NSY107 — Intégration des systèmes client/serveur

Chapître 1 - Introduction aux systèmes répartis

Claude Duvallet

Université du Havre
UFR Sciences et Techniques
25 rue Philippe Lebon - BP 540
76058 LE HAVRE CEDEX
Claude.Duvallet@gmail.com
http://litis.univ-lehavre.fr/∼duvallet/

Plan de la présentation

- Introduction aux applications réparties.
- Les appels de procédure distante (RPC).
- Propriétés d'ordre, algorithmique répartie.
- Objets répartis et composants (CORBA, RMI, EJB) :
  - Les applications réparties avec RMI.
  - Les applications réparties avec CORBA.
  - L'approche composant avec les EJB.
- Les annuaires LDAP (peut-être).
- Mémoire virtuelle, fichiers.
- Tolérance aux pannes.

WEBlographie

J'ai souvent eu recours aux supports disponibles sur le WEB et notamment à ceux des personnes suivantes :

- **Pascal Molli**. Maître de conférences à l'Université Henry Poincaré.
- **Roland Balter**. ScalAgent Distributed Technologies.
- **Cyrille Bertelle**. Maître de conférences à l'Université du Havre.
- **Damien Olivier**. Maître de conférences à l'Université du Havre.
- **Éric Leclercq**. Université de Bourgogne.
- **Samia Bouzefrane**. Maître de conférences au CNAM de Paris.
- **Nicolas Delestre**. Maître de conférences àl'INSA de Rouen.
- **Gérard Florin**. Professeur au CNAM de Paris.

Plan de la présentation

Les applications réparties : 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.
- Distribution du contrôle :
  - données centralisées, contrôle distribué.
- Distribution des utilisateurs :
  - données et contrôle centralisé, utilisateurs distribués.

$\Rightarrow$ une combinaison de tout ça...

Exemples d'applications réparties

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

Construction d'applications réparties

- 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.
  - leur permettre d'échanger des informations de contrôle.
  - leur permettre de se comprendre.
- Prendre en considération l'installation et le déploiement.
- Administration :
  - Surveillance, maintenance et évolution des applications.

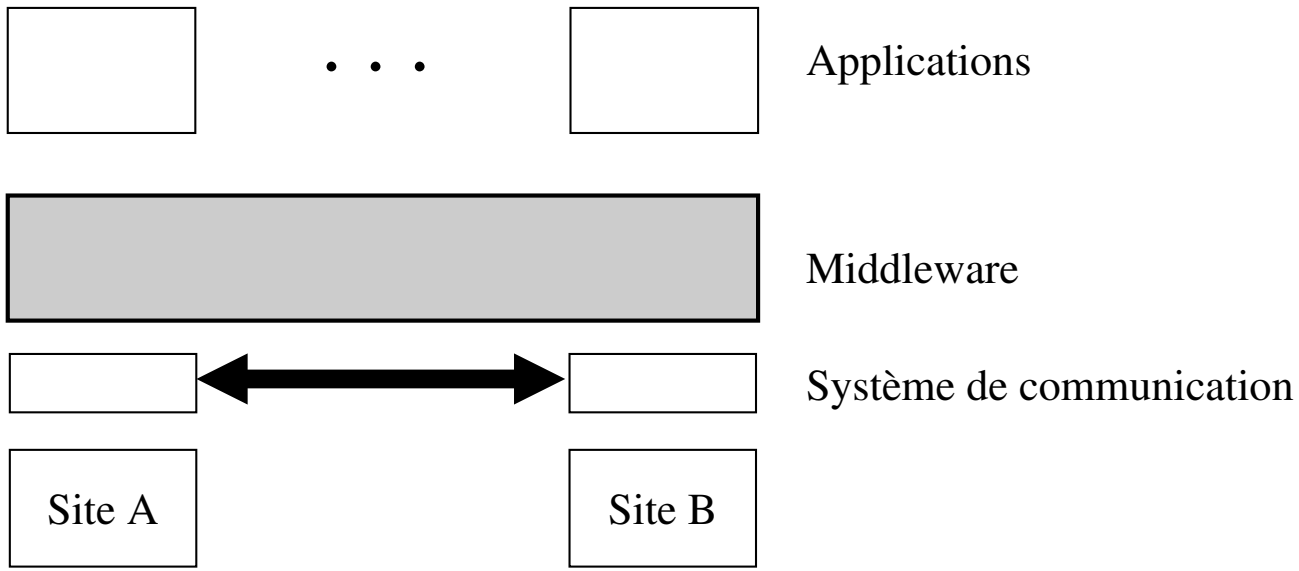
Conception d'un jeu de Morpion (1/2)

- Conception suivant une architecture Client/Serveur.
- Les échanges entre les clients et le serveur :
  - le client passe des commandes au serveur.
  - le serveur notifie au client les changements d'états aux clients (joueur X a joué en (x,y)).
- Choix d'un modèle d'exécution :
  - appel de procédure distante (RMI).
  - implémentation de méthodes d'accès au sein du serveur.
  - implémentation d'un client sous forme d'une applet.

Conception d'un jeu de Morpion (2/2)

- Configuration :
  - mise en place des politiques de sécurité : fichiers de sécurité et serveur d'authentification.
  - configuration du serveur Web pour recevoir l'applet.
- Déploiement (applet client) :
  - copie des fichiers ".class".
  - démarrage de rmiregistry.
  - démarrage du serveur HTTP.
  - démarrage du serveur Morpion.
  - accès distant à la page Morpion.

Schéma général des applications réparties



- 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.

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.

Tolérance aux pannes

- En anglais : 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.

Passage à l'échelle

- En anglais : scalability.
- Ce qui marche pour un utilisateur, marchera-t-il pour 10 000 ?
- 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.

Nommage et accès aux applications

- En anglais : 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, ...

Intégration de l'existant

- En anglais : legacy.
- Connexion sur toutes les ressources d'une entreprise.
- Interopérabilité des applications.
- Transactions réparties.

Déploiement des applications

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

Sécurité et authentification

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

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.

Les différents 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 :
  - Modèles à espace de tuples.
  - Modèles à objets dupliqués.

Le modèle Client-Serveur (1/2)

- 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
- 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 (2/2)

- 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 :
  - Peut-être typé
- Les environnements :
  - les sockets sous Unix.
  - la programmation parallèle en MPI, PVM.
  - 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 et PUSH
  - 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

- Définition d'un composant :
  - module logiciel autonome et réutilisable.
  - composable visuellement et donc dynamiquement pour construire une application.
- Beans, EJB, CORBA Component.
- Caractéristiques d'un composant :
  - des entrées/sorties déclarées pour permettre les connexions entre plusieurs composants.
  - des propriétés déclarées permettant de configurer le composant.
- 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).

Les composants Beans

- Le BeanContainer doit être capable d'interroger les instances pour :
  - découvrir les propriétés qui peuvent être manipulées,
  - comment elles doivent l'être,
  - découvrir les événements qui doivent être émis,
  - utilisation de l'introspection (au moyen de la classe `class`).
- 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 n'est pas forcément un composant visuel : il remplit une fonction (entrées/sorties/propriétés).
- Un bean peut accéder à un objet distant (RMI, CORBA, etc.), se connecter à une base de données.

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 : il s'agit d'un cadre d'exécution pour des composants obéissant à un modèle (COM ou EJB), c'est-à-dire un ensemble de services permettant la bonne exécution des composants :
  - les services de base ;
  - l'administration, l'exploitation, la sécurité ;
  - les passerelles vers l'existant ;
  - la persistances.
- Quatre services de base :
  - accès aux composants,
  - optimisation de l'accès aux ressources locales et distantes,
  - gestion transactionnelle,
  - répartition de charge.

Les modèles par agents mobiles

- Code mobile = programme se déplaçant d'un site à un autre sur le réseau.
- Exemple : les applets = programme exécutable inclut dans une page HTML et qui s'exécute sur le site qui télécharge la page.
- 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 (1/2)

- 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 (2/2)

- Approches utilisées :
  - Modèles à espace de tuples :
    - bases de données relationnelles partagées,
    - modèle de programmation à la « linda » : 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.

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.