Composants logiciels (suite)
Web Services
CCM

Sacha Krakowiak
Université Joseph Fourier
Projet Sardes (INRIA et IMAG-LSR)

http://sardes.inrialpes.fr/people/krakowia

Services Web (1)

- **Motivations**
  - Intégration d'applications "à gros grain"
  - Unité d'intégration : le "service" (interface + contrat)
- **Contraintes**
  - Applications conçues indépendamment, sans avoir prévu une intégration future
  - Applications hétérogènes (modèles, plates-formes, langages)
- **Conséquences**
  - Pas de définition d'un modèle commun, seulement de protocoles et d'interfaces
  - Utilisation d'une base commune de niveau élémentaire
    - Pour les protocoles d'accès aux services
    - Pour la description des services
  - Outil de base : XML (car adaptable)

Services Web (2)

- **Objectifs des Services Web**
  - Permettre l'interopérabilité des applications, indépendamment des plates-formes et des langages
  - Permettre le couplage faible des applications (évolution indépendante) et leur coopération via des interfaces de haut niveau d'abstraction (services globaux)
  - Permettre une coopération des applications avec un minmum d'intervention humaine
- **Conséquences**
  - Un service doit être auto-descriptif
  - Un service doit pouvoir être facilement trouvé (à partir d'une description)

Les Services Web sont définis par un ensemble de standards qui permettent de décrire des interfaces logicielles et d'accéder aux fonctions correspondantes sur un réseau via des messages exprimés en XML. Les constructions ainsi réalisées constituent une architecture à base de services (*Service Oriented Architecture*, ou SOA)

Services Web (3)

- **Apport conceptuel**
  - Pas de nouveaux concepts fondamentaux …
  - … alors, quelle est la nouveauté ?
- **Apport concret**
  - Avant tout, solution en vue au problème de l'hétérogénéité
  - Vers un schéma d'intégration et de coordination d'applications à grande échelle
  - Forte implication des principaux acteurs du domaine
  - Beaucoup de problèmes techniques restent encore à résoudre

Brefs rappels sur XML (1/5)

- XML = *eXtensible Markup Language*
- Motivation
  - Fournir un formalisme pour la description de données structurées (essentiellement structure d'arbre)
- Choix de base
  - Fournir des éléments pour la construction de langages spécialisés plutôt qu'un langage universel. Avantages :
    - formalisme adaptable à des situations très diverses
    - formalisme extensible
- Origines et historique
  - Initialement inspiré par le problème de l'évolution d'HTML ; dérivé de SGML (*Standard Generalized Markup Language*), utilisé pour la description de documents structurés
  - Standard du W3C (depuis février 1998), voir http://www.w3.org/XML/

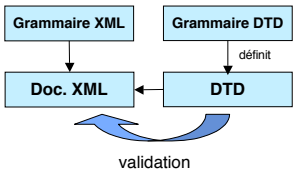
Brefs rappels sur XML (2/5)

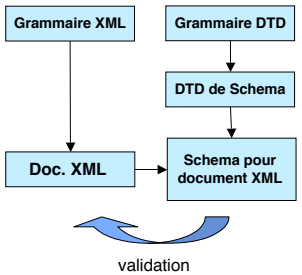
- Document XML
  - Forme
    - une chaîne de caractères
  - Structure
    - données séparées par des balises
- Propriétés
  - Bonne forme (*well formedness*)
    - Conforme à des règles de syntaxe spécifiées ; concerne la forme du document. Exemple : jeu de caractères, parenthésage complet des balises, attributs entre " ", etc.
  - Validité
    - Contraintes logiques (règles de validation) ; concerne la structure du document. Exemple : conformité à un modèle spécifié

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<adresse>
  <lieu>
      <rue> avenue de l'Europe</rue>
      <numero>655</numero>
  </lieu>
  <ville>Montbonnot></ville>
  <code>38330</code>
</adresse>
```


Brefs rappels sur XML (3/5)

- Validation des documents XML
  - Deux modes possibles de spécification de structure (modèles)
- DTD *(Document Type Definition)*
  - Exprimé dans un formalisme spécifique
- XML *Schema*
  - Exprimé en XML





Brefs rappels sur XML (4/5)

DTD

```
<!ELEMENT rue (#PCDATA)>
<!ELEMENT numero (#PCDATA)>
<!ELEMENT ville (#PCDATA)>
<!ELEMENT code (#PCDATA)>
<!ELEMENT lieu(rue, numero)>
<!ELEMENT adresse(lieu, ville, code)>
```

Schema

```
<element name="rue" type="string"/>
<element name="numero" type="string"/>
<element name="ville" type="string"/>
<element name="code" type="string"/>
<element name="lieu">
   <complexType>
     <sequence>
        <element ref="rue"/>
        <element ref="numero"/>
     </sequence>
   </complexType>
</element>
<element name="adresse">
   <complexType>
     <sequence>
        <element ref="lieu"/>
        <element ref="ville"/>
        <element ref="code"/>
     </sequence>
   </complexType>
</element>
```

Document XML

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<adresse>
  <lieu>
      <rue> avenue de l'Europe</rue>
      <numero>655</numero>
  </lieu>
  <ville>Montbonnot></ville>
  <code>38330</code>
</adresse>
```

Brefs rappels sur XML (5/5)

- **Comment exploiter un document XML ?**
  - 2 APIs couramment utilisées par les applications
    - DOM : *Document Object Model*
      - APIs pour des objets de type document définis par une structure XML ; permet de naviguer dans le document (traversées d'arbre), d'accèder à des parties, etc;
    - SAX : *Simple API for XML*
      - Interface plus simple, même usage
  - Ces interfaces sont créées pour une application, à partir d'un document XML, par des outils spécialisés

Usages de XML dans le *middleware*

- **Base pour la construction d'ADLs**
  - Exemples : ADL Fractal, propositions de standards en cours (ACME)
- **Base pour les Services Web**
  - Description de services
  - Description des protocoles
- **Outils pour la manipulation de données XML**
  - Exemples : outils en CORBA
    - Emballage-déballage de données XML dans *strings* IDL

Forme simple de Service Web : RPC-XML

Description en XML d'un appel de procédure à distance
Le type des paramètres est spécifié dans le schéma XML

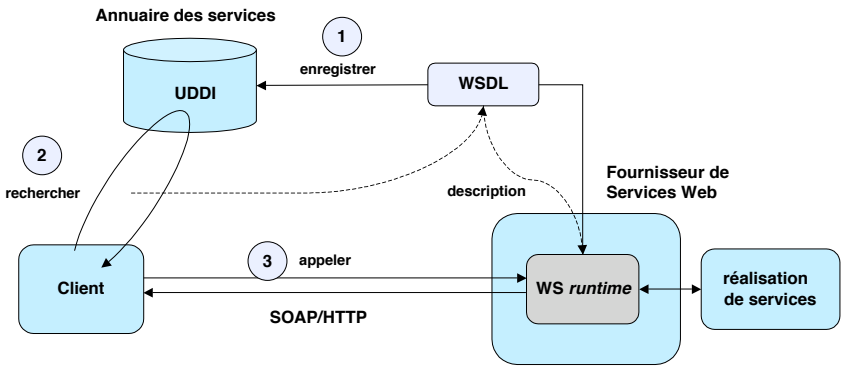
```
<methodCall>
    <methodName>meteo.temperature</methodName>
    <params>
        <param>
            <value>38330</value>
        </param>
    </params>
</methodCall>
```

Description en XML du retour des résultats

```
<methodResponse>
    <params>
        <param>
            <value>12</value>
        </param>
    </params>
</methodResponse>
```

Intérêt : indépendance par rapport aux plates-formes et par rapport au support de communication

Mise en œuvre d'un Service Web

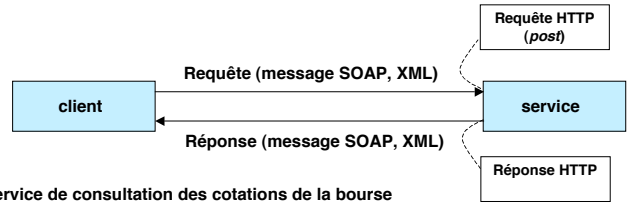


Éléments des Services Web

- **Description d'un service**
  - WSDL : *Web Services Description Language*
  - Notation standard pour la description de l'interface d'un service
- **Accès à un service**
  - SOAP : *Simple Object Access Protocol*
  - Protocole Internet permettant la communication entre applications pour l'accès aux services Web
- **Annuaire des services**
  - UDDI : *Universal Description, Discovery and Integration*
  - Protocole pour l'enregistrement et la découverte de services

SOAP

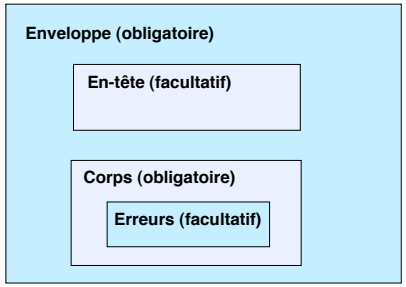
- **Fonction**
  - Un mécanisme simple pour échanger des informations structurées et typées décrites en XML
- **Propriétés**
  - Messages unidirectionnels (un couple de messages peut être utilisé pour réaliser un échange requête-réponse)
  - SOAP peut être réalisé au-dessus d'un protocole quelconque au niveau application (HTTP, FTP, SMTP)



Exemple : service de consultation des cotations de la bourse

Éléments de SOAP

Un message SOAP



Enveloppe (*envelope*) : spécification des espaces de désignation, du codage des données, etc.

En-tête (*header*) : signature (sécurité), informations pour facturation, etc.

Corps (*body*) : élément principal du service : méthodes, paramètres

Erreurs(*fault*) : réactions en cas d'erreur (selon différentes causes)

Un message SOAP sur HTTP (requête)

Prélude pour transport sur HTTP

```
POST /StockQuote HTTP/1.1
Host:www.stockquoteserver.com
Content-Type: text/xml: charset="utf-8"
Content-Length: nnnn
SOAPAction: "SomeURI"
```

```
<SOAP-ENV:Envelope
    xmlns:SOAP-ENV= "http://schemas.xmlsoap.org/soap/envelope/"
    SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"

    <SOAP-ENV:Body>
      <m:getLastTradePrice xmlns:m="Some-URI">
        <symbol>SomeCompany </symbol>
      </m:getLastTradePrice>
    </SOAP-ENV:Body>
</SOAP-ENV:Envelope
```

Message proprement dit

Un message SOAP sur HTTP (réponse)

Prélude pour transport sur HTTP

```
HTTP/1.1 200 OK
Content-Type: text/xml: charset="utf-8"
Content-Length: nnnn
```

```
<SOAP-ENV:Envelope
    xmlns:SOAP-ENV= "http://schemas.xmlsoap.org/soap/envelope/"
    SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"

    <SOAP-ENV:Body>
        <m:getLastrTradePriceResponse xmlns:m="Some-URI">
            < price>34.5 </price>
        </m:getLastTradePriceResponse>
    </SOAP-ENV:Body>
</SOAP-ENV:Envelope
```

Message proprement dit

Messages SOAP

- Un message SOAP est un document XML
- Conséquences
  - Auto-descriptif (modèle en DTD ou Schema XML)
    - Exemple

Schema XML : <element name="price" type="float">
Dans message : <price>6.3</price>

  - Directement lisible (éventuellement utile pour mise au point)
  - L'analyse des messages peut être complexe donc coûteuse

WSDL

- Définition
  - Formalisme pour la description d'un service donnée par le fournisseur
    - Format des messages échangés (SOAP)
    - Relations entre les messages (requête-réponse)
- Propriétés
  - Expression en XML
  - Définition d'interface (cf IDL)
    - Types, messages, opérations, portes
  - Définition des points d'entrée (*endpoints*)
    - Liaisons (protocoles), port, service

Éléments de WSDL

<definitions>	fonctions disponibles, espaces de désignation utilisés dans le reste de la description
<types>	définition des types de données utilisés
<message>	description des paramètres d'appel et de retour
<portType>	description des opérations (utilise les messages)
<binding>	description du mode de transport des messages
<service>	localisation du service fourni (souvent une URL)

Exemple de définition en WSDL (1)

Définition de schéma (par ex. dans http://example.com/stockquote/stockquote.xsd)

```
<?xml version="1.0"?>
<schema
   targetNameSpace="http://example.com/stockquote/schemas"
      xmlns=http://www.w3.org/2000/10/XMLSChema>
   <element name="TradePriceRequest">
      <complexType>
         <all><element name="symbol" type="string"/></all>
      </complexType>
   </element>
   <element name="TradePrice">
       <complexType>
         <all><element name="price" type="float"/></all>
      </complexType>
    </element>
</schema>
```


Exemple de définition en WSDL (2)

Définition en WDSL (par ex. dans http://example.com/stockquote/stockquote.wsdl)

```
<?xml version="1.0"?>
<definitions name = … targetNameSpace=… >
   <import namespace= … location= … />
      <message name ="GetLastTradePriceInput">
            <part name="body" element=xsdl:TradePriceRequest"/>
      < /message>
      <message name ="GetLastTradepriceOutput">
            <part name="body" element= xsdl:TradePrice"/>
      < /message>
      <portType name="StockQuotePortType">
         <operation name="GetLastTradePrice">
            <input message="tns:GetLastTradePriceInput"/>
            <output message="tns:GetLastTradePriceOutput"/>
         </operation>
      </portType>
…
```


Exemple de définition en WSDL (3)

Définition en WDSL (par ex. dans http://example.com/stockquote/stockquote.wsdl) - suite

```
…
 <binding name = "StockQuoteSoapBinding" type="defs:StockQuotePortType">
         <soap:binding style ="document"
    transport=http://schemas.xmlsoap.org/soap/http/>
          <operation name="GetLastTradePrice">
             <soap:operation
                soapAction=http://example.com/GetLastTradePrice/>
            <input>
                soap:body use="literal"/>
            </input>
            <output>
                soap:body use="literal"/>
            </output>
          </operation
 </binding
…
```

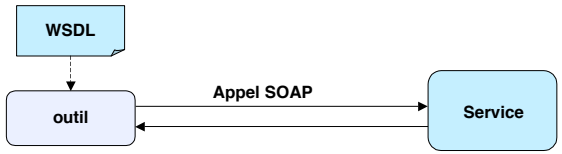

Définition en WDSL (par ex. dans http://example.com/stockquote/stockquote.wsdl) - fin

```
…
   <service name= "StockQuoteService">
      <documentation>First example of simple service </documentation>
      <port
         name="StockQuotePort"
         binding="tns:StockQuoteBinding">
            <soap:address location="http://example.com/stockquote"/>
      </port>
   </service>
</definitions>
```

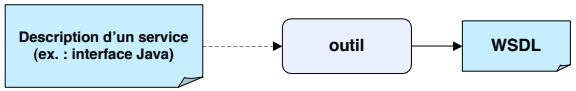
Outils pour Services Web

Des outils pour les services Web commencent à être disponibles

Génération automatique des appels du client



Génération automatique de descriptions WSDL



Conclusion sur Services Web

- **Point fort : interopérabilité d'applications hétérogènes**
  - Mais beaucoup de lacunes fonctionnelles par rapport au *middleware* à composants
    - Pas de modèle de déploiement
      - Donc difficile de diffuser des composants
    - Pas d'environnement standard d'exécution
      - Donc pas de portabilité
    - Pas de projection sur langages de programmation
    - Aspects non-fonctionnels encore peu développés (travaux en cours)
      - Sécurité
      - Qualité de service
      - Transactions
- **Place des Services Web**
  - Le *middleware* "traditionnel" restera pour les applications fortement intégrées (environnement bien maîtrisé)
  - Les services Web serviront à assembler entre elles un ensemble de telles applications
  - C'est une technologie encore jeune, qui doit être consolidée

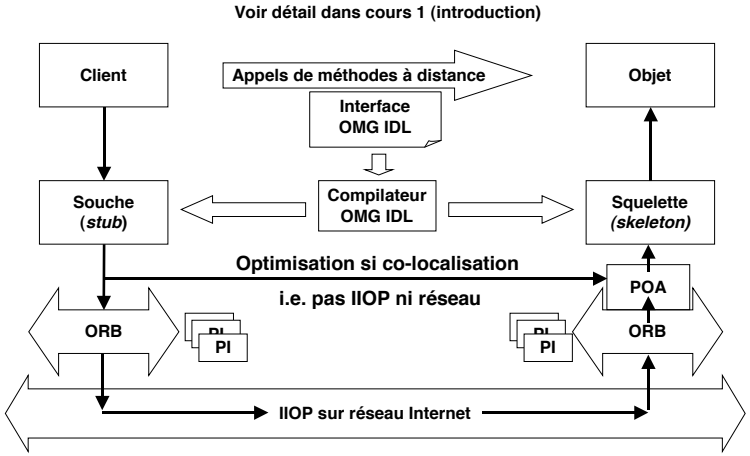
Un autre modèle de composants : CCM
(*CORBA Component Model*)

- **Objet de la présentation**
  - Montrer seulement certains aspects de CCM, en particulier ceux différents des autres modèles de composants
    - en particulier : déploiement d'applications
- **Rappels sur CORBA (*Common Object Request Broker Architecture*)**
  - Spécification d'un modèle d'objets logiciels répartis et des interfaces de l'intergiciel associé
  - Défini par l'OMG (*Object Management Group*)
  - Fait partie d'un ensemble de spécifications (du méta-modèle à l'implémentation)
    - Object Management Architecture (OMA)
      - Des services communs
        - Nommage, courtage, notification, transactions, persistance, sécurité, …
      - Des interfaces orientées métiers / domaines
        - Air Traffic Control, Gene Expression, Software Radio, Workflow Management, …
    - Model Driven Architecture (MDA)
      - Des technologies d'ingénierie dirigée par les modèles
        - Unifed Modeling Language (UML)
        - Meta Object Facilities (MOF)
        - XML Metadata Interchange (XMI)

Introduction à CCM

- **Rappel des objectifs de CORBA**
  - Ouverture
    - Spécification indépendant des fournisseurs et des implantations
  - Hétérogénéité
    - Multi-langages, multi-OSs, multi-réseaux, multi-fournisseurs
    - Via OMG *Interface Definition Language* (OMG IDL)
  - Portabilité
    - Code applicatif indépendant des implantations CORBA
    - Projections standardisées OMG IDL vers langages de programmation
  - Interopérabilité
    - Protocole réseau commun entre implantations CORBA (vu en séance 1)
      - General Inter-ORB Protocol (GIOP)
      - Internet Inter-ORB Protocol (IIOP)
    - Vers d'autres modèles comme OLE, COM et EJB

Introduction à CCM : architecture de CORBA



Motivations pour CCM : les limites de CORBA

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

Apports de CCM

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

Caractéristiques de CCM

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

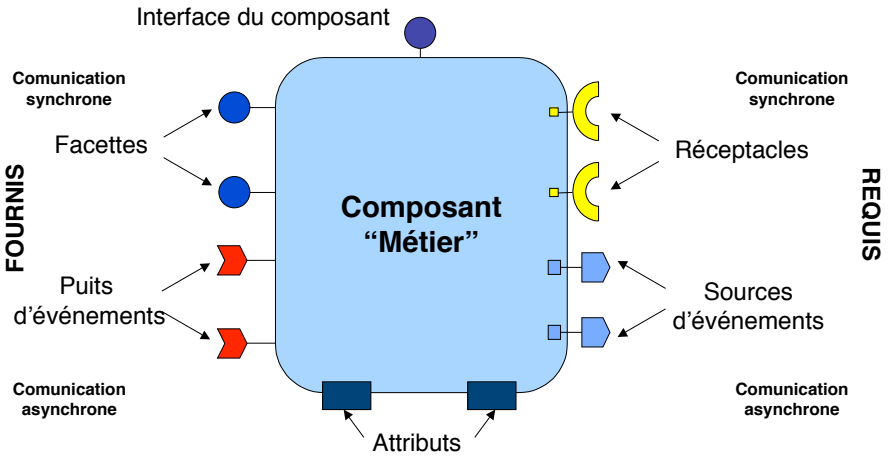
Analogies avec EJB, COM et .NET

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

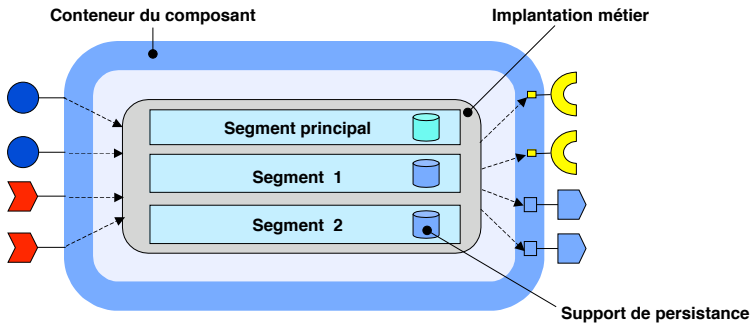
Différences avec EJB, COM et .NET

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

Interfaces d'un composant CCM



Implémentation d'un composant CCM



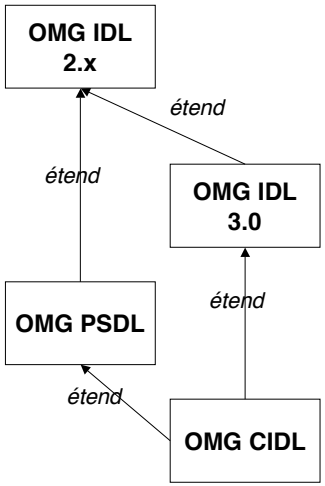
- Implantation de plusieurs ports offerts ayant la même interface
  - Impossible si le composant est implanté par une unique classe
  - Equivalent aux "component parts " dans UML 2.0
- Contrôle fin de la persistance métier
  - Chaque segment peut avoir son propre état persistant stocké sur un support de données distinct

CCM : un cadre global pour la construction d'applications

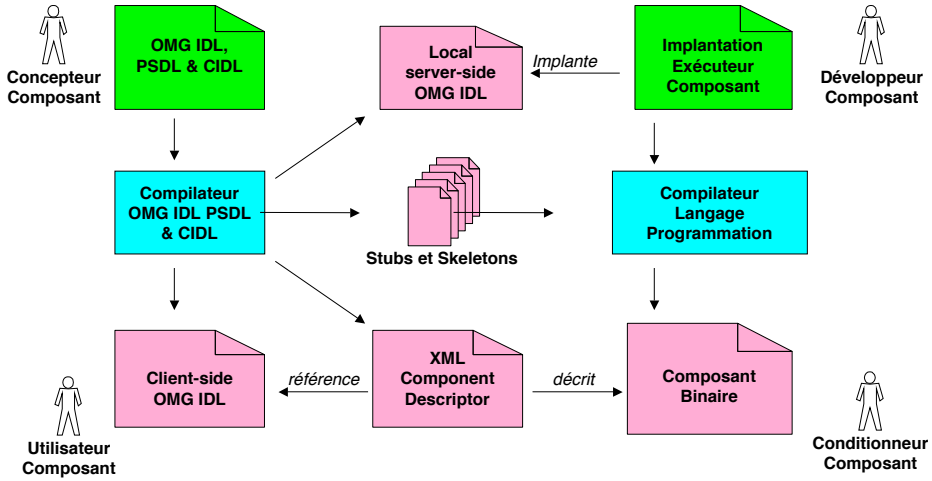
- **Spécification des composants**
  - Via OMG Interface Definition Language (OMG IDL)
    - Description des composants, de leurs ports et des maisons
- **Implantation des composants**
  - Via OMG Component Implementation Definition Language (OMG CIDL)
    - Description du cycle de vie, de la persistance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

Les langages de définition du CCM

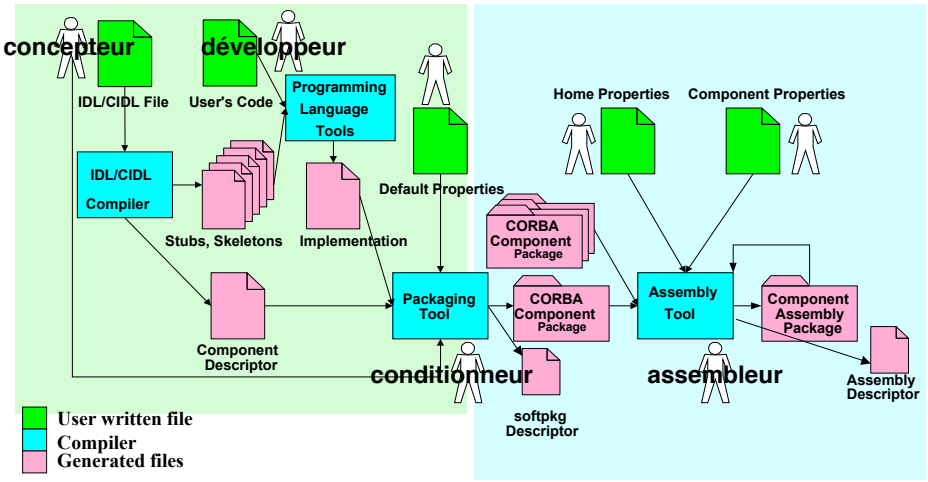
- **OMG IDL 2.x**
  - Modèle orienté objet
  - i.e. types de données, exceptions, interfaces et types valeurs
- **OMG IDL 3.0**
  - Modèle orienté composant
  - i.e. types de composants, maisons et événements
- **OMG PSDL**
  - Définition d'états persistents
  - i.e. types et maisons [abstraites] de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



De la conception au conditionnement des composants CORBA



Conception, développement, conditionnement et assemblage



Construction d'une application CCM

- **Construction d'une application = assemblage de composants**
  - Facette s'assemble avec réceptacle
  - Puits d'événements s'assemble avec source d'événements
  - Compatibilité des interfaces fournies - requises
    - Conformité des types

Types et instances de composants CORBA

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

©2003, Ph. Merle

Maisons de composants

- **Définition**
  - maison (*home*) = fabrique (*factory*) pour un ensemble d'instances de même type
- **Types de maisons**
  - Nouveau méta-type et mot-clé OMG IDL *home*
  - Identification de l'unique type de composants gérés (mot-clé *manages*)
    - Plusieurs types de maisons possibles pour le même type de composants
  - Identification du type de l'identité / clé persistante des composants
    - mot-clé *primarykey* ; pour composants Entity
    - clé primaire identifie les différentes instances de composants dans le contexte d'une maison
  - Opérations `factory` et `finder`
  - N'importe quelle opération métier
  - Héritage simple entre types de maisons
  - Héritage multiple d'interfaces (mot-clé *supports)*
- **Une instance de maison**
  - 1 référence pour l'interface de base, instanciée lors du déploiement
  - Co localisation des instances de composants gérés

©2003, Ph. Merle

Le déploiement CCM

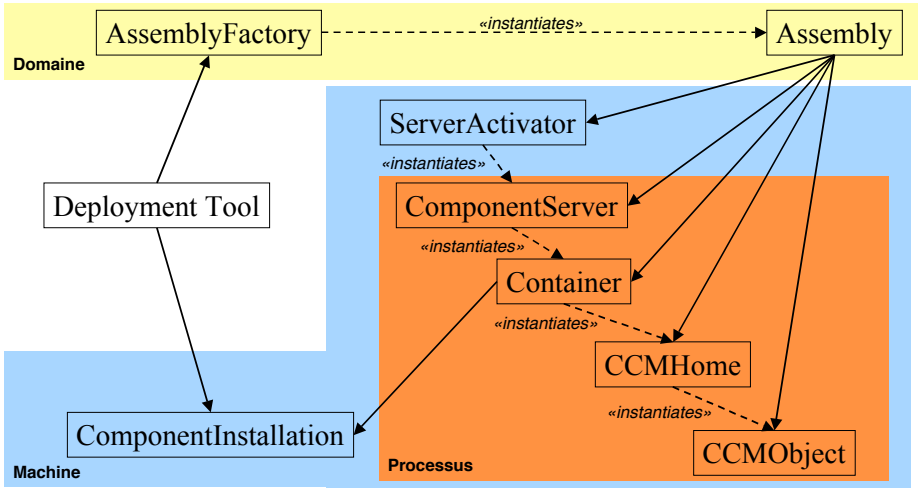
- **Déploiement automatique et réparti d'assemblages de composants CORBA**
  - Installation du code binaire des composants sur leur site d'exécution
    - Demande aux sites de télécharger le code accessible via l'Internet
  - Démarrage des serveurs d'applications nécessaires sur chaque site d'exécution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

©2003, Ph. Merle

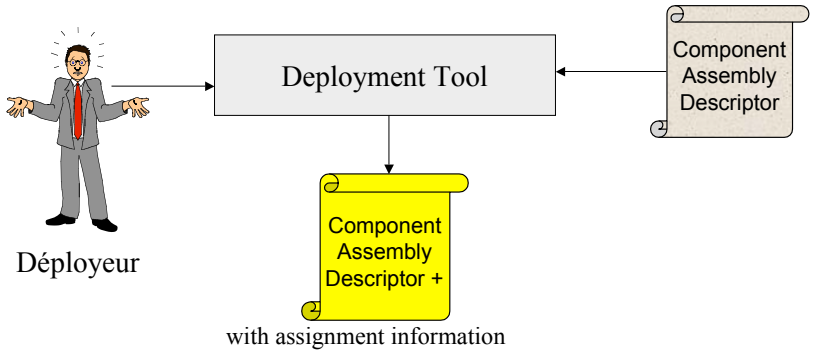
Interfaces de l'infrastructure de déploiement

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

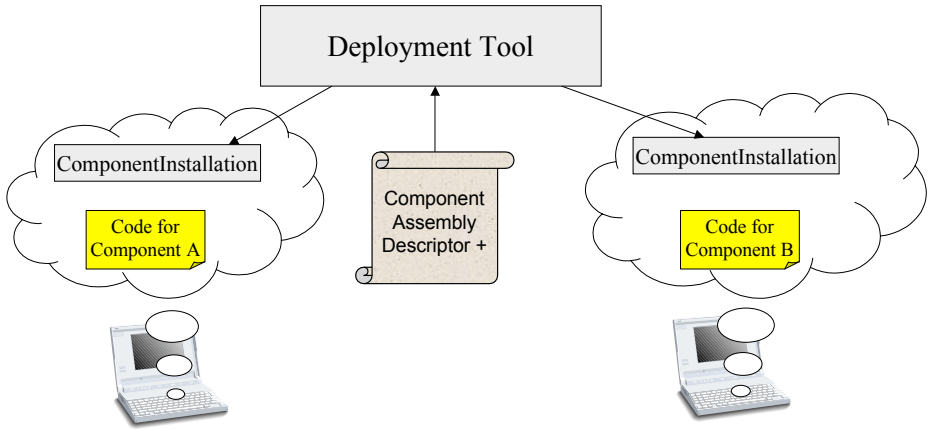
Le processus de déploiement du CCM



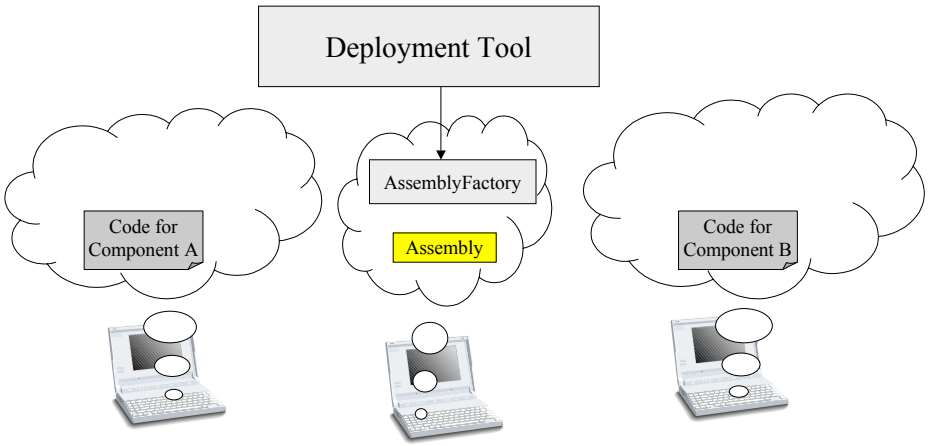
Scénario de déploiement



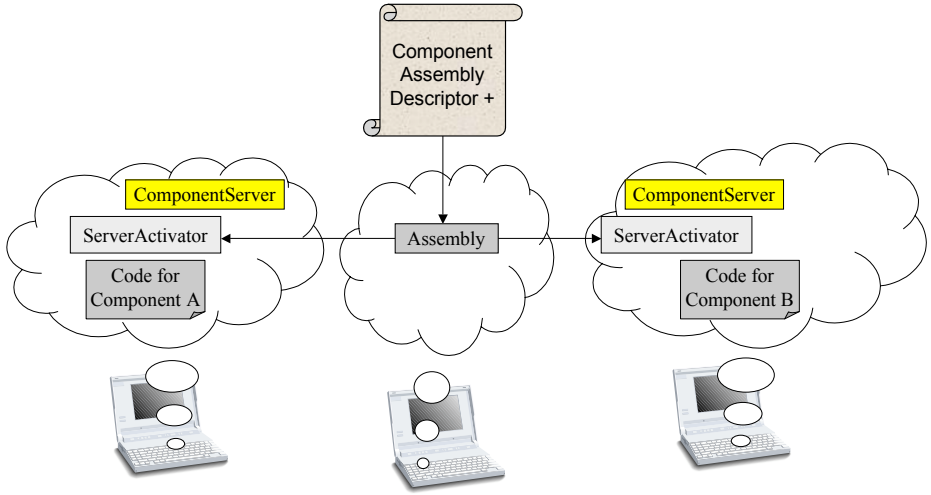
Scénario de déploiement : chargement des implantations



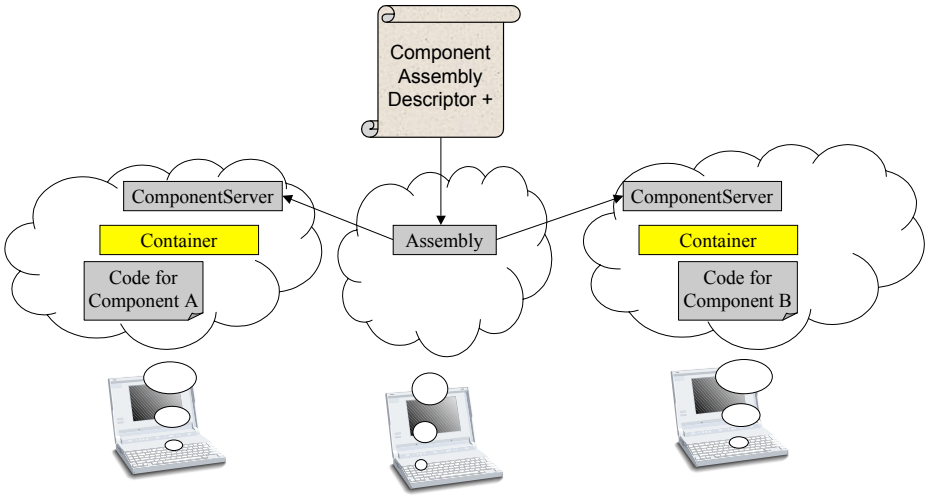
Scénario de déploiement : création de l'objet Assembly



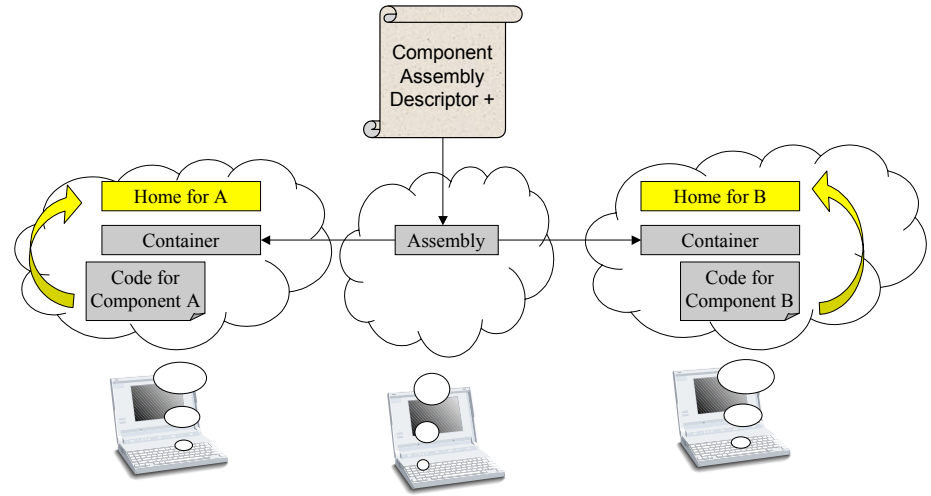
Scénario de déploiement : instanciation des serveurs de composants



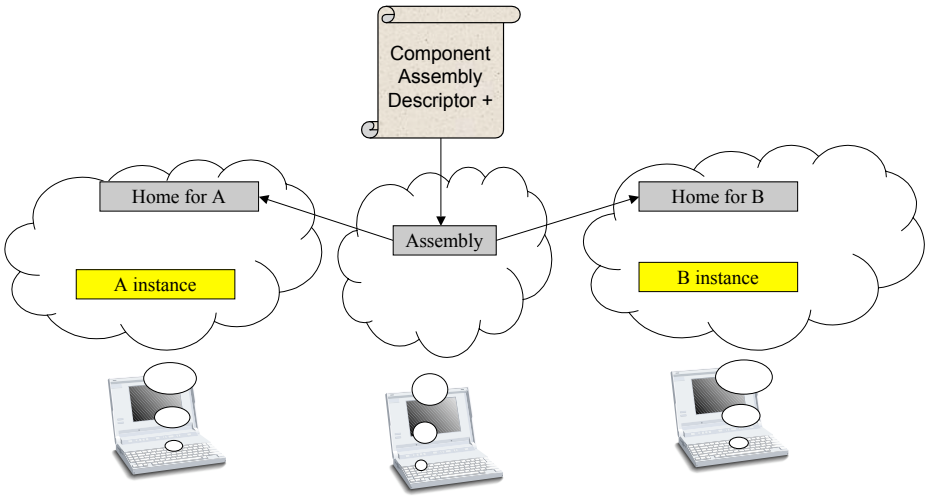
Scénario de déploiement : instanciation des conteneurs



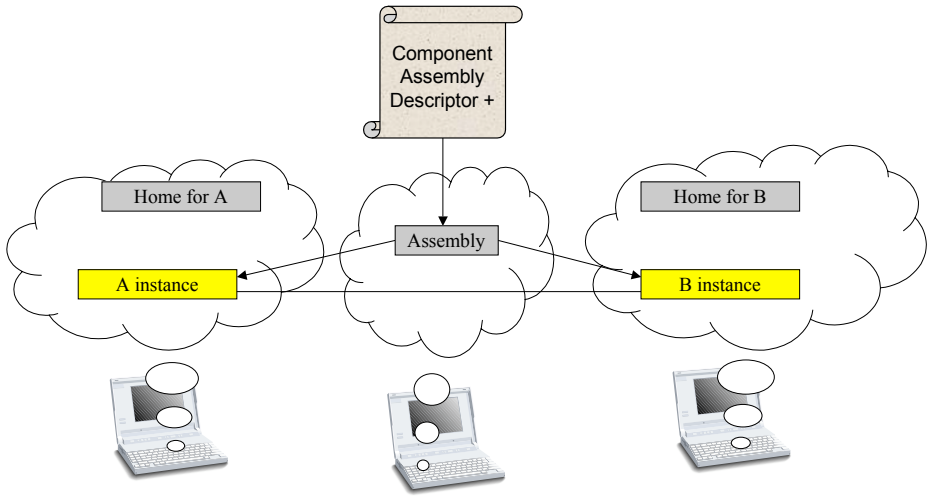
Scénario de déploiement : installation des maisons



Scénario de déploiement : instanciation des composants



Scénario de déploiement configuration des composants



Conclusion sur le modèle de composants CORBA

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