# Introduction aux intergiciels

M. Belguidoum

Université Mentouri de Constantine

**Master2 Académique**

# Plan

# Historique

- Le terme *middleware* est apparu vers 1990 mais le terme intergiciel est apparu autour de 1999-2000
- Des logiciels commerciaux de communication par messages étaient disponibles à la fin des années 1970.
- La notion d'appel de procédure à distance avec sa réalisation en 1978
- Au milieu des années 1980 plusieurs projets développent des infrastructures intergicielles pour objets répartis
- l'OSF (l'*Open Software Foundation*) spécifie une plateforme intergicielle, le Distributed Computing Environment (DCE) en 1996, elle comporte un service d'appel de procédure à distance, un système réparti de gestion de fichiers, un serveur de temps, et un service de sécurité
- L'*Object Management Group* (OMG) est créé en 1989 pour définir des normes pour l'intergiciel à objets répartis

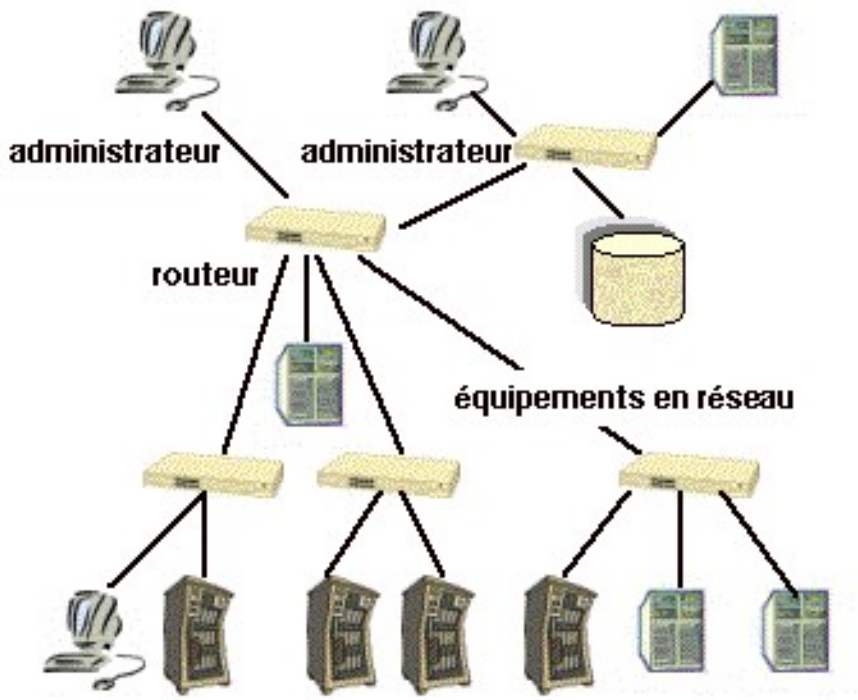
# Historique

- la spécification CORBA 1.0 est la première proposition de l'OMG
- Les propositions ultérieures sont des normes pour la modélisation (UML, MOF) et les composants (CCM)
- L'*Object Database Management Group* (ODMG) définit des normes pour les bases de données à objets, qui visent à unifier la programmation par objets et la gestion de données persistantes.
- La définition du langage Java par Sun Microsystems en 1995 ouvre la voie à plusieurs intergiciels, dont Java *Remote Method Invocation* (RMI) en 1996 et les EJB (Enterprise JavaBeans) en 2002. Ces systèmes et d'autres sont intégrés dans une plate-forme commune J2EE (2005).
- En 1997 Microsoft a développé le *Distributed Component Object Model*(DCOM), un intergiciel définissant des objets répartis composables : COM+ en 1999 et .NET qui est une plateforme intergicielle pour la construction d'applications réparties et de services pour le Web.

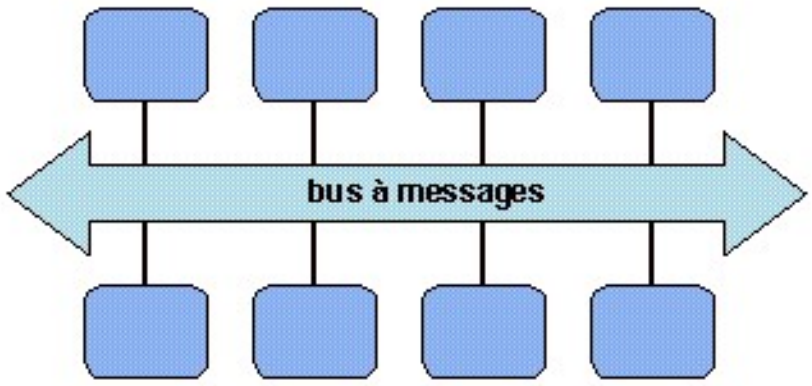
# Système de médiation

- collection d'équipements divers (capteurs) distribués
- différentes tâches : mesure des performances, collecte d'information, maintenance à distance, installation de nouveaux services
- nécessité d'accès à distance au matériel, collecte et agrégation de données, réaction à des évènements critiques
- les systèmes qui réalisent ses tâches sont des systèmes de *médiation*

# Exemple : système de médiation




(a) Organisation physique



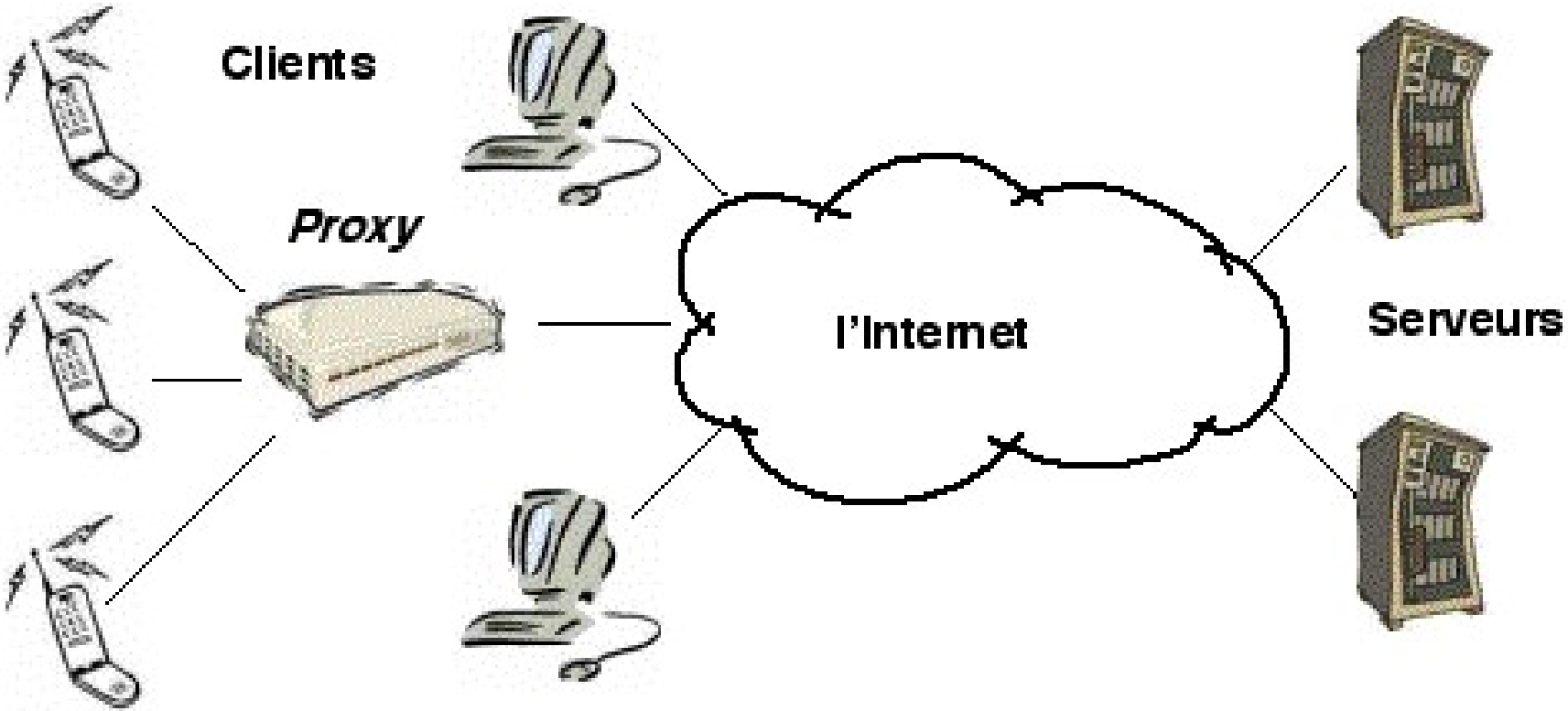

(b) Organisation logique

# Exemple : adaptation de clients par des mandataires

## Proxy ou mandataire

est une **couche d'adaptation** entre les clients et les serveurs. La fonction du mandataire est d'**adapter** les caractéristiques du flot de communication **depuis** ou **vers** le client aux capacités de son point d'accès et aux conditions courantes du réseau. Le mandataire utilise ses propres ressources de stockage et de traitement. Les mandataires peuvent être hébergés sur des équipements dédiés, ou sur des serveurs communs.

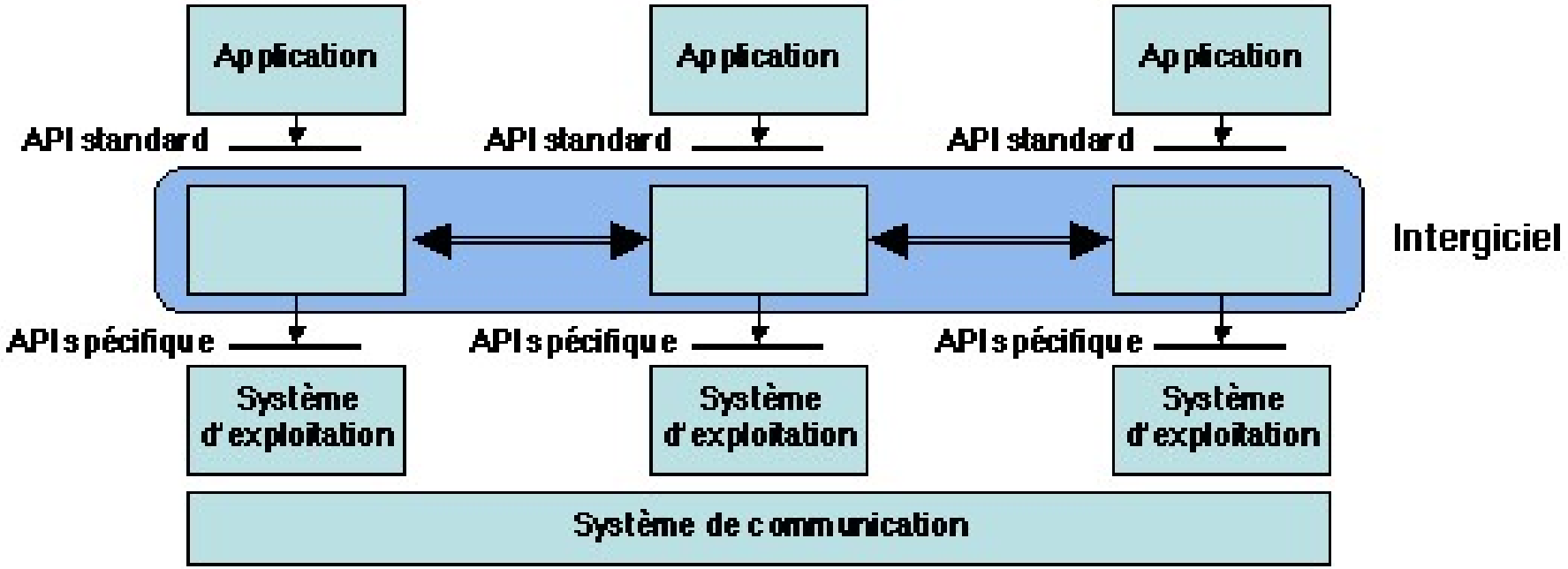
# Exemple : adaptation de clients par des mandataires



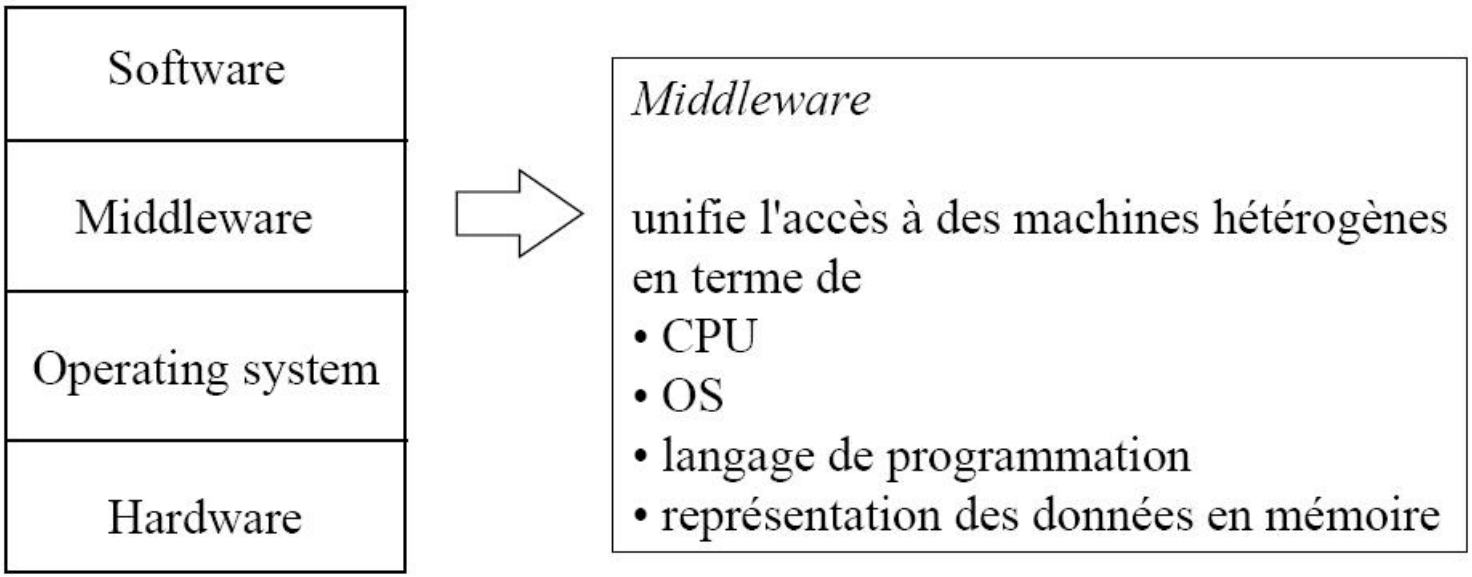
# Caractéristiques communes

- les applications utilisent les logiciels de niveau intermédiare, installés au-dessus des systèmes d'exploitation et des protocoles de communication, qui réalisent :
- cacher la répartition,
- cacher l'hétérogéneité des composants matériels, des systèmes d'exploitation et des protocoles de communication utilisés par les différentes parties d'une application
- fournir des interfaces uniformes, normalisées, et de haut niveau aux équipes de développement et d'intégration, pour faciliter la construction, la réutilisation, le portage et l'interopérabilité des applications
- fournir un ensemble de services communs réalisant des fonctions d'intérêt général, pour éviter la duplication des efforts et faciliter la coopération entre applications.

# L'intergiciel (*middleware*)



# L'intergiciel (*middleware*)



# L'intergiciel (*middleware*)

## L'intergiciel : définition

L'intergiciel (middleware en anglais) est un ensemble de logiciels ou de technologies informatiques qui servent d'intermédiaire entre les applications et le transport des données via le réseau. Ils offrent des services de haut niveau liés aux besoins de communication des applications (temps réel, sécurisation, sérialisation, transaction informatique, etc.)(`fr.wikipedia.org/wiki/Middleware`)

## L'intergiciel : définition

Un middleware permet la communication entre des clients et des serveurs ayant des structures et une implémentation différentes. Il permet l'échange d'informations dans tous les cas et pour toutes les architectures. Enfin, le middleware doit fournir un moyen aux clients de trouver leurs serveurs, aux serveurs de trouver leurs clients et en général de trouver n'importe quel objet atteignable (`cynode.projet-enoch.net/resource/xml/doc/memoire/generatedhtml/memoire_g1107.html`).

# Quelques catégories d'intergiciels

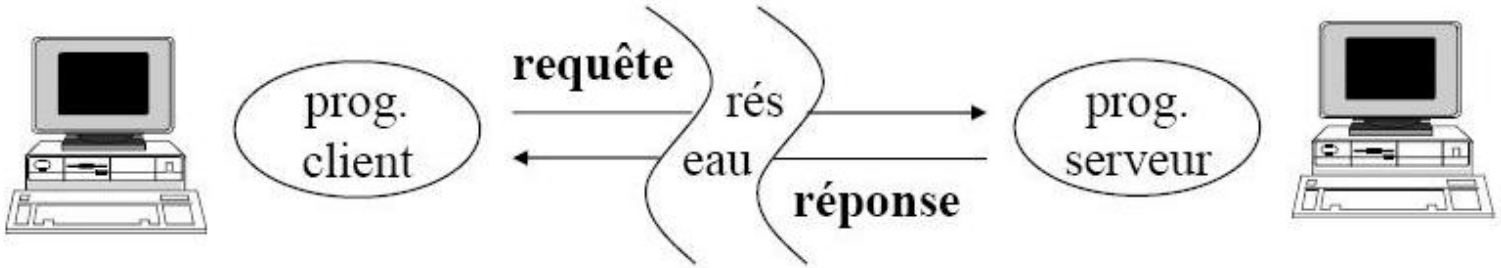
- **orientés objets distribués** : Object Request Broker (ORB), comme Java RMI ou CORBA
- **orientés messages** : Message-Oriented Middleware (MOM), comme MQ Series d'IBM, JMS de Sun, MSMQ de Microsoft
- **transactionnels** : moniteurs transactionnels, comme CICS d'IBM, Tuxedo de BEA Systems, MTS de Microsoft, JTS de Sun.

# Critères de classification

- Nature de entités qui communiquent
  - Objets
  - Composants
  - Agents
- Mode d'accès aux services
  - synchrone (client-serveur)
  - asynchrone (évènement, messages)
  - mixte
- Autres critères
  - entités fixes / mobiles
  - performances / qualité de service garantie ou non
- plusieurs critères de classification

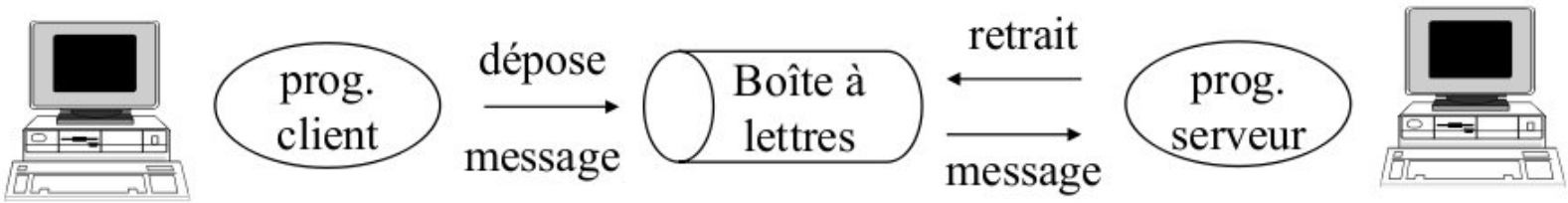
# Modes de communication : Client/Serveur

- intéraction **client/serveur** ou **requête/réponse**
- 1 requête + 1 réponse
- demande d'exécution d'un traitement à distance et réponse
- communication **synchrone**
- appel procédural étendu au cas où l'appelant et l'appelé ne sont pas situés sur la même machine



# Modes de communication : MOM

- Protocoles de niveau applicatif (pas transport) + propriétés (ex transactionnelles)
- Différent de l'envoi de messages sur socket
- communication **asynchrone** (fonctionnement client et serveur découplés)



# Modes de communication : MOM (mode de fonctionnement)

- **Point à point** : une application produit des messages et une application les consomme. Les messages ne sont lus que par un seul consommateur. Une fois qu'un message est lu, il est retiré de la file d'attente.
- **Publish Subscribe** (par abonnement) : les applications consommatrices des messages s'abonnent à un topic (sujet, catégorie de messages). Les messages envoyés à ce topic restent dans la file d'attente jusqu'à ce que toutes les applications abonnées aient lu le message (eg : JMS (*Java Message Service*).

# Mode d'intéraction

- **Synchrone** : le client attend la réponse pour continuer son exécution
- **Asynchrone** : le client n'attend pas de réponse et continue tout de suite. La file d'attente reçoit le message de l'application émettrice et le stocke jusqu'à ce que l'application réceptrice vienne lire le message
- **Semi-synchrone** : le client continue son exécution après l'envoi et récupère le résultat ultérieurement
  - à futur explicite le résultat est stocké dans une boîte à lettres (BAL) le client consulte cette BAL pour récupérer le résultat.
  - à futur implicite le résultat est fourni automatiquement au client par ex. via une variable du programme client

# Mode d'intéraction

- **Synchrones**

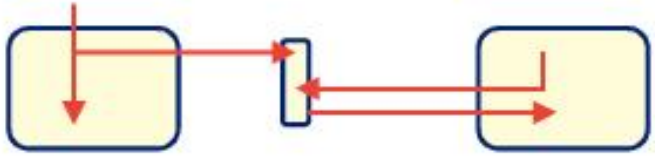
Couplage fort

RMI, CORBA, COM, ...

- **Asynchrones**

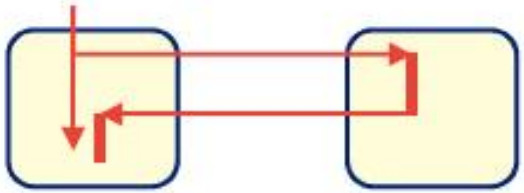
Couplage faible

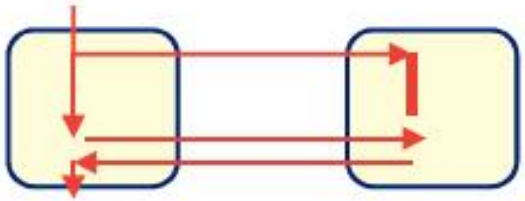
Événements



Queues de messages

- **Semi-synchrones**

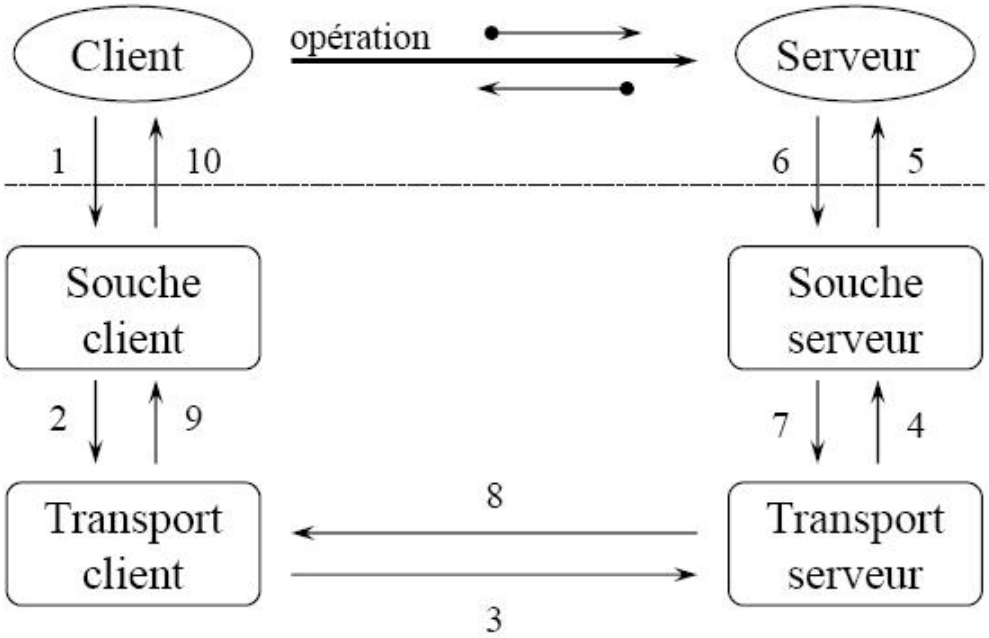




Combinaisons synchrone-asynchrone

# Mise en oeuvre

- Vocabulaire : souche ou talon
- souche client = *stub* ou *proxy*
- souche serveur = *skeleton*




1. Appel local
2. Préparation du message d'appel
2. Envoi
4. «Upcall» vers la souche serveur
5. Décodage du message
6..10 : Chemin inverse

# Evolution des intergiciels : Client/Serveur

- Notion d'application client/serveur 2 tiers ou 3 tiers
- Découpage d'une application
  - présentation
  - traitements
  - données
- Problématique : par rapport à 1 client et 1 (ou +sieurs) serveurs qui assure ces fonctionnalités ?

# Evolution des intergiciels : Client/Serveur 2 tiers

- **Caractéristiques** :
  - 1 gros serveur (mainframes)
  - n terminaux légers connectés au serveur
- **Avantages** :
  - pas de duplication de données (état global observable)
  - gestion simple de la cohérence et de l'intégrité des données
  - maîtrise globale des traitements
- **Inconvénients**
  - modèle trop rigide qui n'assure pas l'évolutivité
  - souvent solutions propriétaires fermés
  - économiquement trop coûteux

# Evolution des intergiciels : Client/Serveur 3 tiers

- **Caractéristiques** :
  - 1 serveur de données et 1 serveur de traitement
  - n PC avec IHM "évoluées"
- **Avantages** :
  - meilleure répartition de charge
  - économiquement moins cher
  - + évolutif
- **Inconvénients**
  - administration + compliquée
  - mise en oeuvre + compliquée

# Evolution des intergiciels : Client/Serveur 3 tiers

- Evolution historique du terme client/serveur 3 tiers
  - tiers 1 (PC), tiers 2 (serveurs départementaux), tiers 3 (serveur central)
  - tiers 1 (client), tiers 2 (BD locale), tiers 3 (BD globale)
- Actuellement
  - tiers 1 (client), tiers 2 (serveur de traitement), tiers 3 (serveur de données)

# Evolution des intergiciels

- envoi de message
- RPC
- RPC objet
- bus logiciel
- serveur d'applications
- service

# Evolution des intergiciels : Envoi de messages

- primitives send & receive
- conception des progs client et serveur en fonction messages attendus et à envoyer
- socket : au-dessus des protocoles TCP & UDP
- primitive bloquante vs non bloquante
- fiabilité
- ordre des messages
- contrôle de flux
- mode connecté vs non connecté

# Evolution des intergiciels : RPC (*Remote Procedure Call*)

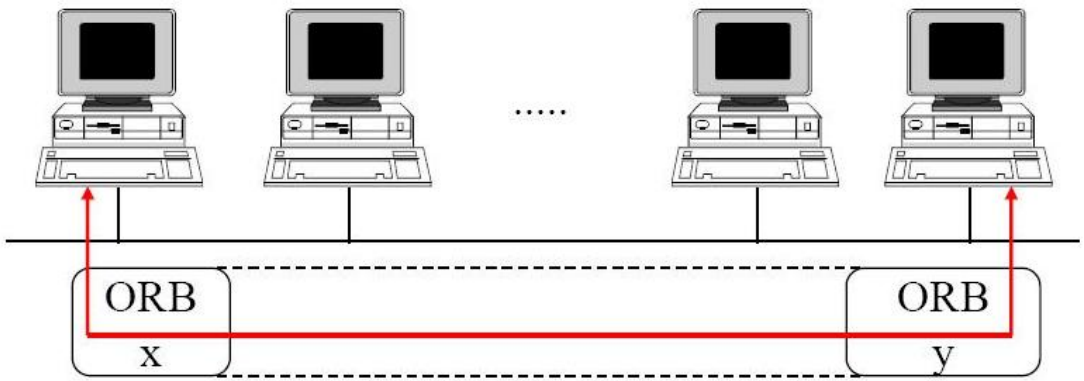
- appel d'une procédure sur une machine distante
- groupement de 2 messages : appel & retour
- adressage : @IP + nom fonction
- définition des signatures des procédures
- compilateur de souches client et serveur

# Evolution des intergiciels : RPC Objet

- mise en commun concepts RPC et prog objet
- appel d'une méthode sur un objet distant
- éventuellement +sieurs objets par machine
- adressage serveur de noms + nom logique

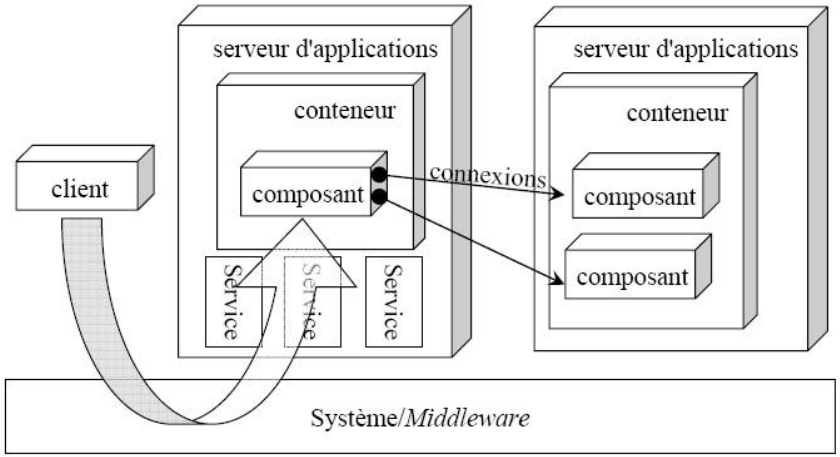
# Evolution des intergiciels : Bus logiciel CORBA

- multi-OS, multi-langage
- métaphore du bus logiciel
- langage IDL
- services : nommage, courtage, transaction, etc.



# Evolution des intergiciels : Composants et serveurs d'application

- CORBA CCM
- Sun EJB
- Microsoft .NET/(D)COM+
- composant
  - monter en abstraction / objet
  - architecture logicielle
- serveur d'applications
  - *framework*
  - héberger, administrer



# Evolution des intergiciels : Service

- SOA : *Software Oriented Architecture*
- SaaS : *Software As A Service*
- Modèle "économique" tout est service
- Principes des SOA
  - *Encapsulation* : encapsulé pour pouvoir être réutilisé avec SOA
  - *Loose coupling* : minimiser les dépenses
  - *Contract* : communication ne se font que via un accord entre services
  - *Abstraction* : masquer la logique du service au monde extérieur
  - *Reusability* : découpage des services pour promouvoir la réutilisabilité
  - *Composability* : services peuvent être composés et coordonnés pour former des services composites
  - *Autonomy* : services contrôlent la logique qu'ils encapsulent
  - *Optimization* : services peuvent être optimisés individuellement
  - *Discoverability* : services sont fait pour être découvert

# Quelques concepts des intergiciels

- objet, composant, service
- interface, contrat
- souche, squelette
- service techniques (aussi appelés non fonctionnels ou extra-fonctionnels)
  - annuaires (registre, moteur de recherche, pages jaunes, Ě)
  - sécurité (contrôle d'accès, cryptage, authentification, Ě)
  - transaction
  - persistence des données,
  - concurrence
  - synchronisation
  - réplication
  - migration

# Panorama des architectures et technologies

- Vision OMG : CORBA (*Common Object Request Broker Architecture*), CCM
- Vision Microsoft : COM (*Component Object Model*), DCOM (*Distributed COM*), OLE (*Object Linking and Embedding*)/ActiveX, COM+, .NET
- Vision Sun : J2EE (*Java 2 Enterprise Edition*), EJB (*Enterprise JavaBeans*)
- ObjectWeb : Fractal
- Les Services Web/Web Services

# Panorama des architectures et technologies : OMG

- OMG : *Object Management Group*
- une association américaine créée en 1989
- but non lucratif
- plus de 850 membres
- standardiser et promouvoir le modèle objet sous toutes ses formes :
  - UML (*Unified Modeling Language*)
  - MOF (*Meta-Object Facility*)
  - CORBA (*Common Object Request Broker Architecture*)
  - IDL (*Interface Definition Language*)
  - MDA (Model Driven Architecture)
  - QVT (Query/View/Transformation).
- `www.omg.org`

# Intergiciels : les avantages et les inconvénients

- **Avantages**
  - la capacité d'abstraction : cacher les détails des mécanismes de bas niveau
  - assurer l'indépendance vis-à-vis des langages et des plates-formes
  - permettre de réutiliser l'expérience et parfois le code
  - faciliter l'évolution des applications
  - réduire le coût et la durée de développement des applications
- **Inconvénients**
  - la perte de performances liée à la traverse de couches supplémentaires de logiciel.
  - prévoir la formation des équipes de développement.

# Limitations des intergiciels

- Passage à large échelle : Web
- Protocoles hétérogènes :
  - IIOP (*Internet Inter-Orb Protocol*), RMI, DCOM
  - Firewall
- Pas d'ouverture des services : notion de moteur de recherche inexistante
- Trop de contraintes sur le client :
  - doit posséder les souches
  - difficulté de construire dynamiquement

# Défis des intergiciels

- **Performance** :
  - les systèmes intergiciels reposent sur des mécanismes d'interception et d'indirection, qui induisent des pertes de performances.
  - L'adaptabilité introduit des indirections supplémentaires,
  - solution partielle : éliminer les coûts inutiles en utilisant l'injection du code de l'intergiciel dans celui des applications
- **Passage à grande échelle** :
  - les applications deviennent de plus en plus interconnectées et interdépendantes
  - le nombre d'objets, d'usagers et d'appareils composants ces applications tend à augmenter.
  - problème de la capacitè de croissance (*scalability*) pour la communication et pour les algorithmes de gestion d'objets et accroît la complexité de l'administration
  - problème de la préservation des différente forme de la qualité de service

# Défis des intergiciels

- **Ubiquité** : (informatique omiprésente) présence d'un nombre croissant et varié de terminaux dans un réeseau d'information.
  - la mobilité et la reconfiguration dynamique seront des traits dominants de ces systèmes,
  - nécessité d'une adaptation permanente des applications.
  - les principes d'architecture applicables aux systèmes d'informatique ubiquitaire restent encore largement à élaborer.
- **Administration** :
  - les applications sont de plus en plus grande, hétérogènes, largement réparties et en évolution permanente
  - problèmes : l'observation cohérente, la sécurité, l'équilibre entre autonomie et interdépendance pour les differents sous-systèmes, la définition et la réalisation des politiques d'allocation de ressources, etc.

# Conclusion

- Présentation générale des intergiciels
- Importance des intergiciels
- Classification des intergiciels
- Mode de communication des intergiciels
- Evolution des intergiciels
- Problématiques et défis des intergiciels