Applications Réparties

Préparé par
M.G. BELKASMI

Principes

- Application répartie = traitements coopérants sur des données réparties.
- Coopération = communications + synchronisation:
  - modèle d'exécution, interface de programmation, outils de développement.
- Distribution
  - des données : données distribuées, traitement centralisé.
  - du contrôle : données centralisées, contrôle distribué.
  - des utilisateurs : données et contrôle centralisé, utilisateurs distribués.

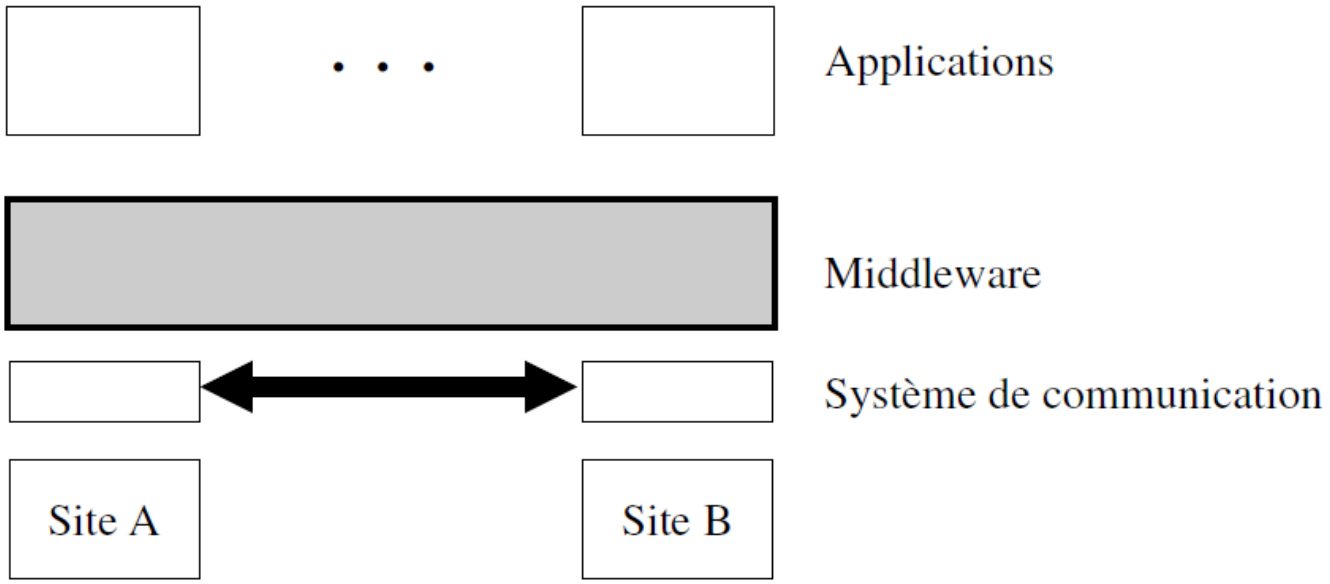
Exemples

- Coordination d'activités :
  - Systèmes à flots de données (« workflow »).
  - Systèmes à agents.
- Communication et partage d'informations :
  - Bibliothèques virtuelles.
- Applications temps réel :
  - Contrôle de procédés industriels
  - Localisation, trafic ...
- Toutes les applications qui nécessitent des utilisateurs ou des données réparties.

Construction

- Conception de l'architecture de l'application.
- Programmation des entités logicielles :
  - Utilisation d'un mécanisme de communication avec un modèle d'exécution (socket, RMI, RPC, CORBA, etc.).
  - Programmation en fonction du modèle d'exécution.
- Configuration des entités de diverses provenances :
  - leur permettre de communiquer, d'échanger des données, d'échanger des informations de contrôle et de se comprendre.
- Prendre en considération l'installation et le déploiement.
- Administration : Surveillance, maintenance et évolution des applications.

Schéma général



Middleware : couche logicielle destinée à

- masquer l'hétérogénéité des machines et des systèmes,
- masquer la répartition des données et des traitements,
- fournir une API de programmation.

Middleware

- Le Middleware conceptualise et réalise les fonctions suivantes :
  - communications entre les applications réparties,
  - échanges de données,
  - facilités de mise en œuvre.
- Il résout les problèmes d'intégration et d'interopérabilité :
  - indépendance entre les applications et le système d'exploitation,
  - portabilité des applications,
  - partage des services distribués.
- Services d'un Middleware :
  - communication,
  - localisation,
  - transactions,
  - sécurité,
  - administration.

Problèmes généraux

- Tolérance aux pannes.
- Passage à l’échelle.
- Nommage et accès aux applications.
- Intégration de l’existant.
- Déploiement des applications.
- Sécurité et authentification.
- Disponibilité de l’application.

Problèmes généraux

Tolérance aux pannes (reliability, fault tolerance)

- Un serveur participant à l'application tombe en panne.
- Un serveur envoie des informations erronées.
- Un serveur n'est plus atteignable (mais pas en panne) puis le redevient.
- Atomicité dans les applications réparties.

Problèmes généraux

Passage à l'échelle (scalability)

- Ce qui marche pour un utilisateur, marchera-t-il pour 10000 ?
- Ce qui marche pour un objet, marchera-t-il pour 1 000 000 ?
- Ce qui marche pour un site, marchera-t-il pour 1000 ?
- Exemple : les applications de gestion de commerce électronique.

Problèmes généraux

Nommage et accès aux applications (naming)

- Comment retrouver les objets distants ?
- Un objet = un identifiant + un état + un comportement.
- Applications non réparties : nommage géré par le langage (référence) ou par l'OS (adressage).
- Applications réparties : nommage explicite, dynamique ?
- Exemple de nommage : DNS, URL, JNDI, LDAP, ...

Problèmes généraux

Intégration de l’existant (legacy)

- Connexion sur toutes les ressources d’une entreprise.
- Interopérabilité des applications.
- Transactions réparties.

Problèmes généraux

Déploiement des applications

- Comment installer tous les composants logiciels sur différents clients et serveurs ?
- Lorsque on change un nom de serveur ou on en ajoute un, on recompile ? on redéploie ? ou on peut configurer automatiquement le redéploiement ?

Problèmes généraux

Sécurité et authentification

- Confidentialité.
- Intégrité : Droits d'accès, Pare-Feu.
- Authentification :
  - Identification des applications partenaires.
  - Messages authentifiés.
- Combien de personnes utilisent l'application, qui sont-ils ?
  - Nécessité de se protéger contre les intrusions.
  - Nécessité de la journalisation des accès des clients

Problèmes généraux

Disponibilité d'une application répartie

- Exemple : un serveur qui fait de la tolérance aux pannes ne peut plus assurer d'autres tâches.
- Permettre des accès simultanés sur un même objet :
  - Sérialiser les objets.
  - Paralléliser les accès.
  - Appliquer différentes politiques.
  - Multi-threading.

Modèles d’exécution

- Modèle Client-Serveur : RPC, RMI, CORBA, Servlet,...
- Modèle de communication par messages :
  - MOM : Message Oriented Middleware.
  - Files de messages.
- Modèle de communication par événements.
- Modèle à base de composants : Bean, EJB.
- Modèle à base d’agents mobiles : Agglet, Voyager.
- Modèles à mémoires « virtuelles » partagées

Le modèle Client-Serveur

Coté serveur :

- Externalisation de services.
- Attente de requêtes en provenances de clients puis exécution des requêtes en séquentiel ou en parallèle.
- Interface : Skeleton
  - reçoit l'appel sous forme de « stream »
  - décapsule les paramètres
  - demande l'exécution
  - renvoi les paramètres (par références) et les résultats

Le modèle Client-Serveur

Coté client :

- Émission de requêtes puis attente de la réponse.
- Initiateur du dialogue.
- Interface : Stub
  - reçoit l'appel en local
  - encapsule les paramètres
  - attends les résultats du serveur
  - décapsule les résultats
  - redonne la main à la fonction appelante

Le modèle Client-Serveur

- Client/Serveur « traditionnel » : RPC
- Client/Serveur « à objets » : RMI, CORBA, DCOM
- Client/Serveur « de données » : Requêtes SQL
- Client/Serveur « WEB » : CGI, Servlet, asp, jsp, php,...

Le modèle de communication par messages

- Module de synchronisation :
  - Communication asynchrone
    - émission non bloquante,
    - réception bloquante (attente jusqu'à réception d'un message).
- Mode de communication :
  - Communication directe entre processus (agents).
  - Communication indirecte (boîtes aux lettres).
- Mode de transmission
- Les environnements :
  - les sockets
  - la programmation parallèle en MPI
  - les Middlewares à messages (MOM).
  - la normalisation JMS (Java Messenging Service).

Le modèle de communication par événements

- Concepts de bases : événements, réactions.
- Principe d'attachement : association dynamique entre un type d'évènement et une réaction.
- Communication anonyme : indépendance entre l'émetteur et les « consommateurs » d'un évènement.
- Deux modes :
  - PULL : les clients viennent prendre régulièrement leurs messages.
  - PUSH : une méthode prédéfinie est attachée à chaque type de message et elle est appelée automatiquement à chaque occurrence de l'évènement.

Le modèle à base de composants

- Un composant :
  - module logiciel autonome et réutilisable.
  - Brique de base pour construire une application.
- Caractéristiques d'un composant :
  - des entrées/sorties déclarées pour permettre les connexions entre plusieurs composants.
  - des propriétés déclarées permettant de configurer le composant.

Le modèle à base de composants

Composant Java Beans :

- Bean = classe + conventions d'écriture :
  - Entrées : les méthodes publiques.
  - Propriétés : variables d'instances et accesseurs/modificateurs.
  - Sorties : les événements émis.
- On connecte et on configure des instances.
- Les instances sont crées par un BeanContainer (modèle par composition).

Le modèle à base de composants

Composant Java Beans :

- Le BeanContainer doit être capable d'interroger les instances pour :
  - découvrir les propriétés qui peuvent être manipulées,
  - découvrir les événements qui doivent être émis,
- Conventions d'écriture :
  - constructeur de la classe sans paramètres,
  - méthodes commençant par set ou get pour manipuler les propriétés,
  - méthodes commençant par add et remove pour manipuler les événements.
- Un bean peut accéder à un objet distant (RMI, CORBA, etc.), se connecter à une base de données.

Le modèle à base de composants

Les Enterprise Java Beans

- Un environnement pour la répartition des objets répartis :
  - un serveur qui gère tous les problèmes de répartitions.
  - le développement d'objets métiers conforme au modèle de l'environnement.
- Les serveurs d'EJB ou serveurs d'applications : ensemble de services permettant la bonne exécution des composants :
  - les services de base :
    - accès aux composants,
    - optimisation de l'accès aux ressources locales et distantes,
    - gestion transactionnelle,
    - répartition de charge
  - l'administration, l'exploitation, la sécurité ;
  - les passerelles vers l'existant ;
  - la persistance

Les modèles par agents mobiles

- Code mobile = programme se déplaçant d'un site à un autre sur le réseau.
- Avantage de la mobilité :
  - efficacité, privilégie les interactions locales,
  - moins de communications distantes effectuées pour les échanges de messages,
  - amener le code aux données plutôt que le contraire,
  - permet à des clients d'étendre les fonctions d'un serveur pour des besoins spécifiques.
- Les agents mobiles :
  - entités logicielles permettant de construire des applications naturellement distribuées,
  - utilisation de ressources allouées,
  - autonomie et déplacement sur différents sites d'un réseau.

Modèles à mémoires « virtuelles » partagées

- Objectifs :
  - Replacer le programmeur dans les conditions d'un système centralisé :
    - utiliser un espace mémoire commun pour les communications,
    - synchronisation des applications par variables partagées.
  - Avantages :
    - transparence de la distribution pour le développeur,
    - efficacité du développement car utilisation des paradigmes usuels de la programmation concurrente.
- Problématique :
  - Utilisation des outils de développement existant.
  - Mise en œuvre efficace d'une mémoire partagée distribuée.

Modèles à mémoires « virtuelles » partagées

Approches utilisées :

- Modèles à espace de tuples :
  - bases de données relationnelles partagées,
  - modèle de programmation : dépôt, retrait et consultation d'objets.
- Modèles à objets répartis partagées :
  - espace d'objets répartis partagés,
  - interface de programmation : langage à objets « étendus »,
  - plusieurs modes de réalisation : objets répliqués ou objets à image unique.