d) Création d'un fichier ejb-jar

Permet de packager et de distribuer des composants serveur de la même manière que l'on distribue des composants GUI

Les étapes à réaliser sont les suivantes:

1. Compiler les sources .java
2. Créer un Descripteur de Déploiement

Déploiement = installer un composant EJB dans un conteneur

Permet de définir, et de personnaliser un certain nombre de propriétés avant le déploiement

- Note: This example is for the BEA Weblogic server which uses the semi-colon (;) to comment out lines.

```
(SessionDescriptor
; This file must start with SessionDescriptor or
; EntityDescriptor

; Indicate the name which the bean will be bound
; into the JNDI name as
beanHomeName              demo.DemoHome                 ; (* 1 *)

; The enterprise Java Bean class (see step 4)
enterpriseBeanClassName       ejb.demo.DemoBean         ; (* 2 *)

homeInterfaceClassName        ejb.demo.DemoHome         ; (* 3 *)
; The home interface implemented by a class
; generated by the container provided tools
; see step 3

remoteInterfaceClassName      ejb.demo.Demo             ; (* 4 *)
; See step 2
```

```
isReentrant               false
; Always false for session beans

stateManagementType       STATELESS_SESSION
; Either STATELESS_SESSION or STATEFUL_SESSION.
; DemoBean is a stateless session bean

sessionTimeout            5 ; seconds

(controlDescriptors
; This section decides the run-time properties when
; a method is called. The DEFAULT sub-section applies
; to all methods, but can be overridden on a per-method
; basis, similar to the "accessControlEntries" above.
(DEFAULT
  isolationLevel          TRANSACTION_SERIALIZABLE
  transactionAttribute    TX_REQUIRED
  runAsMode               CLIENT_IDENTITY
)
; end isolationLevel

)
; end controlDescriptors

(environmentProperties
  maxBeansInFreePool          100
; end environmentProperties
)
; end SessionDescriptor
```

On voit bien que le lien entre la classe du bean et:

- le nom pour le lookup (* 1 *),
- l'interface remote implémentée (* 4*),
- l'interface“Home“ (* 3*)

n'est pas codé dans le code mais donné ici dans le descripteur.

3. Création d'un manifeste

il est créer automatiquement par l'outils jar

ejb/demo/manifest.txt

```
Name: ejb/demo/DemoBeanDD.ser
Enterprise-Bean: True
```

Récapitulatif des fichiers nécessaires:

- bean class + autre classes utilisées par le bean
- bean’s remote interface
- bean’s home interface
- Descripteur de Déploiement
- Éventuellement une instance de java.util.Properties si le bean utilise des propriétés Java
- un fichier manifeste

4. Enfin: création du ejb-jar file

```
jar cvfm Demo.jar ejb/demo/manifest.txt ejb/demo/*.class ejb/demo/*.ser
// DemoBeanDD.ser: fichier de déploiement généré par weblogic (depuis .tx)
```

Des outils existent pour faciliter le packaging et la génération du ficier ejb-jar ...
Standard XML deploiement descriptor in EJB 2.0

```
jar tf Demo.jar
META-INF/MANIFEST.MF
ejb/demo/Demo.class
ejb/demo/DemoBean.class
ejb/demo/DemoHome.class
ejb/demo/DemoBeanDD.ser
```

e) Déploiement du bean

Spécifique à l'implémentation du serveur d'EJB

Exemple de Weblogic server

1. Générer les classes d'implementation

```
java weblogic.ejbc -d /export/weblogic/classes ejb/demo/DemoBeanDD.ser
```

Va générer les fichiers:

```
ejb/demo/DemoBeanEOImpl.class
ejb/demo/DemoBeanHomeImpl.class
ejb/demo/Skel5k5x705r2x671nd1i1vy2v524ua5y.class
ejb/demo/Skel5q585f5sfzo601q4e725b233m5140.class
ejb/demo/Stub5k5x705r2x671nd1i1vy2v524ua5y.class
ejb/demo/Stub5q585f5sfzo601q4e725b233m5140.class
```

2. Rendre le Demo.jar accessible au serveur d'EJB
3. Rendre le descripteur accessible au serveur d'EJB
4. Démarrer ou arrêter l'EJB serveur

f) Le code du client

Le client d'un EJB peut être:

- un client Java
- une servlet, une applet,
- un programme C/C++ (par CORBA, IIOP)

--> Dessin tableau (?)

Les étapes à réaliser pour le client sont:

- Établir un contexte initial (EJB, + JNDI)
- Trouver la Home Interface de l'EJB avec JNDI
- Utiliser la Home interface pour demander au conteneur de créer une instance de l'EJB, et récupérer une référence distante vers l'EJB
- Utiliser cette référence pour appeler des méthodes sur l'EJB (donc par RMI)

```
/**
* DemoClient -- demonstrates using a minimal
* Java application to talk to the DemoBean
* stateless session bean
*/
package ejb.demo;
import javax.ejb.*;
import javax.naming.*;
import java.rmi.*;
import java.util.Properties;
/**
* DemoClient demonstrates using a minimal stateless
* session bean.
* Remember view session beans as an extension of your
* client running in the server.
*/

public class DemoClient {
  public static void main(String[] args) {
        System.out.println("\nBegin DemoClient...\n");

        parseArgs(args);
        try {
        // Create A DemoBean object, in the server
        // Note: the name of the class corresponds to the
        // JNDI property declared in the
        // DeploymentDescriptor
        // From DeploymentDescriptor ...
```

```
        // beanHomeName demo.DemoHome
        Context ctx = getInitialContext();
        DemoHome dhome = (DemoHome)                          // (* 3 *)
              ctx.lookup("demo.DemoHome"); // (* 1 *)
        // Now you have a reference to the DemoHome object
        // factory use it to ask the container to creat an
        // instance of the Demo bean
        System.out.println("Creating Demo\n");
        Demo demo = dhome.create();                          // (* 4 *)
                    // In fact : from POOL

        // Here is the call that executes the method on the
        // server side object
    System.out.println("The result is "+ demo.demoSelect()); (* 4 *)
                                   // CALL: Remote method invocation
        }
            catch (Exception e) {
              System.out.println(" => Error <=");
              e.printStackTrace();
            }
            System.out.println("\nEnd DemoClient...\n");
           }

static void parseArgs(String args[]) {
            if ((args == null) || (args.length == 0))
             return;
            for (int i = 0; i < args.length; i++) {
             if (args[i].equals("-url"))
              url = args[++i];
             else if (args[i].equals("-user"))
              user = args[++i];
             else if (args[i].equals("-password"))
              password = args[++i];
            }
}

static String user = null;
static String password = null;
static String url = "t3://localhost:7001";

/**
* Gets an initial context.
* @return Context
```

```
* @exception java.lang.Exception if there is
* an error in getting a Context
*/
static public Context getInitialContext()
                throws Exception {
  Properties p = new Properties();
  p.put(Context.INITIAL_CONTEXT_FACTORY,
                "weblogic.jndi.T3InitialContextFactory");
  p.put(Context.PROVIDER_URL, url);
  if (user != null) {
          System.out.println ("user: " + user);
                p.put(Context.SECURITY_PRINCIPAL, user);
                if (password == null)
                  password = "";
                p.put(Context.SECURITY_CREDENTIALS, password);
  }
  return new InitialContext(p);
}
}
```

g) Compilation et éxécution du client

```
javac ejb/demo/DemoClient.java

java ejb.demo.DemoClient

            Begin DemoClient...

            Creating Demo

            The result is hello world

            End DemoClient...
```

6.5 Récapitulatif EJB

6.5.1 Classes et interfaces EJB

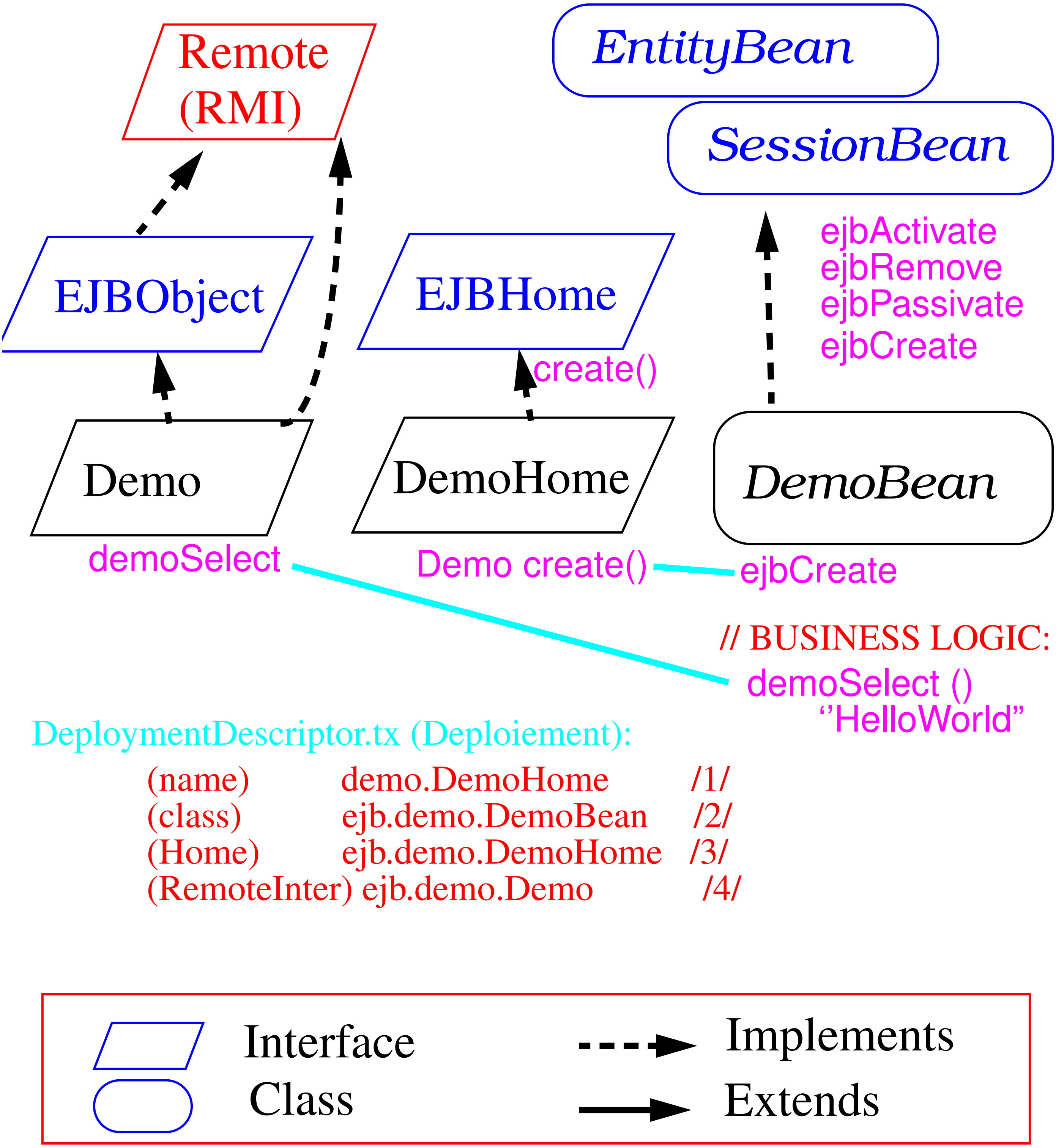


FIGURE 36 Classes et interfaces EJB

6.5.2 Conclusion on EJB

So for EJB components:

Software module = Java Class and Interface
(Home, Remote, Beans, ...)

Standardized description= a file with
a standard format (txt, XML)
with Association in descriptor of :
Lookup Name,
Factory (Home),
Remote (functional) Interface

Tools:
Composition = ? EJBrew ?
Deployment = JVM+
RMI, JTS, +
Generators +
EJB Servers

Examples of Development / Deployment tools:

JBuilder+WebSphere, VisualAge+WebLogic,
Cafe+PowerTier, NetBeans+NetDynamics

some interactives.

Un serveur EJB Open source et Français :

Project ObjectWeb

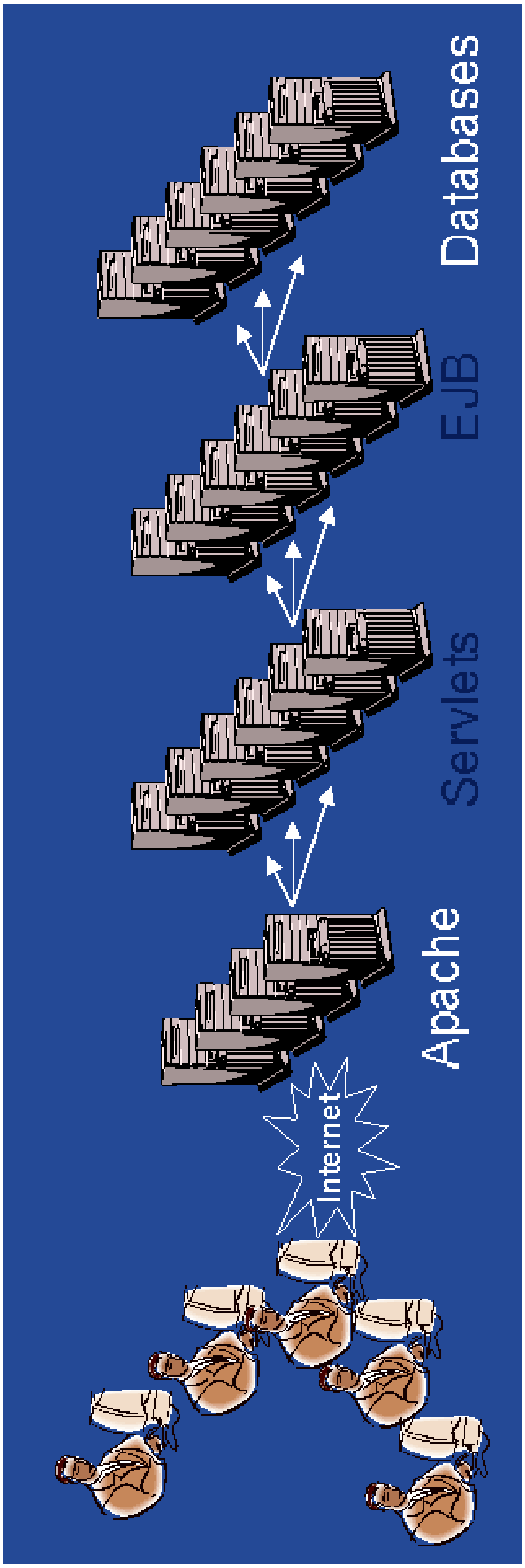
http://www.objectweb.org/

JOnAS (Java TM Open Application Server)

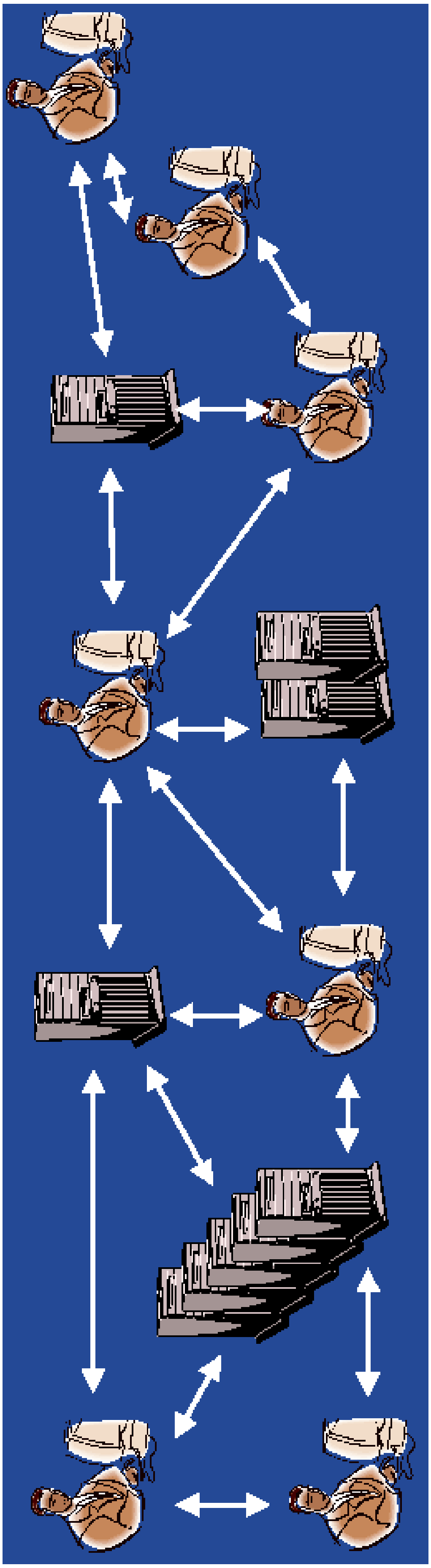
6.6 Web vs P2P

Initial image from Emmanuel Cecchet (ObjectWeb)

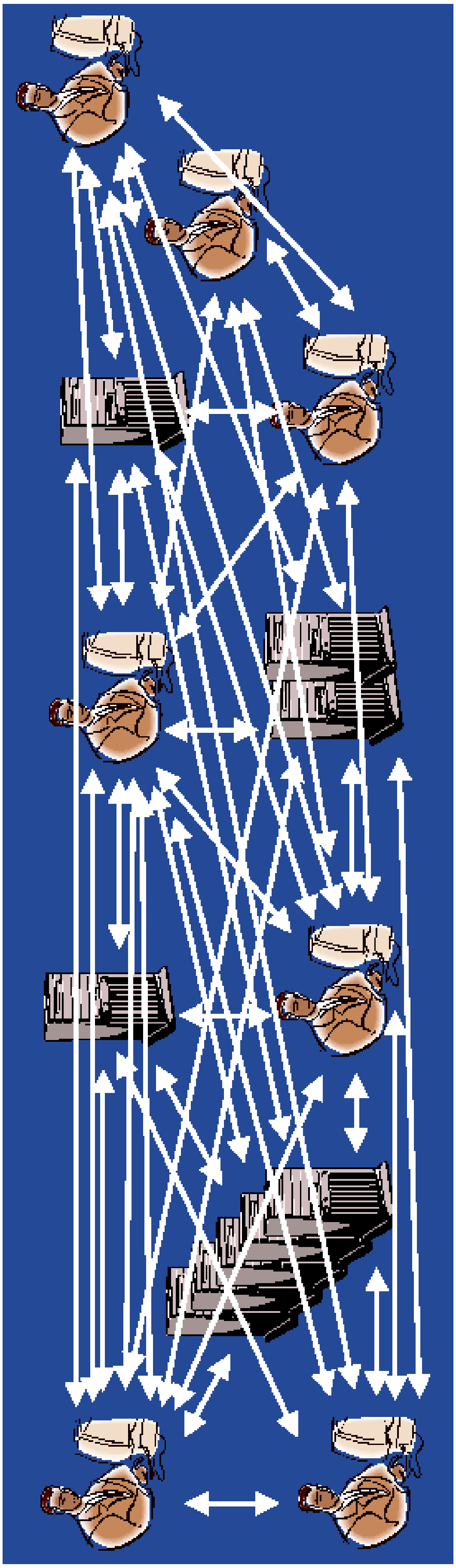
Architecture multi-tiers typique pour le Web :



Architecture P2P typique :



Architecture P2P typique (2) :



CHAPITRE 7 Corba

7.1 Principes, objectifs, IDL, Mapping C++, etc.

Voir transparents annexes.

En particulier, sur le premier support CORBA :

pages 14, 21, 23, 30, 31, 32, 33

+ 66 .. 72 (GIOP <==> JRMP)

Cf. ETSI:
European Telecommunications Standards Institute

Sophia Antipolis, Interop, PlugTests

à vous d'étudier le reste des transparents.

7.2 Conclusion sur CORBA

Bus logiciel:

Recherche d'objets distants sur des interfaces, et nom pas uniquement sur des identifiant (chaînes de caractères, cf. rmiRegistry).

RMI + Rmiregistry + Jini = Bus logiciel Java, "global" (standard: tcp/ip, sockets, démons)

JRMP

7.2.1 Caractéristiques générales

Il existes des implémentations commerciales:

- Component Broker (IBM)
- Object Broker (BEA Systems)
- ORBacus (OOC)
- Orbix (IONA)
- Visibroker (Inprise)
- Voyager (Object Space)

ou libres, gratuites de CORBA:

- JacORB (ORB pour Java)
- Java IDL (Sun)
- ORBit (GNU)
- Fnorb (ORB pour Python)

En pratique, aucun produit n'implémente :

- la totalité des spécifications (qui, de toute manière, évoluent en permanence),

- pour la totalité de langages (C++, Java, Cobol, etc.),

- et la totalité des OS.

==> il y a une explosion combinatoire qui le rend impossible.

La solution:

utiliser l'intéropérabilité offerte par CORBA en utilisant plusieurs implémentations

(Bien sur il faut payer plusieurs produits!)

Note:

Avec CORBA, 2.5 systèmes de Type:

--> • IDL

--> • Langage Cible

--> • Mapping standard IDL --> Lg cible

7.2.2 Choisir un mapping (IDL --> ?)

CORBA avec C++

- projection complexe à utiliser
- efficace et plus ancienne utilisation (fiabilisé)

CORBA avec Java:

- projection plus simple
- plus portable
- pas encore toujours efficace, mais en cours

Autre mapping: CORBA avec CorbaScript:

- interactif, interprété et à objets
- projection simple
- pas de souche et squelettes: utilisation de l'invocation dynamique
- Made in France: Lille, LIFL, thèse Philippe Merle
- accepté par l'OMG comme standard !

7.2.3 Versions successives

Normalement, pas de versions explicites, mais une suites continue de services, normes, etc.

Mais en pratique, des caractéristiques nouvelles donnent souvent un nouveau numéro de version.

C'est également une façon de structurer et comprendre CORBA:

a) CORBA 1

- Langage d'interface: IDL
- mapping C++, etc.
- ORB: bus CORBA
  - communications
  - nommages, cycle de vie, événements
  - autres services (transactions, etc.)

b) CORBA 2

- Interopérabilité entre ORB:
  - GIOP, IIOP

c) CORBA 3

- passage par valeur (sérialisation)

- langage de script (CorbaScript)
- minimumCORBA
- autres (realtimeCORBA, CORBA/COM/DCE)
- firewall

Mais surtout:

- Modèle de composant: CCM
  CORBA Component Model

Largement inspiré des EJB:

Service Components (un appel de méthode)

Session Components

- Stateless Session Components
- Stateful Session Components

Process Components

Entity Components

Container-managed et Component-managed persistance

Fichiers de déploiement en XML

etc.

CCM est un sur-ensemble des EJB:

Un exemple de chose en plus:

CCM a un “assembly descriptor” qui contient des informations (metadata) sur la façon d’interagir de deux composants.

Des questions se pose à l’heure actuelle:

- Le modèle est assez complexe
- Il n’est pas clair si:

  une implémentation C++ a du sens

  si cela est même possible, --> oui

  et dans ce cas, pourquoi ne pas juste utiliser les EJB 2.0 ? (si Java uniquement)
- Les composants sont un modèle très “server-centric”, qui n’utilise pas bien la puissance de calcul disponible chez le client
- Pas de modèle de composants “user interface” en Corba (à la JavaBeans),

  ... à venir .... ?

7.3 Détails sur CCM

CORBA Component Model

Largement inspiré des EJB

Services pour les clients:

- Événements
- Concurrence
- Transactions
- Sécurité
- Persistance

A priori, pour l'instant, pas de composants hiérarchiques, composite. Seulement assemblage au déploiement (diff'erent de Fractal, ObjectWeb)

7.3.1 Principes

OMG IDL3: une version étendue de **IDL**
avec en particulier **component xxxx { ...**

Les composants (IDL3) présentent des "Ports"

Ceux-ci permettent de spécifier :

--> • Connexions possibles
--> • Services offerts (classique)
--> • Services Fournis (Required, Nouveau)

Différents types de ports:

--> • Attributes: *configuration*

--> • Facets: *un point de vue sur un composant*

--> • Receptacles: *gestion des connexions (Client Interface)*

--> • Event Sources/Sinks: *notification*

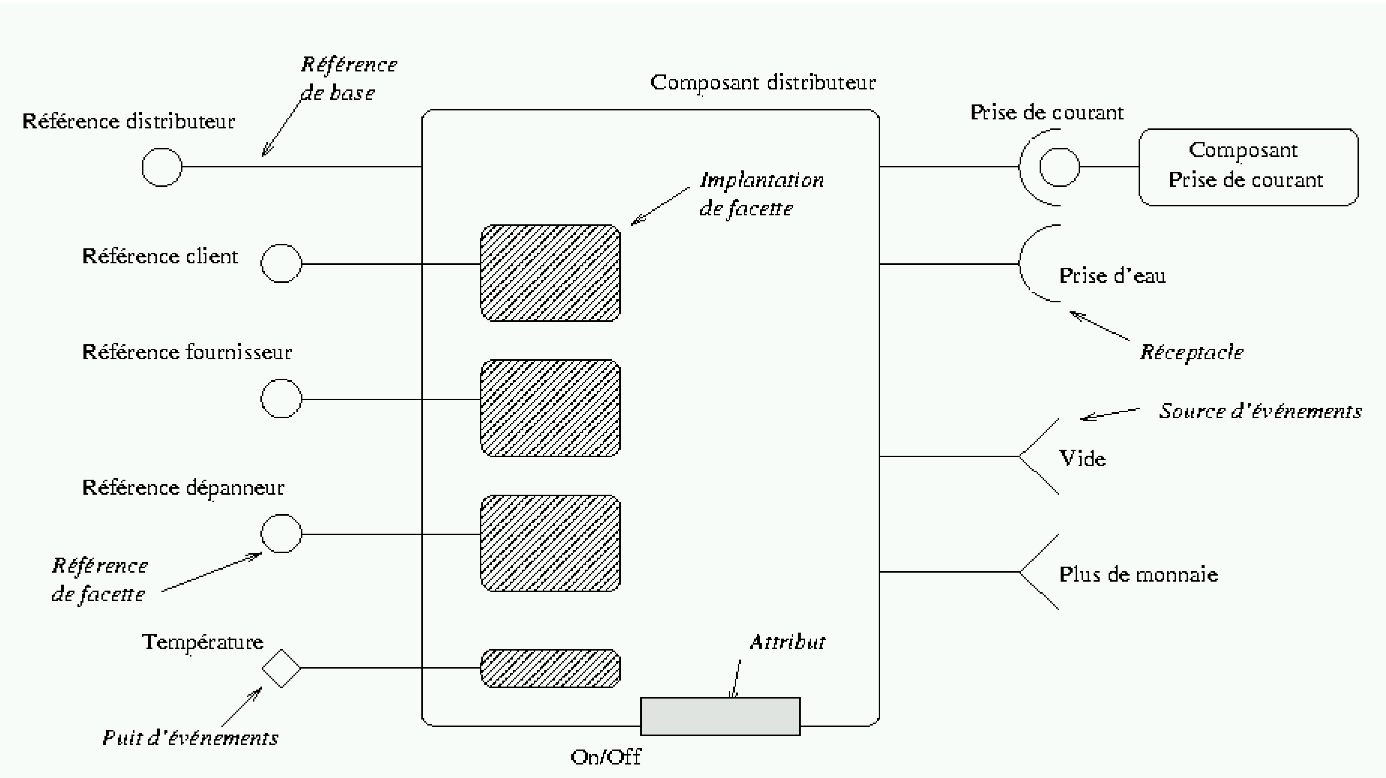


FIGURE 37 Composants Corba CCM,

Un distributeur de boisson CCM
D'après:

Raphaël Marvie et Philippe Merle, Vers un modèle de composants pour CESURE - Le CORBA Component Model, Rapport Technique no 3, projet RNRT'98 CESURE, novembre 2000.

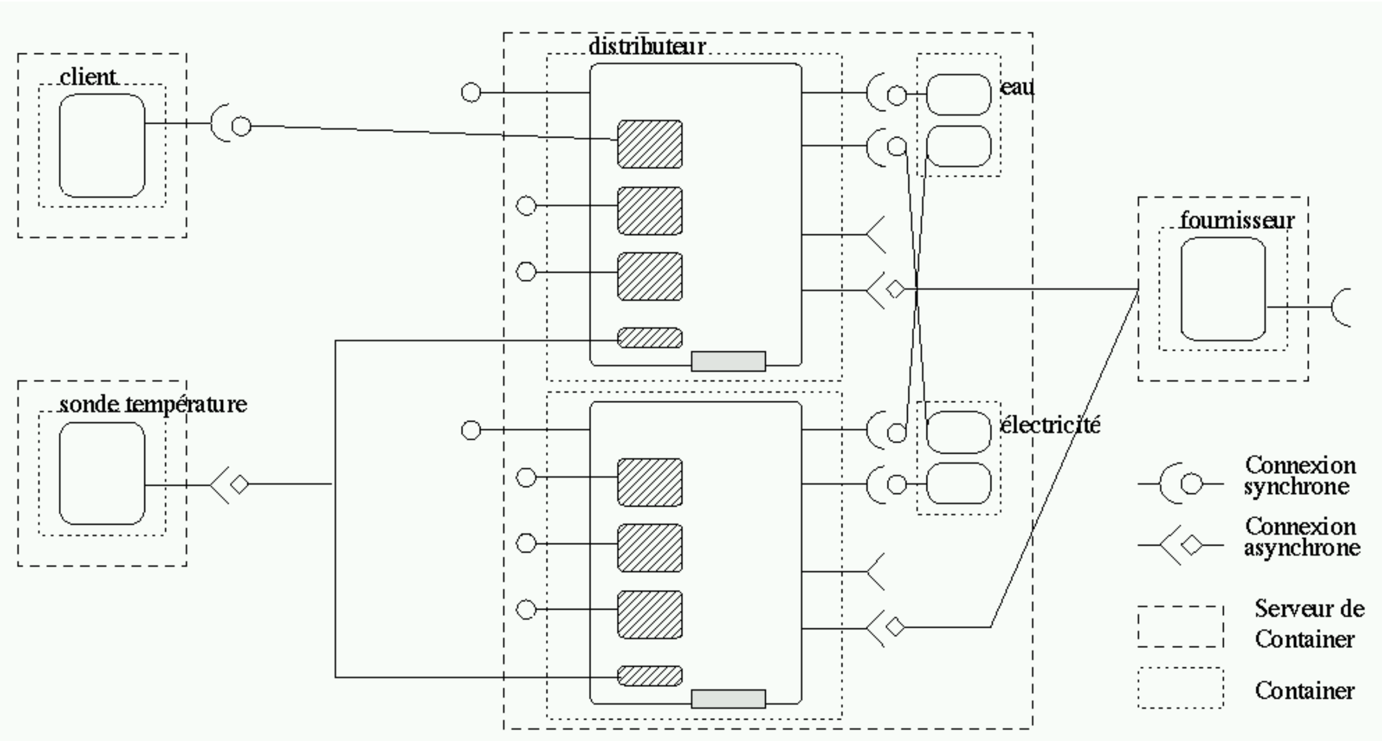


FIGURE 38 Exemple de composition

On peut donc maintenant faire de la composition de composants, assemblage, au déploiement.

Mais pas de la composition hiérarchique, composants composites, par assemblage.

Ceci n'est pas, a priori ou pour l'instant, possible avec EJB, car pas de "Uses" dépendances.

7.3.2 IDL 3 spécifications:

IDL3:

Version étendue du langage IDL pour traiter les composants

Ces définitions vont être “mappé” par un compilateur IDL3 vers OMG IDL2 afin de permettre une implémentation:

- les développeurs des composants les utilisent comme les interfaces à implémenter (contrat à réaliser)
- les clients de composants comme interfaces avec les instances de composants à l’exécution.

D’après:

Raphaël Marvie and Philippe Merle,
CORBA Component Model: Discussion and Use with OpenCCM,
Submitted to Special Issue of the Informatica - An International Journal of Computing and Informatics Dedicated to Component Based Software Development"

Voir :

“Resources about the CORBA Component Model”

http://corbaweb.lifl.fr/OpenCCM/CCM.html

a) Attributes:

Exemple de composant CCM, IDL3:

```
component VendingMachine {
        attribute boolean on ;
        attribute DrinkSeq drinks ;
};
```

OMG IDL2 Mapping of CORBA Component Type Denitions (IDL3)

```
interface VendingMachine :
        Components::CCMObject {

        attribute boolean on ;
        attribute DrinkSeq drinks ;
};
interface CardVendingMachine :
        VendingMachine, CardReader {
// ...
};
```

b) Facets

PROVIDES

Exemple de Facets :

(Server Interface)

OMG IDL3 définition de Facets:

```
interface ClientFacet { ... } ;
interface ProviderFacet { ... } ;
interface RepairmanFacet { ... } ;
component VendingMachine {
        provides ClientFacet client ;
        provides ProviderFacet provider ;
        provides RepairmanFacet repairman ;
};
```

ClientFacet: Nom de Type

client: Nom de Facet

OMG IDL2 Mapping of Facets:

```
interface VendingMachine : Components::CCMObject {
        ClientFacet provide_client () ;
        ProviderFacet provide_provider () ;
        RepairmanFacet provide_repairman () ;
}
```

c) Receptacles:

USES

Permet à un composant d'accepter et d'utiliser une référence.

(Client Interface)

C'est aussi un moyen d'expliciter la dépendance d'un composant vis à vis d'une autre interface, c'est à dire d'un autre composant.

Ainsi, il sera possible d'assembler les composants au déploiement, éventuellement de changer dynamiquement les inter-connexions entre composants.

```
interface PowerPlug { ... } ;
interface WaterPlug { ... } ;
component VendingMachine {
        uses WaterPlug water ;
        uses multiple PowerPlug power ;
};
```

OMG IDL2 Mapping of Receptacles:

```
interface VendingMachine : Components::CCMObject {
        void connect_water (in WaterPlug cnx)
                raises (Components::AlreadyConnected,
                        Components::InvalidConnection) ;
        WaterPlug disconnect_water ()
                raises (Components::NoConnection) ;
                        WaterPlug get_connection_water () ;
        struct powerConnection {
                PowerPlug objref ;
```

```
            Components::Cookie ck ;
        };
typedef sequence<powerConnection> powerConnections;

Components::Cookie connect_power (in PowerPlug cnx)
        raises (Components::ExceededConnectionLimit,
                    Components::InvalidConnection) ;
PowerPlug disconnect_power (in Components::Cookie ck)
        raises (Components::InvalidConnection) ;
powerConnections get_connections_power () ;
};
```

d) Events:

EMITS

PUBLISHES

Push, or pull

Définition IDL3 de 2 sources d'événements et un puis (sink):

```
valuetype NoChangeEvt : Components::EventBase { ... } ;
valuetype EmptyEvt : Components::EventBase { ... } ;
valuetype TemperatureEvt : Components::EventBase { ... } ;
component VendingMachine {
        emits NoChangeEvt change ;
        publishes EmptyEvt empty ;
        consumes TemperatureEvt temp ;
};
```

--> Mapping ...

OMG IDL2 Mapping of Events: Source et Sink

Event Sources:

```
interface NoChangeEvtConsumer :
                                Components::EventConsumerBase {
        void push (in NoChangeEvt evt) ;
};
interface EmptyEvtConsumer :
                                Components::EventConsumerBase {
        void push (in EmptyEvt evt) ;
};
interface VendingMachine :
        Components::CCMObject {
void connect_change
                                (in NoChangeEvtConsumer consumer)
        raises(Components::AlreadyConnected) ;
NoChangeEvtConsumer disconnect_change ()
        raises(Components::NoConnection) ;
Components::Cookie subscribe_empty
                                (in EmptyEvtConsumer consumer)
        raises (Components::ExceededConnectionLimit) ;
EmptyEvtConsumer unsubscribe_empty
                                (in Components::Cookie ck)
        raises(Components::InvalidConnection) ;
};
```

Event Sinks:

```
interface TemperatureEvtConsumer :
                                Components::EventConsumerBase {
        void push (in TemperatureEvt evt) ;
};
interface VendingMachine : Components::CCMObject {
        TemperatureEvtConsumer get_consumer_temp () ;
};
```

Résumé:

Composant IDL3 pour le distributeur de boissons:

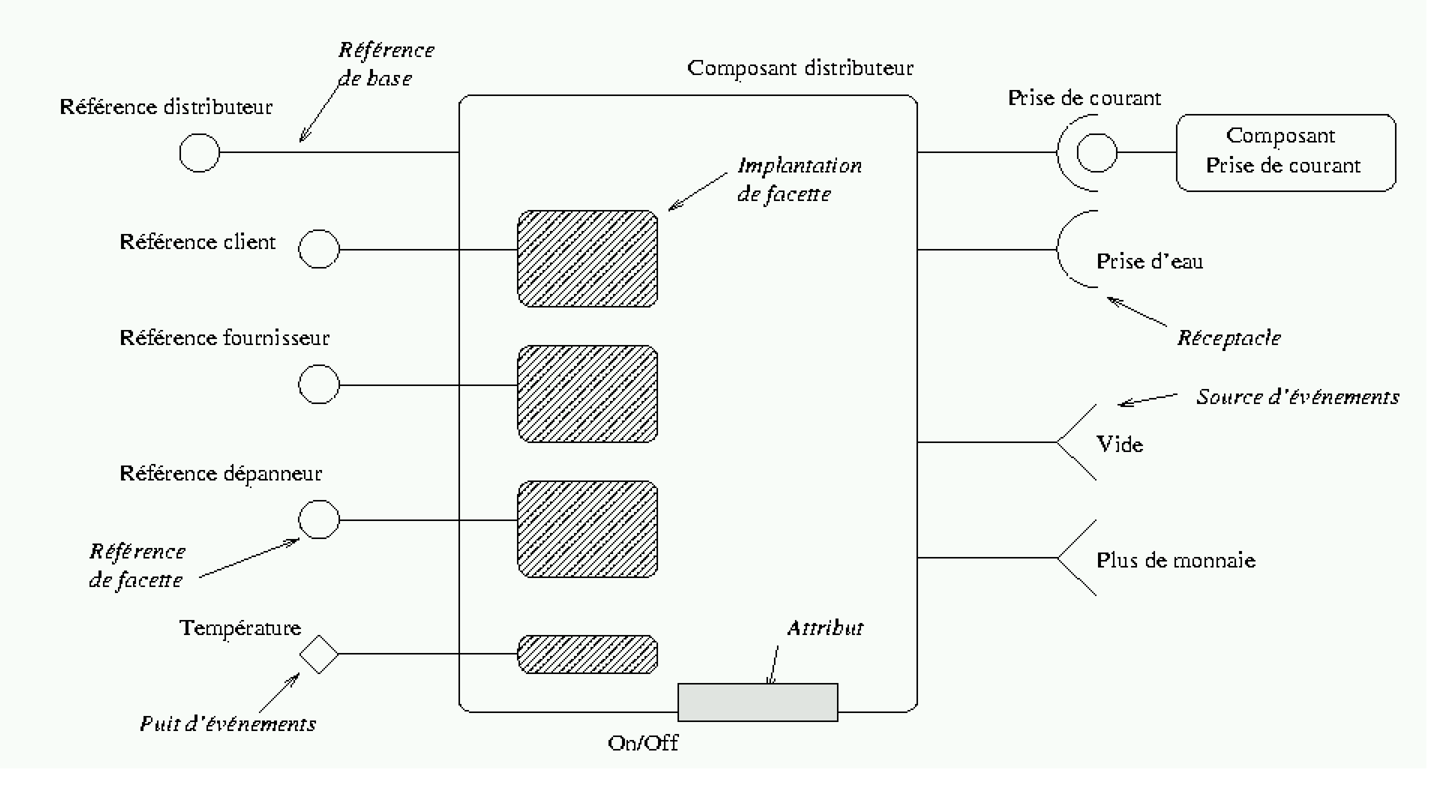
```
component VendingMachine {
    attribute boolean on;
    readonly attribute DrinkSeq drinks;

    provides Client client;
    provides Supplier supplier;
    provides Repairman repairman;

    consumes TemperatureEvt temp;

    uses WaterSupply water;
    uses multiple PowerSupply power;

    emits NoMoreChangeEvt noChange;
    publishes EmptyEvt empty;
};
```



7.3.3 Framework de développement: CIF

Un framework:

CIF: Component Implementation Framework

permet de décrire comment les parties fonctionnelles et non-fonctionnelles interagissent

CIDL: Component Implementation Definition Language

Permet de décrire l'implémentation d'un composant.

Exemple de définition CIDL

```
composition entity VendingMachineImpl {
    home executor VendingMachineHomeImpl {
        implements VendingMachineHome ;
        manages VendingMachineImpl ;
    };
};
```

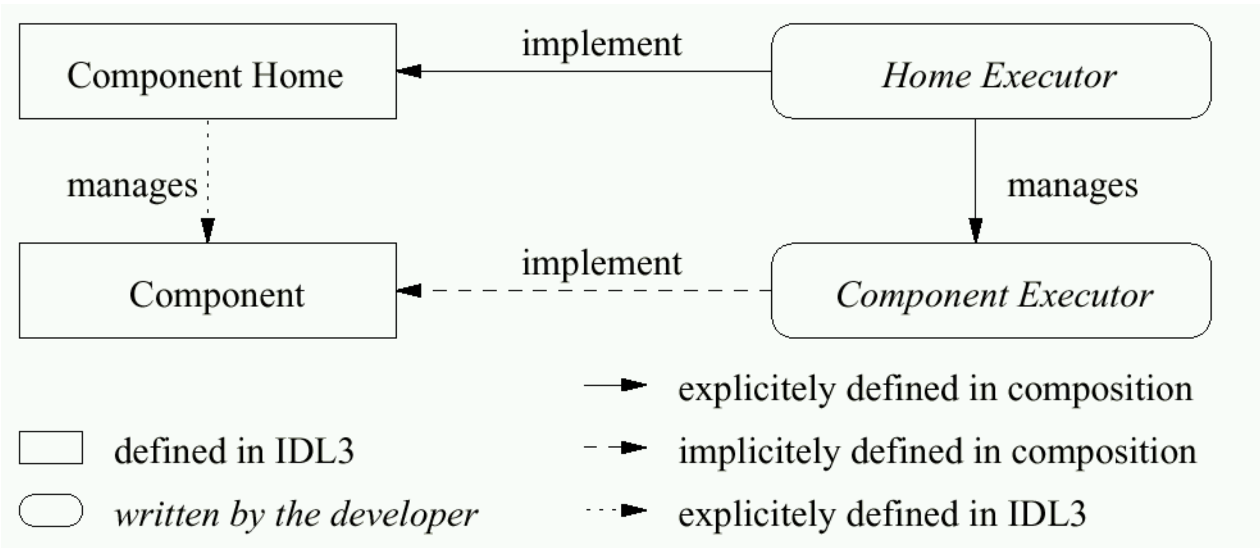


FIGURE 39 Role des specifications CIDL

;

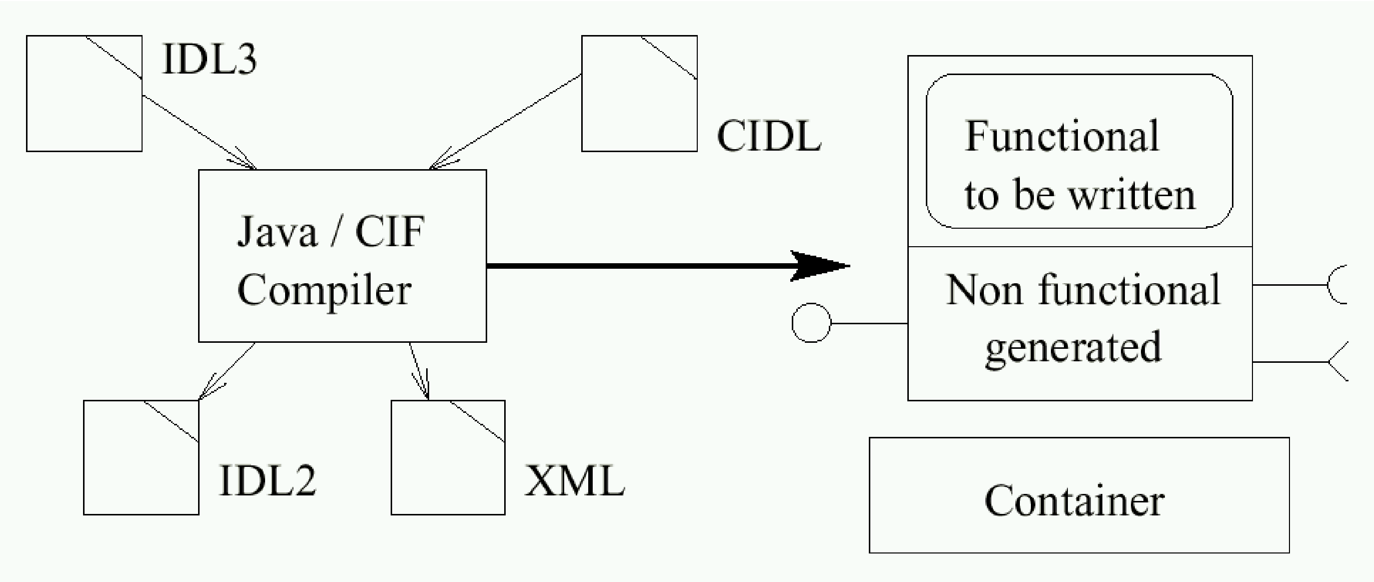


FIGURE 40 Génération des stub/proxy des composansts

Fonctionnel, Modules

Objets

Composants

Service

..... Service-Oriented Architecture ??

CHAPITRE 8 Conception et paternes pour la programmation répartie:

8.1 Principes et objectifs

8.2 Exemples de paterns

8.2.1 Acceptor et Connector

8.2.2 Thread par session

8.2.3 Thread par requête

8.2.4 Objets actifs vs. Moniteur

8.2.5 Services asynchrones

8.2.6 Migration (mobilité)