Composants Logiciels

Christian Pérez

Le modèle de composant de CORBA

Année 2010-11

Plan

- Un rapide tour d'horizon de CORBA 2
- Introduction au modèle de composant de CORBA
- Définition de composants CORBA
  - Exemple du dîner des philosophes
- Introduction à CIDL

Un rapide tour d'horizon de CORBA2

Qu'est ce que CORBA

- CORBA = Common Object Request Broker Architecture
  - Défini par l'Object Management Group (OMG)
- CORBA 2: Standard pour la programmation d'objets distribués
  - Bus logiciel
  - Orienté objet
  - Mécanisme d'invocation de méthode à distance
  - Indépendant des machines, des systèmes d'exploitation et des langages de programmation
  - Indépendant des vendeurs (interopérabilité)

4

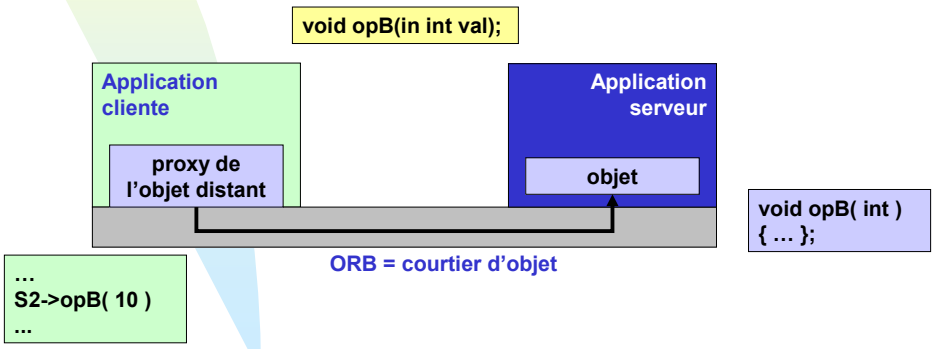
Qu'est ce que CORBA 2 (suite)

- CORBA définit
  - Un langage de spécification (IDL)
    - ~ interfaces en Java + sens des données (IN, OUT, INOUT)
  - Des interfaces pour invoquer des méthodes à distance
  - Un ensemble de services pour manipuler aisément les objets
- CORBA simplifie
  - La localisation des objets
  - L'invocation des méthodes distantes

5

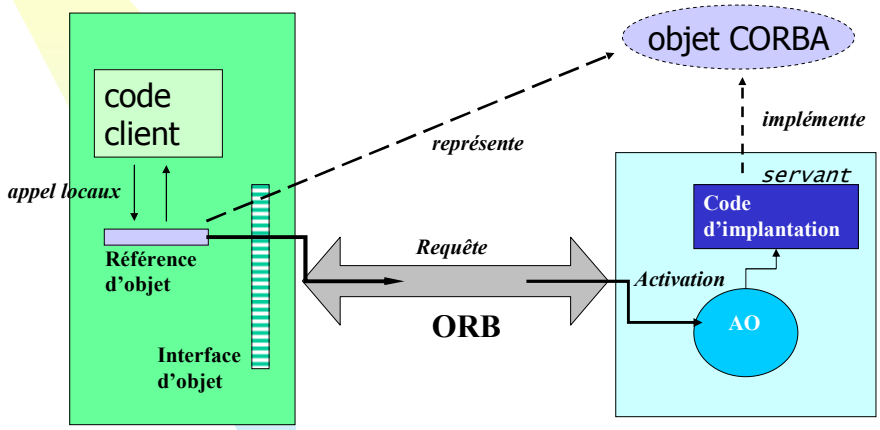
Application CORBA 2

- Ensemble d'objets et de "clients"
- Chaque objet possède une description publique
  - Opérations et attributs pouvant être accédés à distance



6

CORBA
un modèle client/serveur objet

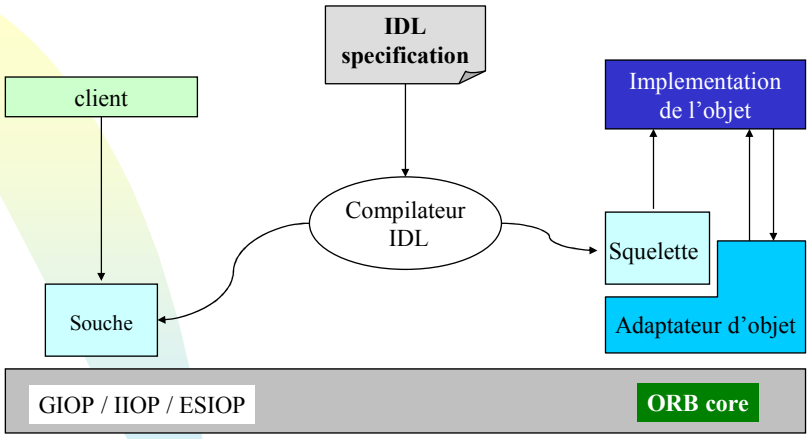


Coté client

Coté service

7

Rôle du langage IDL



- Description des interfaces des objets
- Langage pivot entre applications
- Génération des souches & squelettes

8

Exemple de description d'interface en IDL

```
module ServiceDate {

  typedef unsigned short Jour;
  enum Mois {
    Janvier, Février, Mars, Avril, Mai, Juin, Juillet,
    Août, Septembre, Octobre, Novembre, Décembre
  };
  typedef unsigned short Annee;
  struct Date {
    Jour le_jour;
    Mois le_mois;
    Annee l_annee;
  };
  typedef sequence<Date> desDates;

  interface Calendrier {
    attribute Annee annee_courante;
    boolean vérifier_une_date(in Date d);
    void le_jour_suivant(inout Date d);
  }
};
```

9

Introduction au modèle de composant de CORBA

De CORBA 2 . . .

- Un modèle orienté objet distribué
  - Hétérogénéité: OMG Interface Definition Language
  - Portabilité: des projections standardisées
  - Interopérabilité: GIOP / IIOP
  - Plusieurs modèles d'invocation: SII, DII et AMI
  - Middleware: ORB, POA, etc.
- Pas de support standard pour l'empaquetage et le déploiement !!!
- Programmation explicite des propriétés non fonctionnelles!
  - cycle de vie, (de)activation, service de nommage, de courtier, de notification, de persistance, de transactions, ...
- Pas de notion d'architecture logicielle

11

...au modèle de composant de CORBA (CCM)

- Un modèle orienté composant distribué
  - Une architecture pour définir des composants et leurs interactions
  - Une technologie d'empaquetage pour déployer des binaires exécutable multi-langages
  - Un framework à conteneur pour gérer le cycle de vie, (de)activation, sécurité, transactions, persistance et les évènements
  - Interopérabilité avec Enterprise Java Beans (EJB)
- Le premier modèle de composant industriel multi-langages
  - Multi-langages, multi-OSs, multi-ORBs, multi-vendeurs, etc.
  - CORBA 3.0 – Juillet 2002

12

Contenu des spécifications CCM

- Un modèle abstrait de composants
  - Étend l'IDL et le modèle objet
- Framework d'implémentation des composants (CIF)
  - Composant Implementation Definition Language (CIDL)
- Un modèle de programmations des conteneurs de composants
  - Vues de l'implémenteur et du client
  - Intégration avec les services de sécurité, de persistance, de transactions et d'évènements

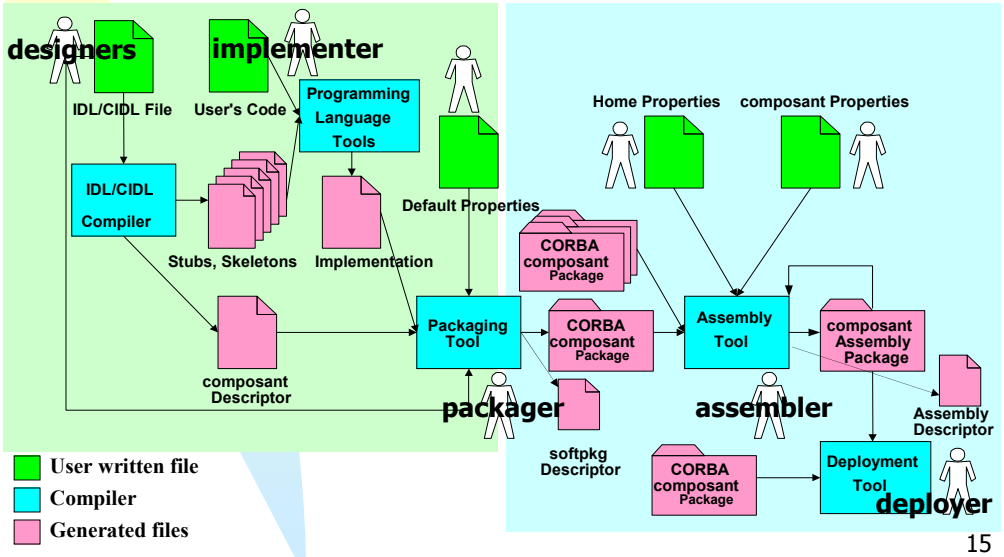
13

Contenu des spécifications (2)

- Des facilités d'empaquetage et de déploiement
  - Fichiers de descriptions en XML
- L'interopérabilité avec EJB via des passerelles
- Des méta-modèles
  - De composants
  - Une interface « Repository » et des extensions au MOF

14

Le grand schéma de CCM



15

Définition de composants CORBA

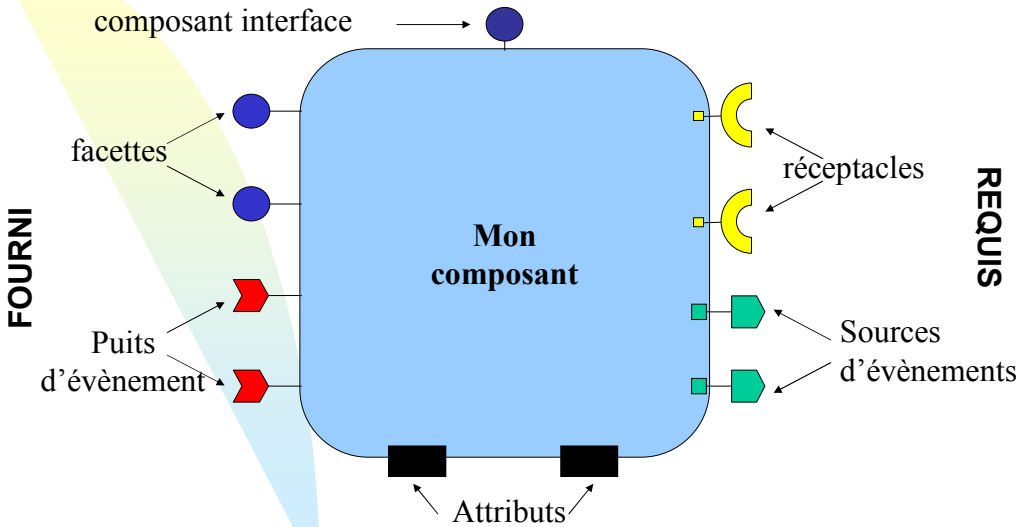
- Le modèle abstrait des composants

Le modèle abstrait des composants

- Il décrit comment un composant CORBA est vu par les autres composants et par les clients
  - Ce qu'offre un composant aux autres composants (*provide*)
  - Ce qu'il demande des autres composants (*use*)
  - Le modèle de collaboration utilisé entre composant
    - Synchrone via les invocations d'opération
    - Asynchrone via la notification d'évènement
  - Quelles propriétés du composant sont configurables
  - Quelles sont les opération du cycle de vie (*home*)

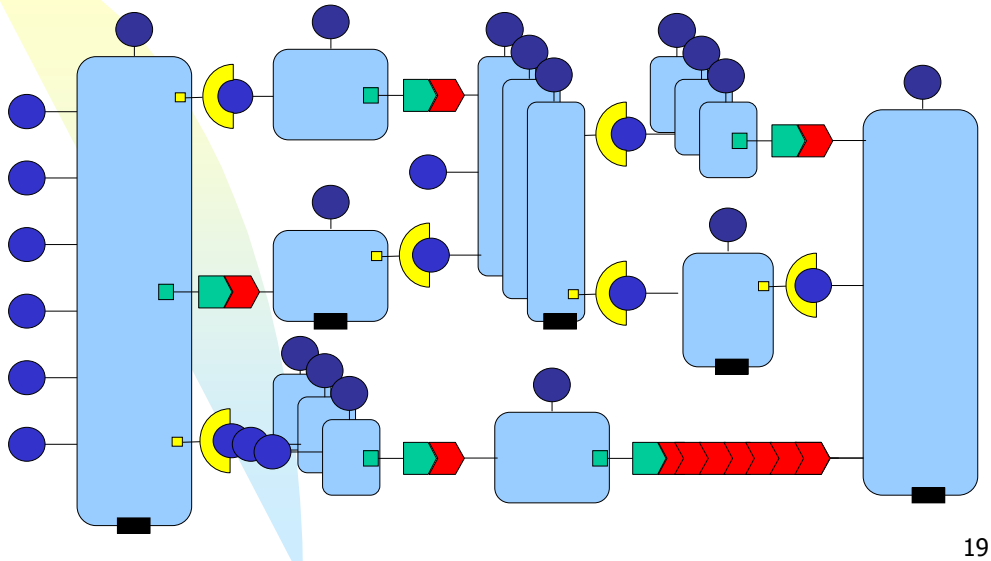
17

Un composant CORBA



18

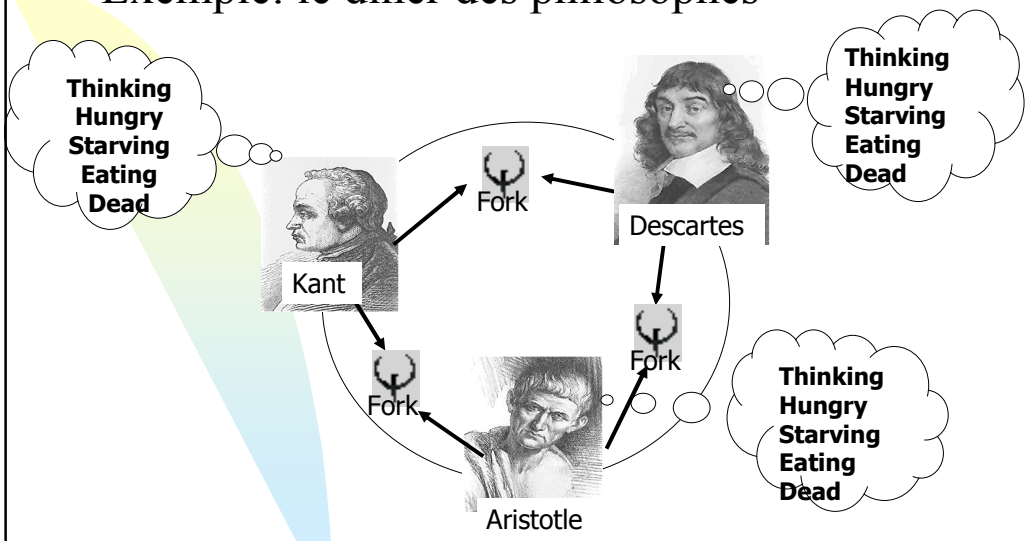
Construction d'une application CCM =
Assemblage d'instances de composants



19

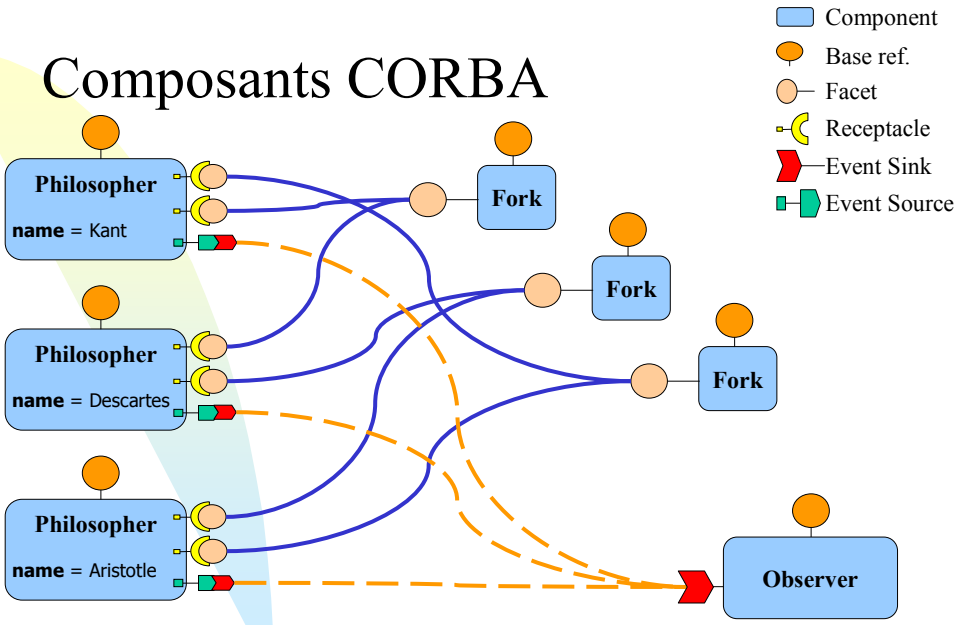
Exemple du dîner des philosophes

Exemple: le dîner des philosophes



21

Composants CORBA



22

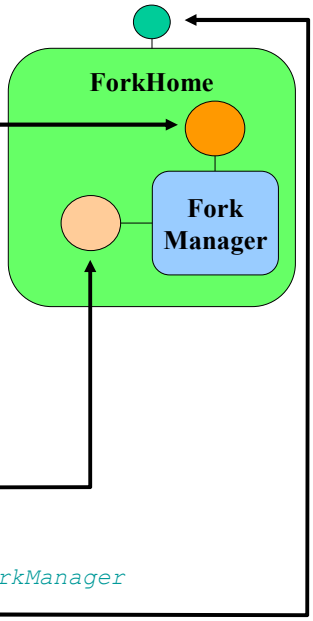
Le composant Fork

```
exception InUse {};

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

// Le composant
component ForkManager {
  // Facette utilisé par les phisolphes.
  provides Fork the_fork;
};

// Maison pour instancier les composants ForkManager
home ForkHome manages ForkManager {};
```



23

Les types d'état d'un philosophe

```
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;
};
```

24

Le composant philosophe

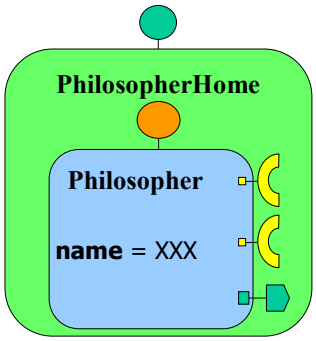
```
component Philosopher {
  attribute string name;

  // 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);
};
```

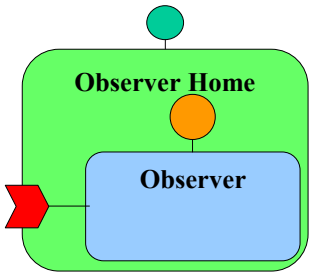


25

Le composant observateur

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

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



26

Introduction à CIDL

- Description du problème
- CIDL
- Exemple

Quel est le problème ?

- L'interface abstraite du composant définie en IDL3
- Question: comment on l'implémente ?

28

Implémentation de composants v1

- Première solution
  - Implémentation dépendant
- Exemple
  - les webservices
- Avantage
  - Très grande liberté d'implémentation
  - Contrôle des dépendances (bibliothèques)
- Inconvénient
  - Pas portabilité du code écrit
    - Quid de la génération de code ?
  - Pas de gestion des services systèmes

29

Implémentation de composants v2

- Deuxième solution
  - Définir un environnement d'implémentation
- Exemple
  - CCM
- Avantage
  - Portabilité du code écrit
  - Gestion automatique de service système
  - Gain de temps
- Inconvénient
  - Il faut maîtriser le modèle
  - Liberté d'implémentation *a priori* restreinte
    - Non restreinte car CIDL non obligatoire

30

Implémentation de composants CCM

- Deux modèles de CCM sont principalement utilisés
- Framework d'implémentation des composants (CIF)
  - Composant Implementation Definition Language (CIDL)
- Un modèle de programmations des conteneurs de composants
  - Vues de l'implémenteur et du client
  - Intégration avec les services de sécurité, de persistance, de transactions et d'évènements

31

Component Implementation Definition Language (CIDL)

- Décrit une composition de composant
  - Entité agrégé qui décrit tous les artéfacts requis pour implémenté un composant et sa maison
- Gère la persistance d'un composant
  - Basé sur le OMG Persistent State Definition Language (PSDL)
  - Liens entre des types de stockage et les exécuteurs segmentés
- Génère des squelettes d'exécuteur fournissant
  - La segmentation des exécuteurs des composants
  - Une implémentation par défaut des opérations de callback
  - La persistance de l'état d'un composant

32

Notion de Composition CCM

- Entité agrégée décrivant les artéfacts requis pour implémenter un composant et sa maison
  - Nom de la composition
  - Catégories de cycle de vie des composants
    - Service, session, process, entity
  - Un type de maison de composant (1)
    - Le type du composant à implémenter est défini implicitement
  - Une définition d'exécuteur de la maison (2)
  - Une définition d'exécuteur du composant (3)
  - *Une définition de délégation*
  - *Une définition pour un proxy de la maison*
  - *Une association à un espace de stockage abstrait*

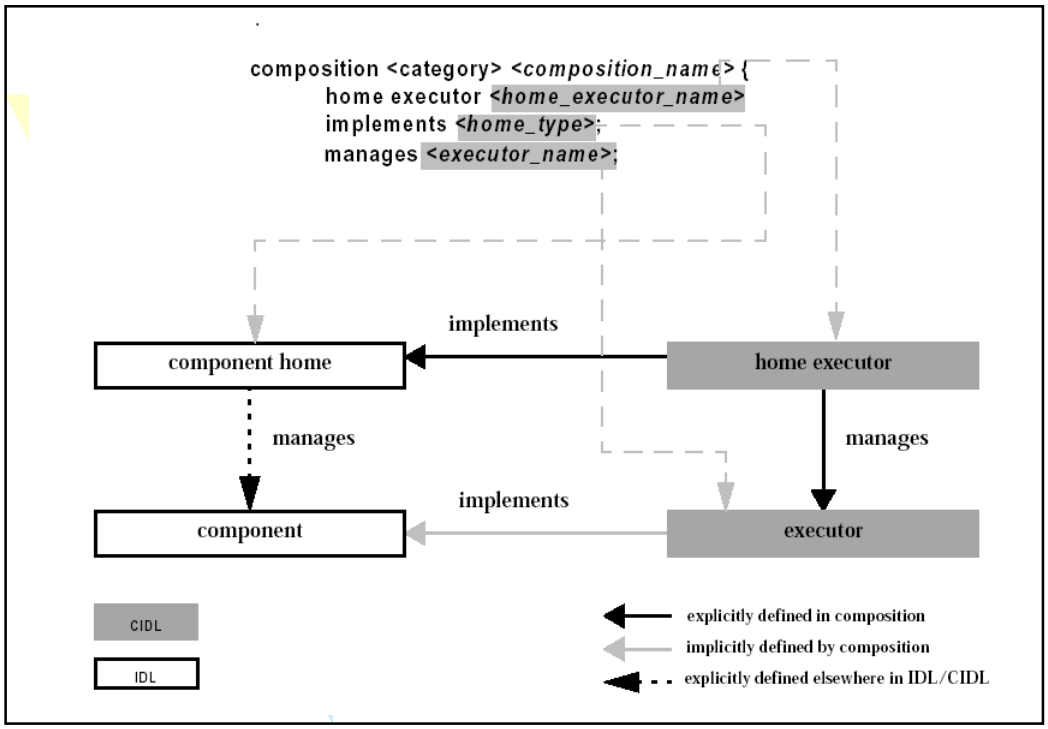
```
composition <categorie> nom_composition {
   home executor nom_executeur_maison {          // (2)
      implements nom_type_maison;                // (1)
      manages nom_executeur;                     // (3)
   }
};
```

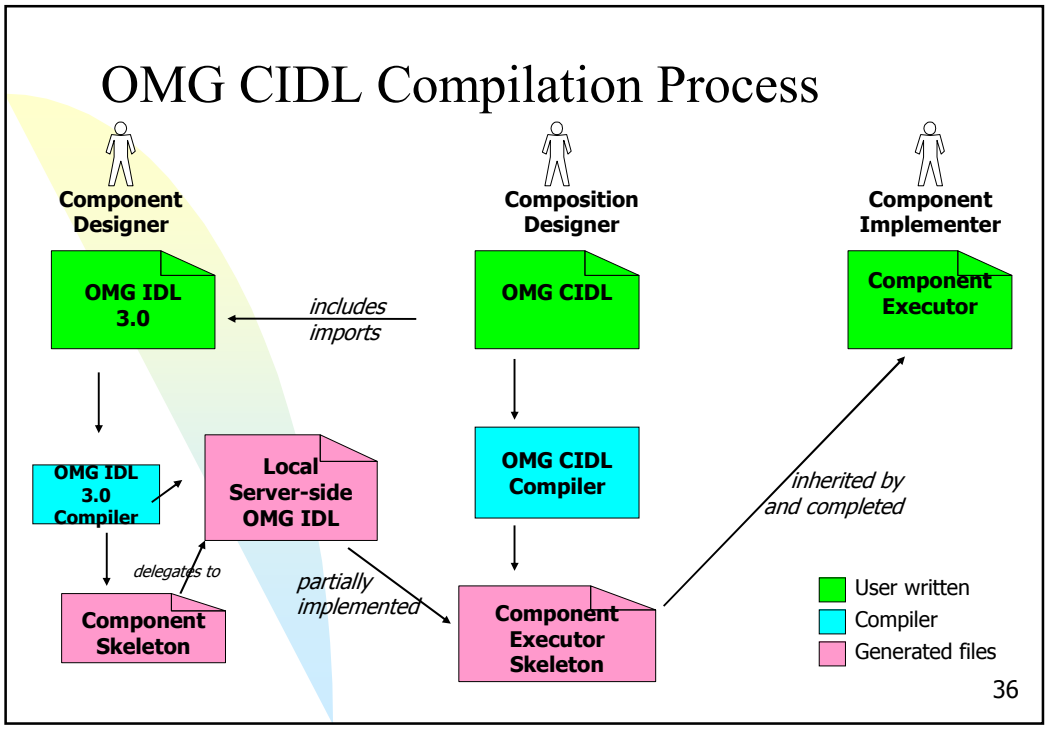
33

Notion d'exécuteur

- Artéfact de programmation implémentant une abstraction.
- Dans les langages objets, un exécuteur = un objet

34


composition <category> <composition_name> {
home executor <home_executor_name>
implements <home_type>;
manages <executor_name>;
implements
component home
home executor
manages
manages
implements
component
executor
CIDL
IDL
explicitly defined in composition
implicitly defined by composition
explicitly defined elsewhere in IDL/CIDL


OMG CIDL Compilation Process
Component Designer
Composition Designer
Component Implementer
OMG IDL 3.0
includes imports
OMG CIDL
Component Executor
OMG IDL 3.0 Compiler
Local Server-side OMG IDL
OMG CIDL Compiler
inherited by and completed
delegates to
partially implemented
Component Skeleton
Component Executor Skeleton
User written
Compiler
Generated files
36

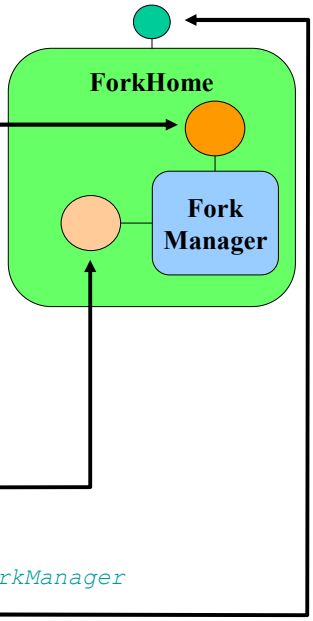
Maison du composant

```
exception InUse {};

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

// Le composant
component ForkManager {
  // Facette utilisé par les phisolphes.
  provides Fork the_fork;
};

// Maison pour instancier les composants ForkManager
home ForkHome manages ForkManager {};
```



37

CIDL Composition for Observer Component

```
#include <philo.idl>
// or import DiningPhilosophers;

composition session ForkManagerSessionComposition
{
  home executor ForHomeImpl
  {
    implements DiningPhilosophers::ForkHome;
    manages ForkManagerImpl;
  };
};
```

38

Implémentation de la maison

```
package DiningPhilosophers.monolithic;
import DiningPhilosophers.*;
public class ForkHomeImpl extends org.omg.CORBA.LocalObject
                          implements CCM_ForkHome {
  public  ForkHomeImpl(){}
  public org.omg.Components.EnterpriseComponent create()
  {
      return new ForkManagerImpl();
  }
  public static org.omg.Components.HomeExecutorBase create_home()
  {
      return new ForkHomeImpl();
  }
}
```

39

Implémentation du composant

```
package DiningPhilosophers.monolithic;
import DiningPhilosophers.*;
public class ForkManagerImpl extends org.omg.CORBA.LocalObject
                         implements CCM_ForkManager, CCM_Fork,
                                    org.omg.Components.SessionComponent
{
  private boolean available_;
  public CCM_Fork get_the_fork() { // From CCM_ForkManager
      return this; // Returns an implementation of the Fork facet
  }
  public void get() throws InUse { // From CCM_Fork
      if (! available_) throw new InUse();
      available_ = false;
  }
  public void release() {          // From CCM_Fork
      if (available_) return;
      available_ = true;
  }
}
```

40