USTL – Master mention Informatique – M1

Construction d'applications réparties

VIII. Applications réparties et composants

Lionel.Seinturier@lifl.fr

Plan

1. Code mobile

1. Code mobile

Définition
- Assurer les interactions en déplaçant le code à exécuter pour rapprocher le traitements des données et réduire le volume d'échange

Exemples
- Requêtes SQL, Applet Java, sérialisation d'objets Java

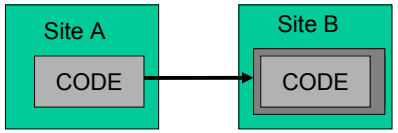
Utilisation
- Collecte d'information disséminées dans un réseau
- Surveillance de données
- Administration de réseaux
- Reconfiguration, placements
- Réseaux actifs
- Workflow
- Négociation
- Jeux en réseaux
- ….

1. Code mobile

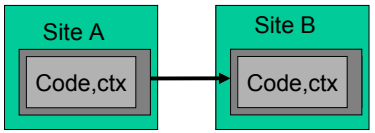
Des problèmes
- Interopérabilité des codes exécutables
  - Même architecture de machine, de système
  - Utilisation d'interpréteurs (ex JVM)
- Sécurité (avec méfiance mutuelle)
  - Du site acceptant le code
  - De l'agent mobile
- Structuration des applications à base de code mobile
  - Intégration avec d'autres approches C/S
  - Transparence de la programmation en code mobile
- Partage des ressources (cohérence ?)

1. Code mobile

Migration faible ou forte
- Code à la demande : mobilité faible – on bouge du code sans contexte variable (Exemple applet java)



- Agents Mobiles : mobilité forte – on bouge du code exécutable, des données locales, un contexte d'exécution



1. Code mobile

Avantages
- Dynamique, adaptabilité
- Exécution à distance
- Code à la demande

Domaine porteur
- Nombreux prototypes de recherche
- Concepts en cours d'évolution

Des problèmes
- Sécurité
- Environnements de développement
- Applications à réaliser

2. Mémoire partagée répartie

2. Mémoire partagée répartie

Concept

- Mémoire commune répartie commune = espace de communication
- Les interactions passent par cette mémoire

Exemples

- Opérations R/W sur des pages (Treadmark)
- Module à espaces de tuples (BD répartis, javaspaces)
- Module à espaces d'objets répartis partagés

Motivation

- Programmation de la coopération par variables partagées en masquant les difficultés de la répartition
- Offrir au développeur les conditions de travail de l'espace centralisé

2. Mémoire partagée répartie

Les problèmes

- Les performances des réalisations
- Support des diverses cohérences
  - Cohérences fortes
    - Atomique, séquentielle
  - Cohérences faibles
    - Causale, relâchée
- Support des outils de développement
  - Langages, compilateurs, debug

2. Mémoire partagée répartie

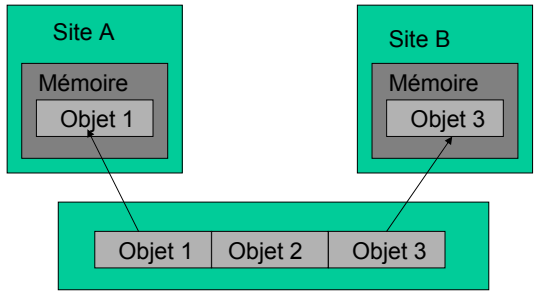
Principes de réalisation

- Simulation d'un espace mémoire unique dans un ensemble d'espaces mémoires réels distribués

Espace d'objets répartis partagés

Interface d'accès commune (nomage, invocations,…)

Mode de réalisation par projection de l'espace virtuel partagé dans l'espace mémoire réel des dif. sites

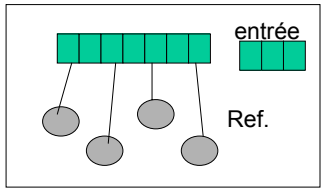


Espace virtuel commun

2. Mémoire partagée répartie

Exemple : JavaSpace

- Un JavaSpace = ensemble de tuples ou entrées
- Une entrée : un ensemble de références à des objets Java
- Les opérations de base
  - Écriture dans un javaspace (write)
  - Lecture dans un javaspace (read)
  - Retrait dans un javaspace (take)
  - Notification de l'écriture d'une entrée (notify)
  - Transaction : regroupement de plusieurs opérations avec le respect des propriétés ACID



2. Mémoire partagée répartie

Avantages
- Facilité de développement d'une application répartie ou de portage d'une application centralisée en univers réparti
- Performances excellentes lorsque l'objet est chargé localement

Inconvénients
- Performances mauvaises lorsque l'objet est distant (les cohérences fortes sont très coûteuses en réparti)
- Les cohérences faibles sont plus performantes mais délicates d'emploi
- Approche peu ouverte : un seul langage interprété ou une seule forme de présentation de données… sinon convertir les données à chaque déplacement

3. Composants

3. Composants

Passé (année 1990) : objet mouvance « à la CORBA »

Besoins
- configuration
- déploiement
- empaquetage (*packaging*)
- assemblage
- dynamicité
- gestion des interactions et des dépendances

Présent : plusieurs tendances
- composant, aspect, MDE, réflexivité

3. Composants

Composant vs objet

- plus haut niveau abstraction
- meilleure encapsulation, protection, autonomie
  - programmation + systématique + vérifiable
- communications plus explicites
  - port, interface, connecteur
- connectables
  - schéma de connexion (ADL) : « plan » applicatif
- séparation « métier » - technique
- meilleure converture du cycle de vie
  - conception, implémentation, *packaging*, déploiement, exécution

Définition composant

- 1ère apparition terme [McIlroy 68]
- 30 ans + tard : Sun EJB, OMG CCM, MS .NET/COM+, …
- recensement [Szyperski 02] : 11 définitions +/- ≡

A component is a unit of composition with **contractually specified interfaces** and **context dependencies** only. A software component can be **deployed** independently and is subject to **composition** by third parties. [Szyperski 97]

3.1 CORBA Component Model

Plan

1. Composant

Modèle de composant CORBA

Un composant CCM possède
• 1 ou +sieurs interfaces (appelées **port**)
• une **fabrique** (appelée *home*)

• 3 types de ports (facette, réceptacle, événement)
• un composant possède une référence de base (≡ IOR pour un objet CORBA)
• tous les composants héritent du composent de base `ComponentBase`
(≡ `CORBA::Object` pour les objets)
• les composants se déclarent en IDL3 (extension de l'IDL de base)

```
component Distributeur { ... };
```

• traducteur IDL3 → IDL

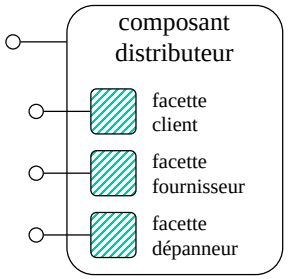
1.1 Facette

Facette = interface fournit par un composant CORBA

- facette fournit un point de vue sur un composant
- permet à +sieurs clients d'avoir des points de vue ≠ sur un même composant
- chaque facette possède une référence et correspond à une interface IDL

```
interface facadeClient { ... };
interface facadeFournisseur { ... };
interface facadeDepanneur { ... };

component distributeur {
   provides facadeClient fCli;
   provides facadeFournisseur fFour;
   provides facadeDepanneur fDep;
};
```



Existence de méthodes de navigation/recherche (par clé) de facettes

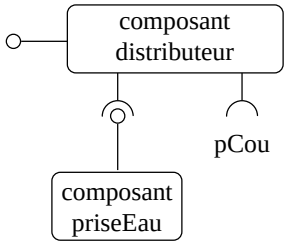
1.2 Réceptacle

Réceptable = point de connexion entre composants

- permet de créer des liens entre composants
- 2 types de cx : simples ou multiples
- utilise des méthodes de connexion et de déconnexion dynamiques

```
interface priseCourant { ... };
interface priseEau { ... };

component distributeur {
   uses multiple priseCourant pCou;
   uses pEau;
};
```



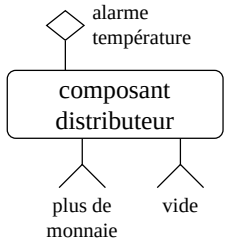
1.3 Evénement

Ports de réception/émission d'evts asynchrones

- 2 types de ports : producteur/consommateur (ou source/puits)
- 2 types de producteurs : 1-1 et 1-n
- les clients s'abonnent à des types d'évts
- modèle de communication de type *push*
- compatible avec les COSEvent et COSNotification

```
interface plusDeMonnaie: EventBase { ... };
interface distributeurVide: EventBase { ... };
interface alarmeTemperature: EventBase { ... };

component distributeur {
   emits plusDeMonnaie pdm;        // 1-1
   publishes distributeurVide dv;  // 1-n
   consumes alarmeTemperature at;
};
```



1.4 Fabrique

Gestionnaire du cycle de vie d'un type de composant

- gère les instances d'un composant
- offre des outils de recherche d'instances basés sur des clés
- ≡ intégration au niveau de IDL3 du COSLifeCycle

```
component distributeur { ... };

interface idDistributeur: PrimaryKeyBase { long identifiant; };
exception clientInconnu {};

home maisonDeDistributeurs
   manages distributeur
   primaryKey idDistributeur {
      factory fabriqueDeDistributeurs( in string clientName );
      finder repertoireDeDistributeurs( in string clientName )
         raises ( clientInconnu );
};
```

2. Modèle de déploiement

Adapté du langage OSD (Open Software Description)

- soumis au W3C par Marimba et Microsoft
- une DTD XML pour décrire comment un logiciel peut être installé

Modèle de déploiement du CCM

- fichier archive (.car) contenant
  - une description du composant = 4 documents XML
    - Software Package Descriptor
    - CORBA Component Descriptor
    - Component Assembly Descriptor
    - Property File Descriptor
  - une ou +sieurs implantations (pour des syst./lang. ≠)
  - un fichier d'état des propriétés du composant

2.1 Soft Pkg Descr

Software Package Descriptor (.csd)

- informations générales sur le composant
- 1 ou plusieurs sections sur chaque implantation

```
<softpkg name="Bank" version="1,0,1,0">

<pkgtype>CORBA Component</pkgtype>
<title>Bank</title>
<author>
  <company>Acme Component Corp.</company>
  <webpage href="http://www.acmecomponent.com/>
</author>

<description>Yet another bank example</description>
<license href="http://www.acmecomponent.com/license.html"/>
<idl id="IDL:M1/Bank:1.0"><link href="ftp://x/y/Bank.idl"/></idl>
<propertyfile><fileinarchive name="bankprops.cpf"/></propertyfile>
```


2.1 Soft Pkg Descr

```
<implementation id="DCE:700dc518-0110-11ce-ac8f-0800090b5d3e">
  <os name="WinNT" version="4,0,0,0" />
  <os name="Win95" />
  <processor name="x86" />
  <compiler name="MyFavoriteCompiler" />
  <programminglanguage name="C++" />
  <dependency type="lib"><name>ExLIB</name></dependency>
  <descriptor type="CORBA Container">
    <fileinarchive>processcontainer.ccd</fileinarchive>
  </descriptor>
  <code type="DLL">
    <fileinarchive name="bank.dll"/>
    <entrypoint>createBankHome</entrypoint>
  </code>
  <dependency type="DLL">
    <localfile name="rwthr.dll"/>
  </dependency>
</implementation>

<implementation> <!-- another implementation --> </implementation>

</softpkg>
```


2.1 Soft Pkg Descr

Principaux éléments

<idl>	l'interface du composant
<implementation>	informations sur une implantation
<dependency>	dépendances (logiciels nécessaires pour l'exécution)
<descriptor>	référence une description de composant (.ccd)

2.2 CORBA Comp Descr

CORBA Component Descriptor (.ccd)

- décrit un composant
- fait référence à un élément `<descriptor>` du *software package descriptor*
- généré (en grande partie) à partir de la **définition IDL3** du composant
- informations sur
  - les **interfaces** du composant
  - les **ports fournis** par le composant
  - les **ports utilisés** par le composant

2 parties

- caractéristiques générales sur l'exécution du composant (fixé par le développeur)
- structure du composant (générée à partir de l'IDL3)

2.2 CORBA Comp Descr

```
<corbacomponent>

<persistence responsibility="container" usepss="true">
  <persistentstoreinfo datastorename="oracle"
                       datastoreid="mainofficedb" />
</persistence>

<transaction use="supports" />
<eventpolicy emit="normal" />
<threading policy="multithread" />

<ports>
<provides providesname="book_search" repid="IDL:BookSearch:1.0" />
<provides providesname="shopping_cart" repid="IDL:CartFactory:1.0" />
<uses usesname="ups_rates" repid="IDL:ShippingRates:1.0" />
<emits emitsname="low_stock" eventtype="StockRecord" />
</ports>

</corbacomponent>
```


2.2 CORBA Comp Descr

Principaux éléments

<persistence>	infos sur la persistance nécessaire au composant
<transaction>	infos sur le niveau de transaction nécessaire au comp.
<eventpolicy>	infos sur la façon dont sont gérés les événements
<threading>	politique de concurrence du conteneur accueillant le comp.
<ports>	infos sur les ports du composant

2.3 Comp Ass Descr

Component Assembly Descriptor (.cad)

- un **assemblage** de composants est un ensemble de composants
  - **interconnectés**
  - **répartis** logiquement sur un ensemble de machines

- lors du déploiement (= installation)
  un assemblage est **projeté** sur une configuration φ
  - plusieurs configurations sont éventuellement satisfaisantes
  - en choisir une en fonction de critères fournit par l'installateur

- le *component assembly descriptor* décrit la configuration initiale
  - des instances de composants
  - des fabriques
  - des connexions

2.3 Comp Ass Descr

```
<componentassembly id="ZZZ">
<componentfiles>
  <componentfile id="A"><fileinarchive name="ca.car"/></componentfile>
  <componentfile id="B"> ... </componentfile>
</componentfiles>
<partitioning>
  <componentplacement id="Aa"> ... </componentplacement>
  ...
</partitioning>
<connections> ... </connections>
</componentassembly>
```

<componentfile>	composants utilisés lors de l'assemblage
<componentplacement>	placement des composant
<connections>	connexions entre composants

2.4 Prop File Descr

Property File Descriptor (.cpf)

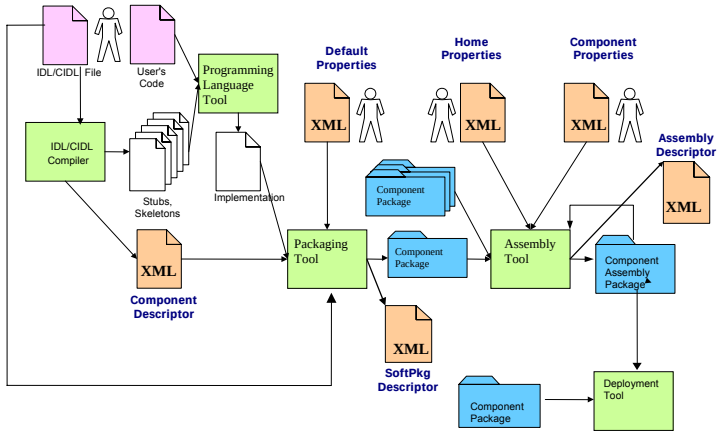
- utilisé au moment du déploiement pour configurer les instances du composant et les fabriques

```
<properties>
  <simple name="bufSize" type="long"><value>4096</value></simple>
  <sequence name="niceGuys" type="sequence<string>">
    <simple type="string"><value>Dave</value></simple>
    <simple type="string"><value>Ed</value></simple>
  </sequence>
</properties>
```

<simple>	type de propriété basique
<value>	valeur pour la propriété
<sequence>	propriété de type tableau

2.5 Vue générale

Cycle de développement d'une application avec le CCM



3. Modèle d'exécution

Conteneurs CCM

Les composants s'exécutent dans une structure d'accueil

POA spécialisé pour composants

- supportant l'activation/passivation d'instances
- fournissant un lien direct avec les services
  - d'événements
  - de transaction
  - de sécurité

4. Conclusion

Conclusion

∃ indéniablement un marché pour les composants logiciels

CCM vs EJB

+ complet
- avancé
+ complexe (trop ?)

CCM

• Qques implantations opérationnelles CCM vs. qques **dizaines** pour EJB