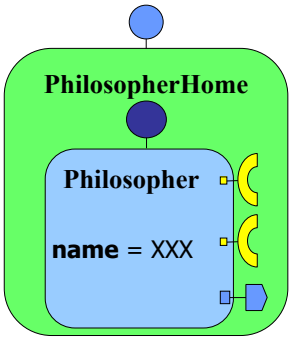

```
   void connect_left(in Fork cnx) raises(…)
   Fork disconnect_left() raises(…);
   Fork get_connection_left();

   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(…);
};
```

La projection OMG IDL de la maison PhilosopherHome

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

Traduit en

```
interface PhilosopherHomeExplicit :
          ::Components::CCMHome {
  Philosopher new(in string name);
};
interface PhilosopherHomeImplicit :
          ::Components::KeylessCCMHome {
  Philosopher create();
};
interface PhilosopherHome :
          PhilosopherHomeExplicit,
          PhilosopherHomeImplicit {};
```



Le modèle de programmation des clients

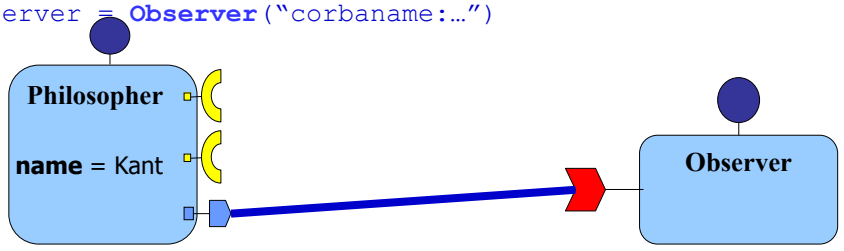
- **Deux canevas de conception**
  - Factory – Recherche une maison et l'utilise pour créer de nouvelles instances de composant
  - Finder – Recherche instances de composant existantes via le service de Nommage, de Courtage ou via les opérations de recherche des maisons
- **Optionnellement démarcation des transactions**
  - `begin`, `commit` et `rollback`
- **Peut établir les crédits de sécurité**
- **Invoque les opérations des instances de composant**
  - Celles définies par la projection vers OMG IDL
  - Client ne sait pas nécessairement qui interagit avec des composants

L'utilisation de composants CORBA via OMG IDLscript

```
# Obtenir le service de recherche de maisons de composants.
chf = CORBA.ORB.resolve_initial_references
                    ("ComponentHomeFinder")
# Rechercher une maison selon son type.
forkHome = chf.find_home_by_type(ForkHome.id())
# Créer un composant fourchette.
forkManager = forkHome.create()
# Obtenir la facette fourchette.
fork = forkManager.provide_the_fork()
# Utiliser la facette fourchette.
fork.get()
. . . . . .
fork.release()
```


L'interconnexion de composants CORBA avec OMG IDLscript

```
# Obtenir les composants CORBA à interconnecter.
kant = Philosopher("corbaname:…")
observer = Observer("corbaname:…")
```




```
# Connecter les composants kant et observer.
ck = kant.subscribe_info(observer.get_consumer_info())
. . . . . .
# Déconnecter les composants kant et observer.
kant.unsubscribe_info(ck)
```

La navigation et l'introspection des composants CORBA

- **Navigation d'une facette vers le composant via opération** `CORBA::Object::get_component()`
  - **Retourne la référence de base du composant ou nil si pas une facette**
- **Navigation du composant vers les facettes et puits via les opérations introduites lors de la projection vers OMG IDL**
- **Opérations génériques de navigation, de contrôle et d'introspection fournies par l'interface** `CCMObject`
  - `provide_facet`, `get_[all|named]_facets`, `same_component`
  - `connect`, `disconnect`, `get_connections`, `get_[all|named]_receptacles`
  - `get_consumer`, `get_[all|named]_consumers`
  - `subscribe`, `unsubscribe`, `[dis]connect_consumer`, `get_[all|named]_[emitters|publishers]`
  - `get_all_ports`, `get_ccm_home`, `get_component_def`, `get_ccm_home`, `get_primary_key`

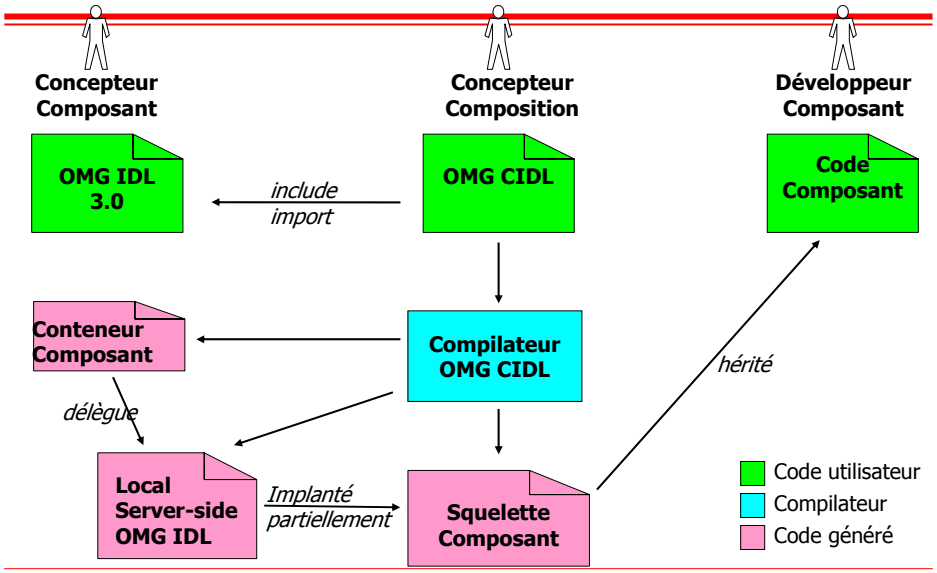
➔ **Au cœur de l'interconnexion des composants au déploiement et de l'administration des composants à l'exécution**

Le développement des composants CORBA

Le développement des composants CORBA

- Décrire la structure d'implantation des composants
  - Via OMG Component Implementation Definition Language (CIDL)
  - Compositions, segments et états persistants
- Compiler les descriptions OMG CIDL
  - Génération du code conteneur + squelette exécuteur
- Implanter les composants et les maisons
  - Héritage du code généré
  - Règles de programmation à respecter
- Compiler le code utilisateur + le code généré

La compilation OMG CIDL



L'implantation Java des composants

- **Héritage des squelettes OMG CIDL générés**
  - 1 classe par maison implantant
    - Opérations métiers explicites du concepteur
    - 1 méthode de fabrique des implantations des segments
    - 1 méthode statique point d'entrée pour le déploiement
  - 1 classe par composant implantant
    - Opérations métiers des facettes supportées
    - 1 opération de réception des événements pour chaque puit supporté
    - Surcharge des opérations de l'interface *`SessionComponent`* ou *`EntityComponent`*
  - 1 classe par segment implantant
    - Opérations métiers des facettes supportées
    - 1 opération de réception des événements pour chaque puit supporté
- **2 classes par événement**
  - Implantation concrète et fabrique

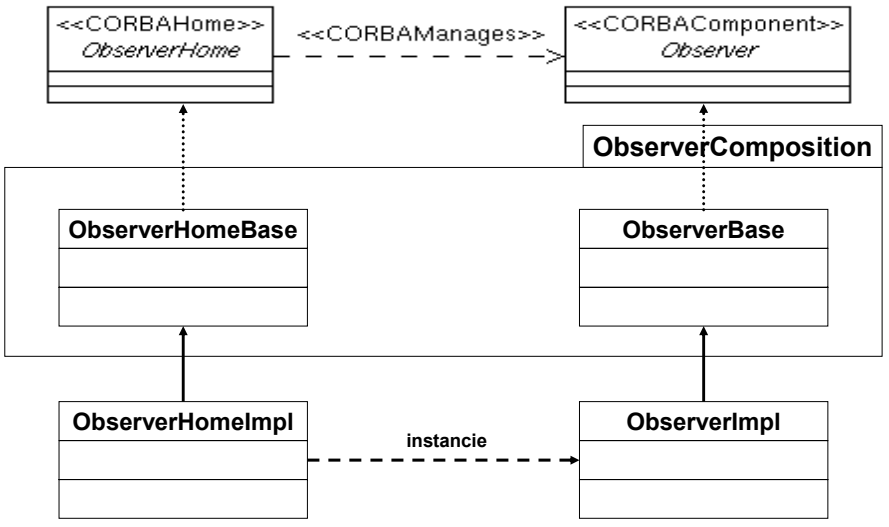
L'interface SessionComponent

```
module Components {
  local interface EnterpriseComponent
  {
    void configuration_complete()
         raises(InvalidConfiguration);
  };
  local interface SessionComponent : EnterpriseComponent
  {
    void set_session_context(in SessionContext ctx)
         raises(CCMException);
    void ccm_activate()
         raises(CCMException);
    void ccm_passivate()
         raises(CCMException);
    void ccm_remove()
         raises(CCMException);
};};
```

La composition OMG CIDL pour composant Observer

```
composition session ObserverComposition
{
    home executor ObserverHomeBase
    {
        implements DiningPhilosophers::ObserverHome;
        manages ObserverBase;
    };
};
```

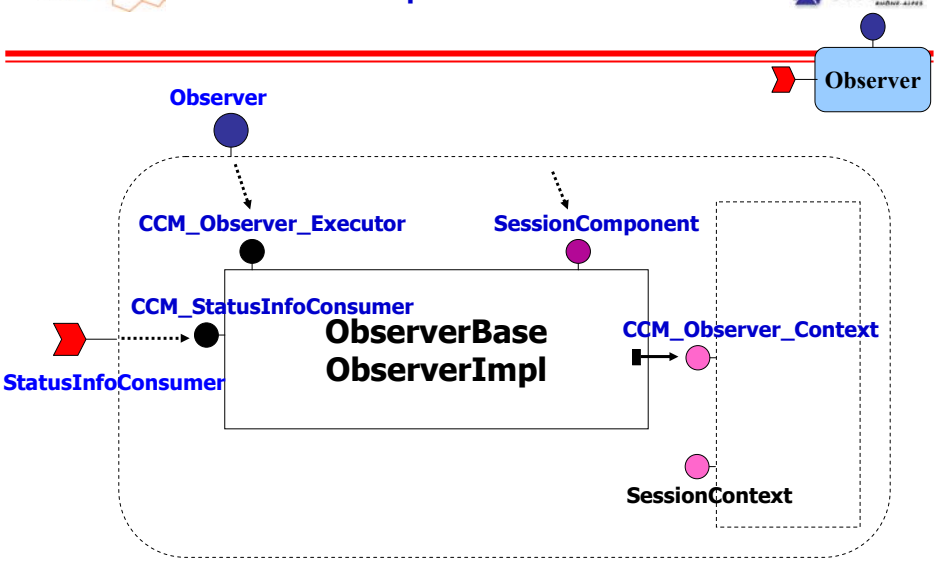

La structure d'implantation du composant Observer



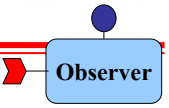
L'implantation Java de la maison ObserverHome

```
public class ObserverHomeImpl
     extends ObserverComposition.ObserverHomeBase
{
  // Le constructeur.
  public ObserverHomeImpl() {}

  // La fabrique des implantations de segments.
  public org.omg.Components.ExecutorSegmentBase
  create_executor_segment(int segid)
  { return new ObserverImpl(); }

  // Le point d'entrée pour le déploiement.
  public static org.omg.Components.HomeExecutorBase
  create_home()
  { return new ObserverHomeImpl(); }
}
```


Le composant Observer



L'implantation Java du composant Observer (1/2)




```
public class ObserverImpl
     extends ObserverComposition.ObserverBase
{
  // Attributs internes pour IHM.

  // Le constructeur.
  public ObserverImpl() { ... }

  // Pour interface CCM_StatusInfo_Consumer.
  public void
  push(StatusInfo event)
  {
    ... Mise à jour IHM ...
  }
../..
```

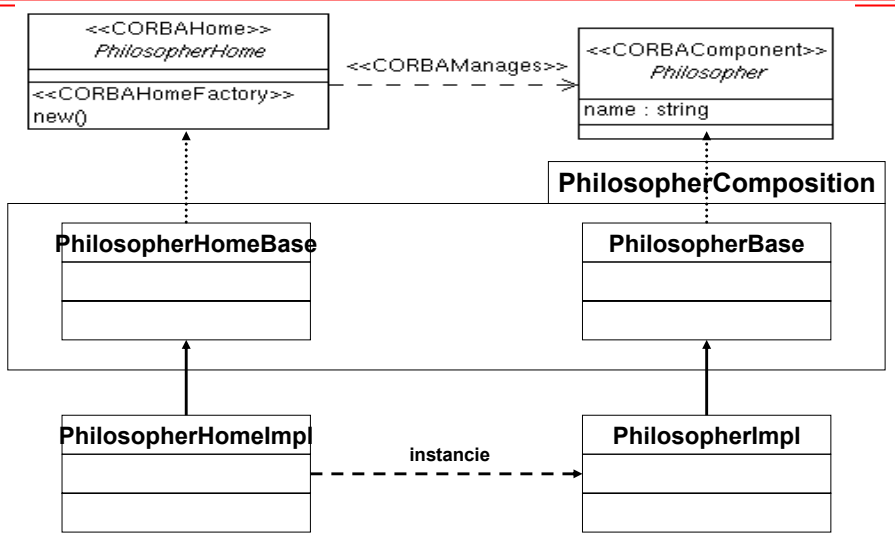

L'implantation Java du composant Observer (2/2)


```
  // Surcharge des opérations SessionComponent.

  public void ccm_activate()
  throws org.omg.Components.CCMException
  { ... Afficher IHM ... }

  public void ccm_passivate()
  throws org.omg.Components.CCMException
  { ... Cacher IHM ... }

  public void ccm_remove ()
  throws org.omg.Components.CCMException
  { ... Libérer les ressources IHM ... }
}
```

La composition OMG CIDL pour composant Philosopher

```
composition session PhilosopherComposition
{
  home executor PhilosopherHomeBase
  {
    implements DiningPhilosophers::PhilosopherHome;
    manages PhilosopherBase;
  };
};
```

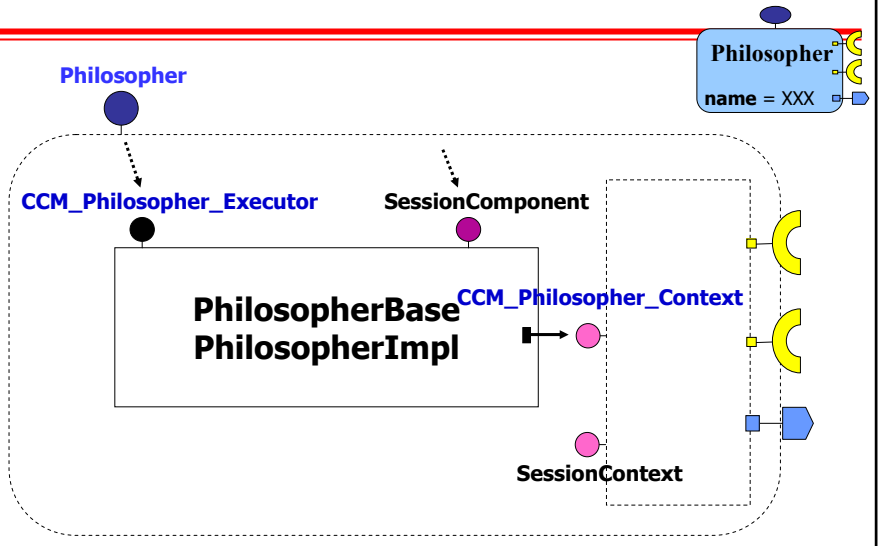

La structure d'implantation du composant Philosopher



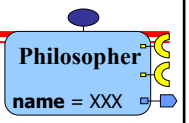
L'implantation Java de la maison PhilosopherHome

```
public class PhilosopherHomeImpl
     extends PhilosopherComposition.PhilosopherHomeBase
{ // Le constructeur.
  public PhilosopherHomeImpl() {}

  // Pour CCM_PhilosopherHome
  public org.omg.Components.EnterpriseComponent
  _new(String name)
  { return new PhilosopherImpl(name); }

  // Fabrique des implantations de segments.
  public org.omg.Components.ExecutorSegmentBase
  create_executor_segment(int segid)
  { return new PhilosopherImpl(null); }

  // Le point d'entrée pour le déploiement.
  public static org.omg.Components.HomeExecutorBase
  create_home()
  { return new PhilosopherHomeImpl(); }
}
```


Le composant Philosopher

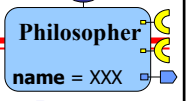


Les interfaces de l'exécuteur Philosopher



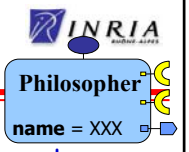

```
local interface CCM_Philosopher_Executor :
                ::Components::EnterpriseComponent
{
  attribute string name;
};
local interface CCM_Philosopher_Context :
  ::Components::CCMContext
{
  // Obtenir la fourchette gauche actuellement connectée.
  Fork get_connection_left();

 // Obtenir la fourchette droite actuellement connectée.
  Fork get_connection_right();

  // Envoyer un événement à tous les puits connectés.
  void push_info(in StatusInfo ev);
};
```


L'implantation Java du composant Philosopher (1/4)



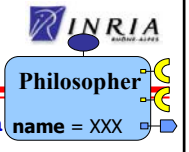

```
public class PhilosopherImpl
     extends PhilosopherComposition.PhilosopherBase
  implements java.lang.Runnable
{
  // Attribut pour stocker le nom du philosophe.
  private String name_;

  // Le constructeur.
  public PhilosopherImpl(String name) {name_=name;}

  // Pour interface CCM_Philosopher_Executor.
  public String name() { return name_; }
  public void name(String n) { name_ = n; }

../..
```

L'implantation Java du composant Philosopher (2/4)



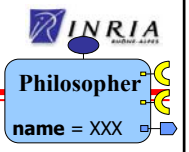

```
// Surcharge des opérations EnterpriseComponent.

public void configuration_complete()
throws org.omg.Components.InvalidConfiguration
{
  // Vérifier si la configuration est correcte.
  if( (name_ == null)
   || (get_context().get_connection_left()==null)
   || (get_context().get_connection_right()==null))
  {
    throw new
        org.omg.Components.InvalidConfiguration();
} }
```


L'implantation Java du composant Philosopher (3/4)




```
// Surcharge des opérations SessionComponen

// L'activité associée au philosophe.
private java.lang.Thread comportement_;

public void ccm_activate()
throws org.omg.Components.CCMException
{
  comportement_ = new Thread(this);
  comportement_.start();
}

public void ccm_remove()
throws org.omg.Components.CCMException
{ comportement_.stop(); }
```

L'implantation Java du composant Philosopher (4/4)

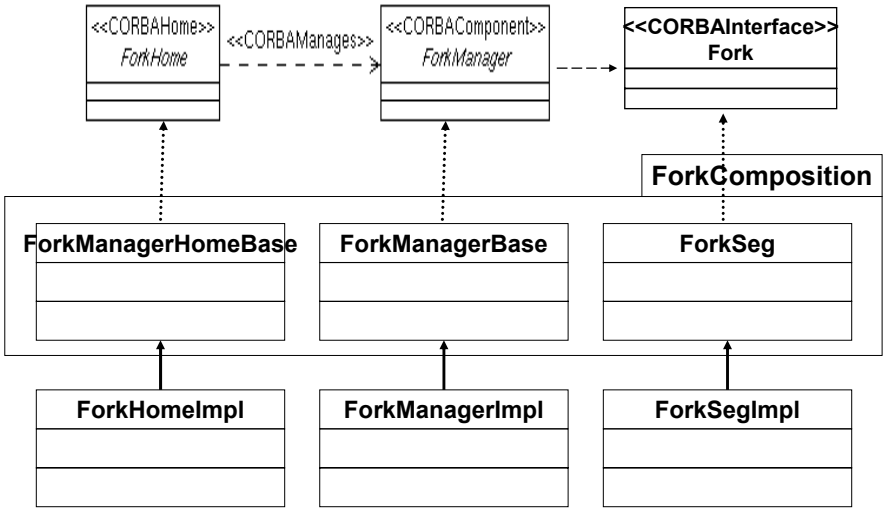



```
// Le comportement du philosophe.
public void run()
{
  ...
  // Envoyer son état aux observateurs.
  get_context().push_info(new StatusInfoImpl(...));
  ...
  // Prendre les fourchettes.
  get_context().get_connection_left().get();
  get_context().get_connection_right().get();
  ...
  // Rendre les fourchettes.
  get_context().get_connection_left().release();
  get_context().get_connection_right().release();
  ...
} }
```


La composition OMG CIDL pour composant ForkManager

```
composition session ForkComposition
{
  home executor ForkManagerHomeBase
  {
    implements DiningPhilosophers::ForkHome;
    manages ForkManagerBase
    {
      segment ForkSeg
      {
        provides facet the_fork;
      }
    };
  };
};
```

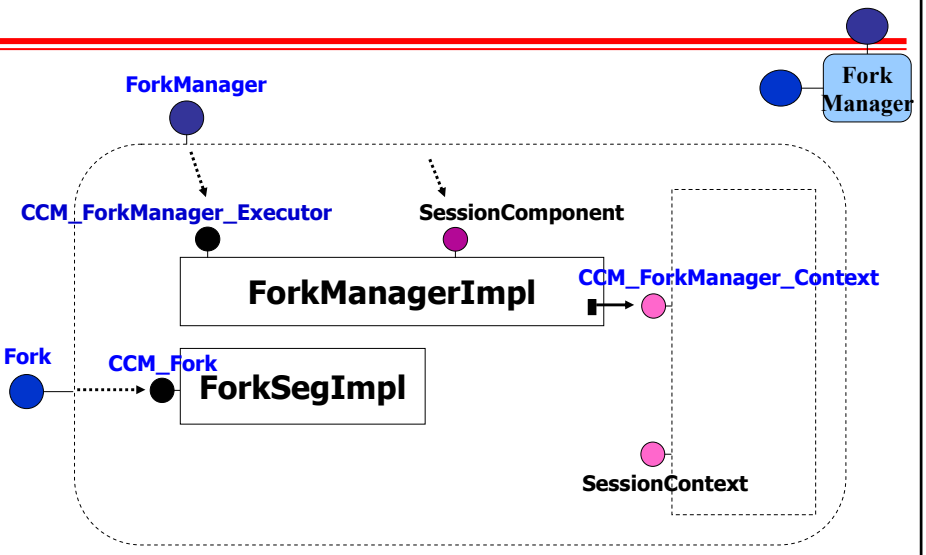
La structure d'implantation du composant Philosopher



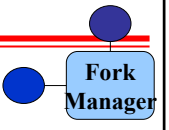
L'implantation Java de la maison ForkHome

```
public class ForkHomeImpl
     extends ForkComposition.ForkManagerHomeBase
{ // La fabrique d'implantation des segments.
  public org.omg.Components.ExecutorSegmentBase
  create_executor_segment(int segid)
  {
    switch (segid) {
      case 0:
        return new ForkManagerImpl();
      case ForkSegImpl._segment_id_value:
        return new ForkSegImpl();
  } }

  // Le point d'entrée pour le déploiement.
  public static org.omg.Components.HomeExecutorBase
  create_home()
  { return new ForkHomeImpl(); }
}
```

Le composant ForkManager

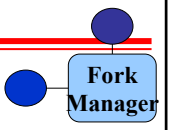


L'implantation Java du composant ForkManager (1/2)




```
public class ForkManagerImpl
     extends ForkComposition.ForkManagerBase
{
  // Le constructeur.
  public ForkManagerImpl() {}

  // Surcharge des opérations SessionComponent.
  // e.g. ccm_activate, ccm_passivate, ccm_remove.
}
```

L'implantation Java du composant ForkManager (2/2)




```
public class ForkSegImpl
     extends ForkComposition.ForkSeg
{
  private boolean disponible_ = true;

  // Prendre la fourchette.
  public void get() throws InUse
  {
    if(!disponible_) throw new InUse();
    disponible_ = false;
  }

  // Rendre la fourchette.
  public void release()
  { disponible_ = true; }
}
```


L'implantation Java de l'événement StatusInfo (1/2)

```
public class StatusInfoImpl
     extends StatusInfo
{
  public StatusInfoImpl(PhilosopherState s,
                        String n, int ticks,
                        boolean left_fork,
                        boolean right_fork)
  {
    state = s; name = n;
    ticks_since_last_meal = ticks;
    has_left_fork = left_fork;
    has_right_fork = right_fork;
  }
  public StatusInfoImpl()
  { this(PhilosopherState.DEAD, "", 0, false, false); }
}
```

L'implantation Java de l'événement StatusInfo (2/2)

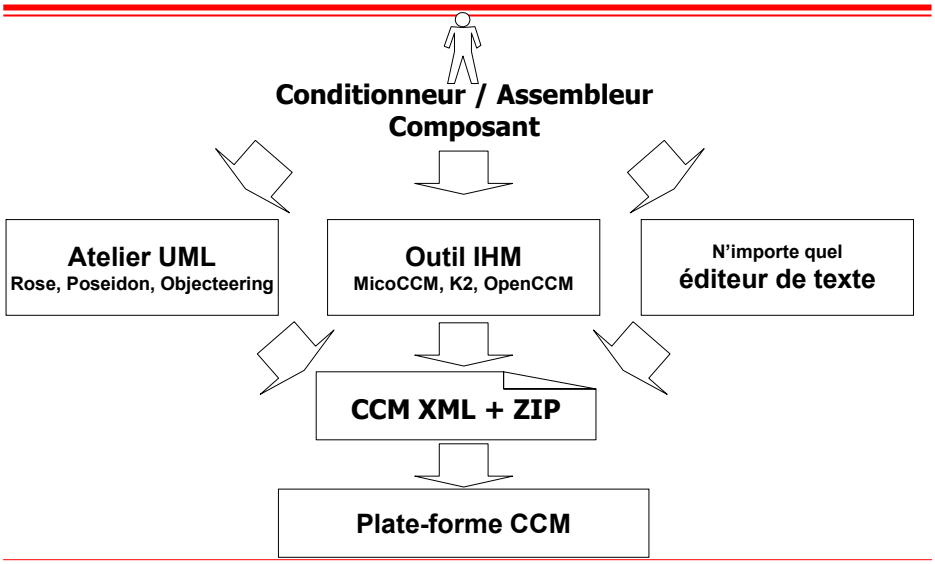
```
public class StatusInfoFactory
  implements org.omg.CORBA.portable.ValueFactory
{
  // Lecture depuis un flux CORBA.
  public java.io.Serializable
  read_value(org.omg.CORBA_2_3.portable.InputStream in)
  {
    java.io.Serializable v = new StatusInfoImpl();
    return in.read_value(v);
  }
}
```


Le conditionnement et l'assemblage des composants CORBA

Le conditionnement et l'assemblage des composants CORBA

- **Construire des archives ZIP**
  - De composants contenant
    - XML Component Software Descriptor (.csd)
    - XML CORBA Component Descriptors (.ccd)
    - XML Component Property Files (.cpf)
    - Fichiers OMG IDL + code binaires
  - D'assemblage contenant
    - XML Component Assembly Descriptor (.cad)
    - Archives de composants
    - XML Component Property Files (.cpf)
- **Besoin d'outils de conditionnement et d'assemblage pour CCM**
  - Diverses approches et outils existent
  - Combinaison de différents outils possible

Des outils de conditionnement et d'assemblage des composants CORBA



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>
```

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" />
```

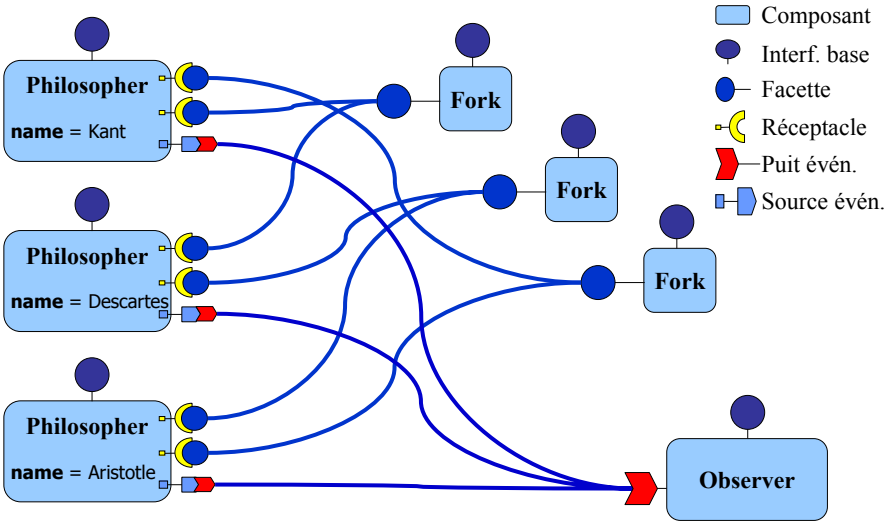

CORBA Component Descriptor pour Composant Philosopher

```
<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"/>
```

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>
```


L'assemblage des composants 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>
```

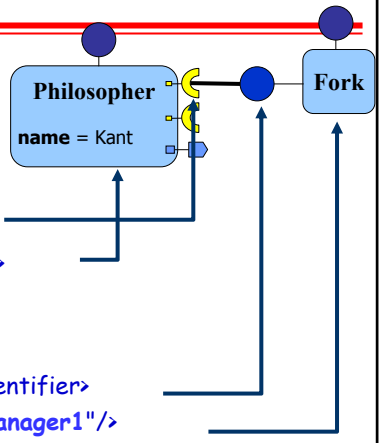

Component Assembly Descriptor pour dîner des philosophes

```
 <partitioning>
  <homeplacement id="ObserverHome">
   <componentfileref idref="ObserverComponent"/>
   <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

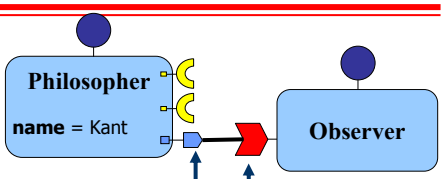
```
<homeplacement id="PhilosopherHome">
 <componentfileref idref="PhilosopherComponent"/>
 <componentinstantiation id="Kant">
   <componentproperties><fileinarchive name="Kant.cpf"/>
 </componentproperties></componentinstantiation>
 <componentinstantiation id="Descartes">
   <componentproperties><fileinarchive name="Descartes.cpf"/>
 </componentproperties></componentinstantiation>
 <componentinstantiation id="Aristotle">
   <componentproperties><fileinarchive name="Aristotle.cpf"/>
 </componentproperties></componentinstantiation>
</homeplacement>
</partitioning>
```


Component Assembly Descriptor pour dîner des philosophes



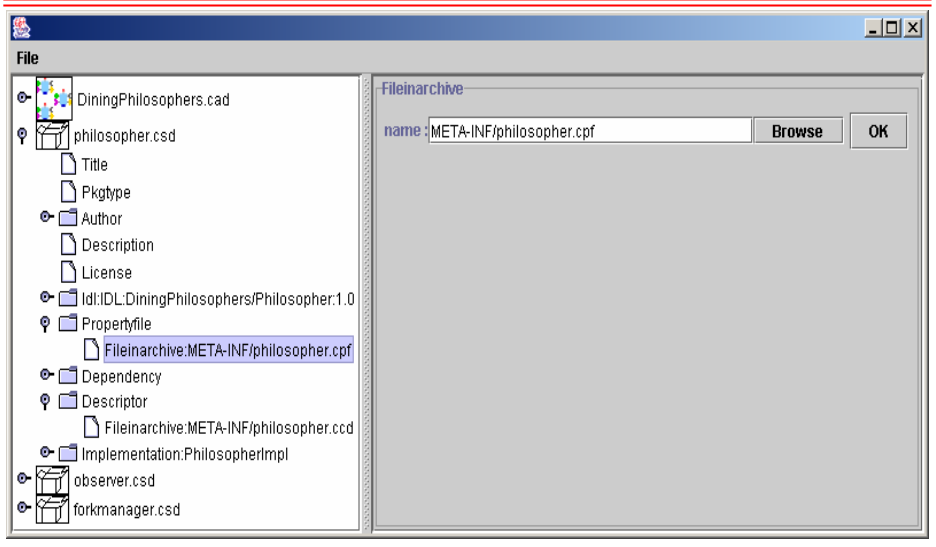

```
<connections>
 <connectinterface>
  <usesport>
   <usesidentifier>left</usesidentifier>
   <componentinstantiationref idref="Kant"/>
  </usesport>
  <providesport>
   <providesidentifier>the_fork</providesidentifier>
   <componentinstantiationref idref="ForkManager1"/>
  </providesport>
 </connectinterface>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

Component Assembly Descriptor pour dîner des philosophes

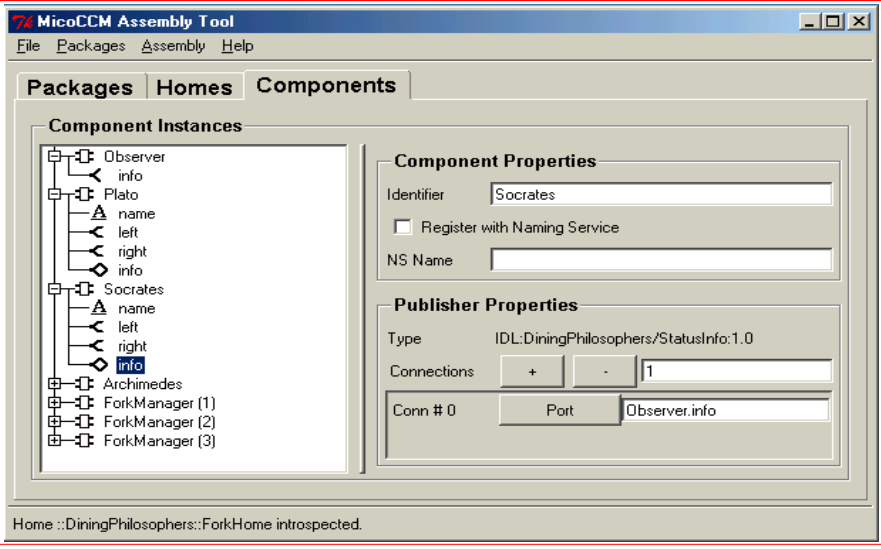



```
<connectevent>
  <publishesport>
    <publishesidentifier>info</publishesidentifier>
    <componentinstantiationref idref="Kant"/>
  </publishesport>
  <consumesport>
    <consumesidentifier>info</consumesidentifier>
    <componentinstantiationref idref="Freud"/>
  </consumesport>
</connectevent>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

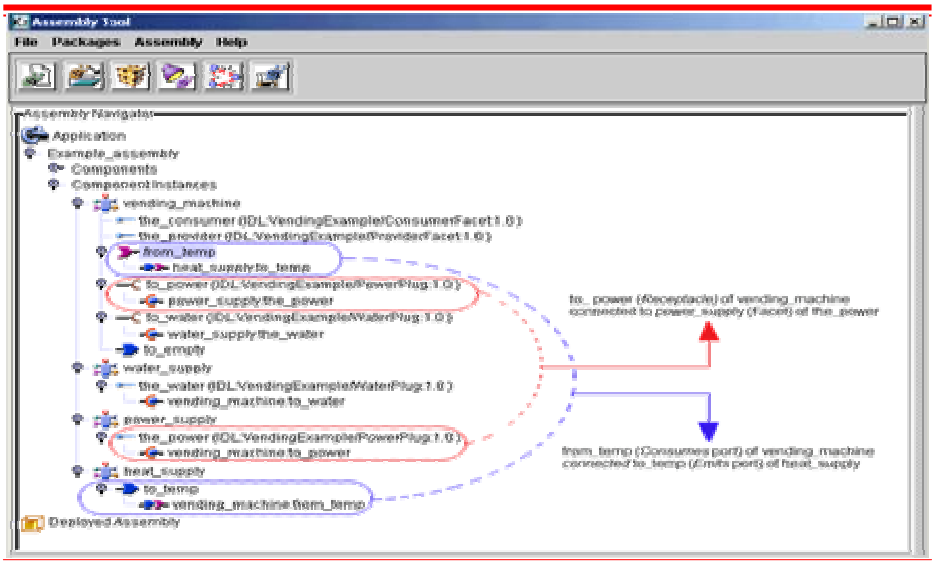

L'outil de conditionnement et d'assemblage d'OpenCCM



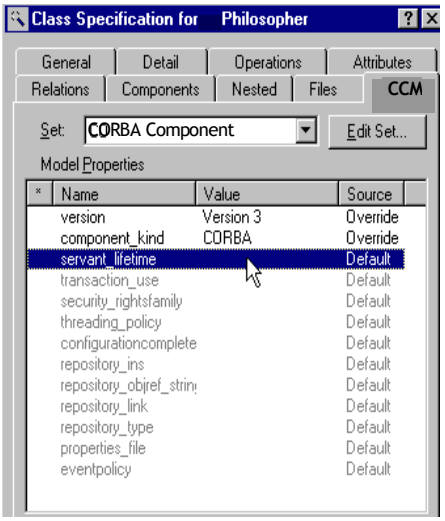
L'outil d'assemblage de MicoCCM



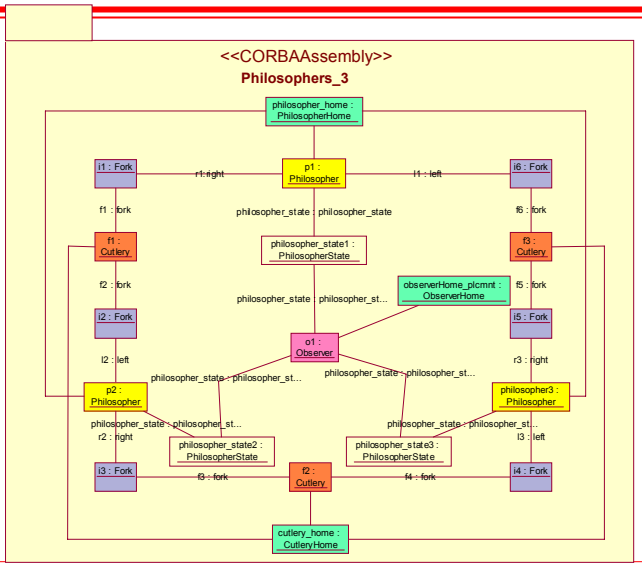
L'outil d'assemblage de K2



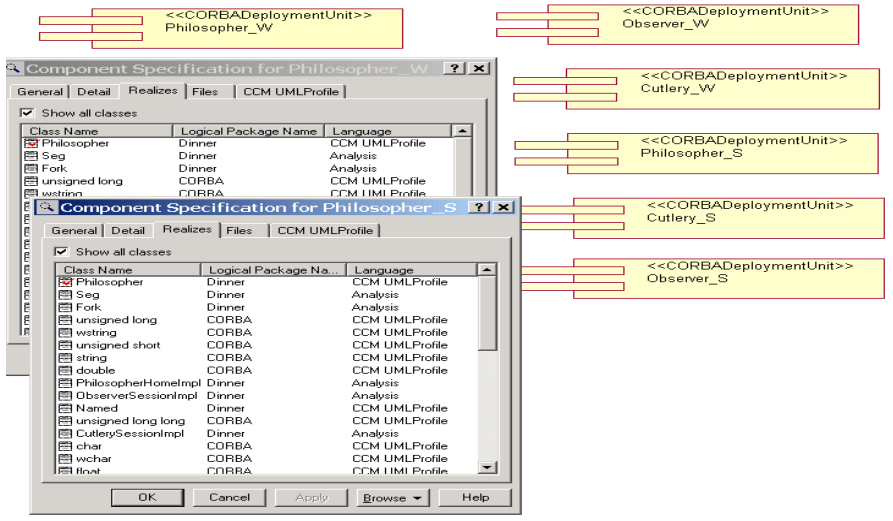
Le plug-in Rational Rose du projet IST COACH



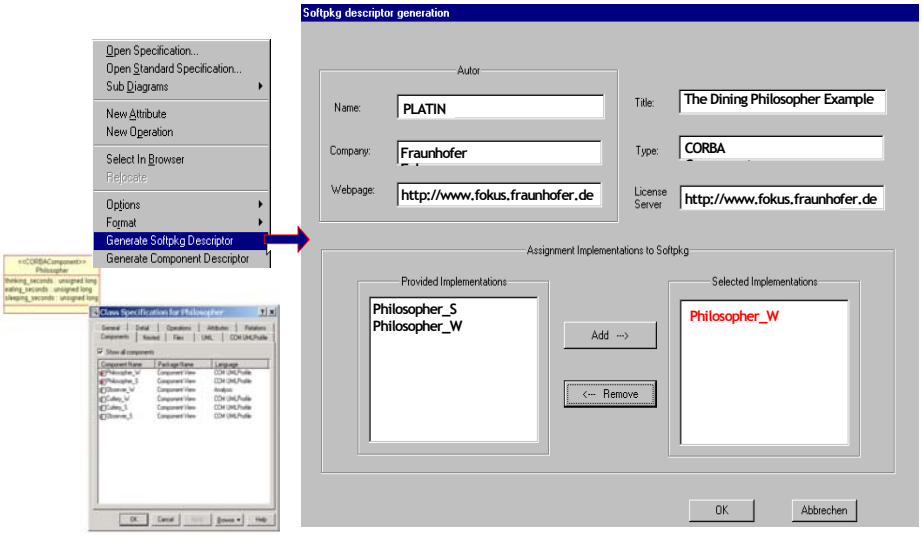
Le plug-in Rational Rose du projet IST COACH



Le plug-in Rational Rose du projet IST COACH



Le plug-in Rational Rose du projet IST COACH



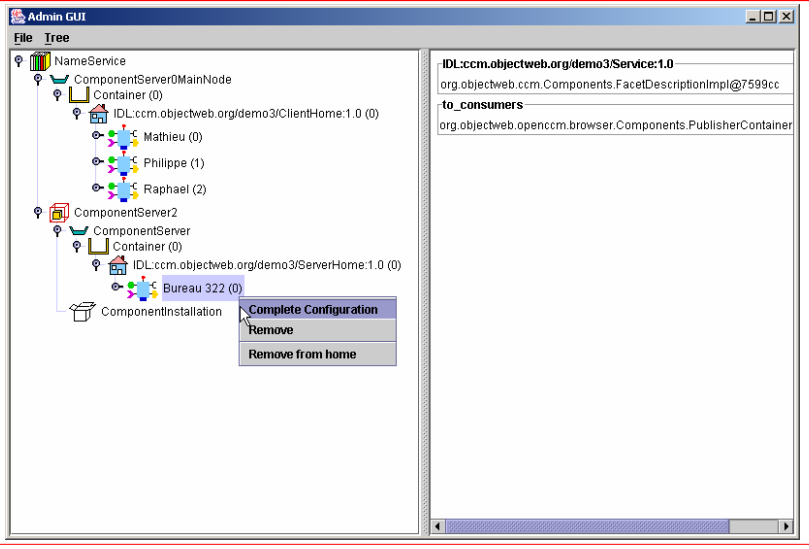
Le déploiement, l'exécution et l'administration de l'application

Le déploiement, l'exécution et l'administration de l'application

- **Déploiement automatisé**
  - Lancement via un outil de déploiement
- **Exécution automatisée**
  - Lancement de démons à faire manuellement
    - 1 domaine
      - 1 CORBA Naming Service
      - 1 CCM Assembly Factory
    - 1 machine
      - 1 CCM Server Activator
      - 1 CCM Component Installation
- **Administration de l'application**
  - Des programmes clients ad hoc à l'application
  - Utilisation d'une console générique

Déployer le dîner des philosophes avec OpenCCM

La console d'administration OpenCCM



Conclusion

Conclusion

- Pas si compliqué si correctement expliqué (?)
- Pas parfait mais puissant
- Bonne source d'inspiration pour faire mieux ;-)

Merci