Intergiciels

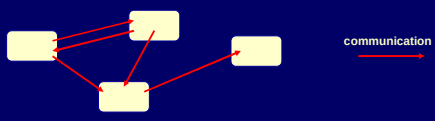
Dr. Philippe Merle
ADAM / INRIA Futurs – GOAL / LIFL - USTL
http://www.lifl.fr/~merle

Patrons et canevas pour l'exécution répartie

- Objets répartis : ***Broker***
- Désignation et liaison : ***Export-Bind*** ;
- Coordination : ***Observer***, ***Publish-Subscribe***

Objets répartis

- Schéma de base
  - Application = ensemble d'objets répartis sur un réseau, communiquant entre eux (1 objet intégralement sur un site)



Autres modèles (non considérés ici)
- Objets fragmentés
- Objets dupliqués
- Objets mobiles
- ...

Intergiciel pour objets répartis

- Exemples
  - *Java Remote Method Invocation* (RMI) : appel d'objets distants en Java - Sun
  - *Common Object Request Broker Architecture* (CORBA) : support pour l'exécution d'objets répartis hétérogènes - OMG
  - DCOM, COM+ : *Distributed Common Object Model* - Microsoft
- Schéma commun : ORB (*Object Request Broker*)
  - Modèle de base : client-serveur
  - Identification et localisation des objets
  - Liaison entre objets clients et serveurs
  - Exécution des appels de méthode à distance
  - Gestion du cycle de vie des objets (création, activation, …)
  - Services divers (sécurité, transactions, etc.)

Problèmes de l'exécution répartie

- Schéma d'interaction
  - Communication
  - Synchronisation
- Désignation et localisation des objets
  - Identification et référence
- Liaison
  - Établissement de la chaîne d'accès
- Cycle de vie
  - Création, conservation, destruction des objets
- Mise en œuvre (réalisation, services)

Notre objectif
- élaborer des patrons pour les aspects ci-dessus
- proposer un canevas pour la structure d'un ORB

Schémas d'interaction

- Synchrones

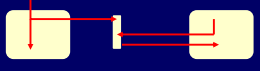
Couplage fort

RMI, CORBA, COM, …

- Asynchrones

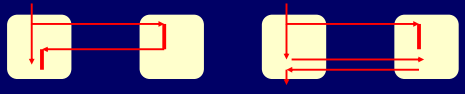
Couplage faible

Événements



Files de messages

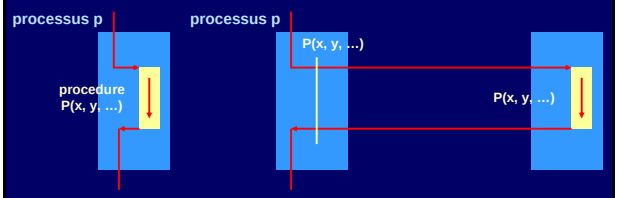
- Semi-synchrones



Combinaisons synchrone-asynchrone

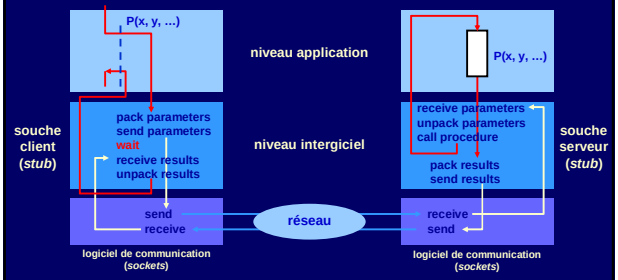
Bref rappel sur RPC (1/2)

- L'appel de procédure à distance (RPC), un outil pour construire des applications client-serveur



L'effet de l'appel doit être identique dans les deux situations. Cela est impossible à réaliser en présence de défaillances

Bref rappel sur RPC (2/2)

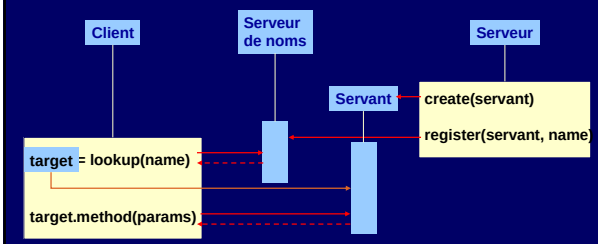


- Réalisation de l'appel de procédure à distance

Du RPC aux objets répartis…

- Pourquoi les objets répartis
  - Avantages d'un modèle à objets pour la programmation
  - L'encapsulation se prête bien à la répartition (objet = unité naturelle de répartition)
  - Réutilisation de l'existant (par *wrappers*)
- Différences avec RPC
  - Encapsulation de données
  - Création dynamique d'objets
    - Donc liaison dynamique
  - Intégration de services
    - Persistance, duplication, transactions, etc.

Les étapes d'un appel d'objet réparti
(vues du programmeur)



Connaissances requises :
- le client et le serveur connaissent le serveur de noms
- le client et le serveur s'accordent sur le nom *name*
- le client connaît l'interface du servant

Les étapes d'un appel réparti
(vues du système)

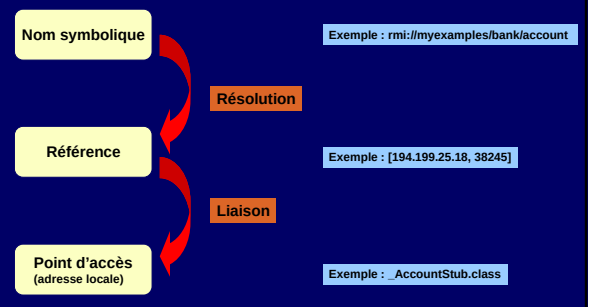
En partant de la fin…
Pour réaliser l'appel réparti, il faut avoir établi une chaîne d'accès entre le client et le servant

L'opération de liaison (*binding*) est la création de cette chaîne d'accès (également appelée "objet de liaison")

Désignation et liaison

- **Désignation**
  - Associer des noms à des objets
  - Retrouver un objet à partir de son nom
- **Liaison**
  - Créer une chaîne d'accès à un objet (à partir d'un nom)
- **Spécificité de la liaison répartie**
  - En centralisé (très schématiquement)
    - 2 sortes de noms : nom symboliques, adresses
    - Liaison = recherche de l'adresse (souvent avec indirection)
  - En réparti
    - "Adresse" = référence (exemple : [adresse IP, n°de p ort])
    - Mais référence ≠ chaîne d'accès !

Étapes de la liaison répartie

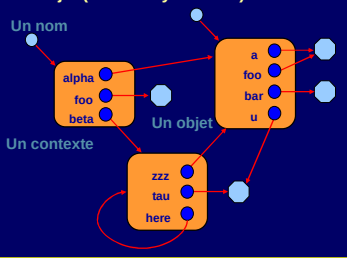


Rappel rapide sur les noms

- Deux sortes de noms
  - Identificateurs : distinguer un objet des autres
  - Références : localiser un objet (en vue d'y accéder)
- Noms contextuels

Un espace plat est inutilisable
recherche inefficace
pas de localité

Contexte = partie de l'univers
Graphe de contextes
(souvent hiérarchique : arbre, etc.)



Résolution et liaison des noms

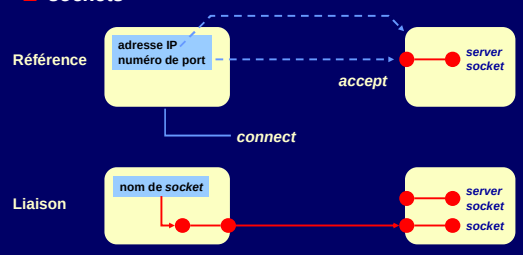
- Résoudre un nom (dans un contexte)
  - À partir du nom, trouver l'objet
  - Processus récursif

    target = context.resolve(name) [ou name.resolve()]

    3 issues possibles pour *target*
    - Une valeur typée : c'est l'objet
    - Une référence (ex : adresse) => l'objet est localisé
    - Un autre nom (dans un autre contexte) => on rappelle *resolve*
- Lier un nom
  - À partir du nom, construire une chaîne d'accès à l'objet
  - Rappel : en réparti, résolution ≠ liaison !
    - Exemple : une référence (adresse IP, n°de porte) ne suffit pas pour accéder à l'objet

Liaison en réparti : exemple

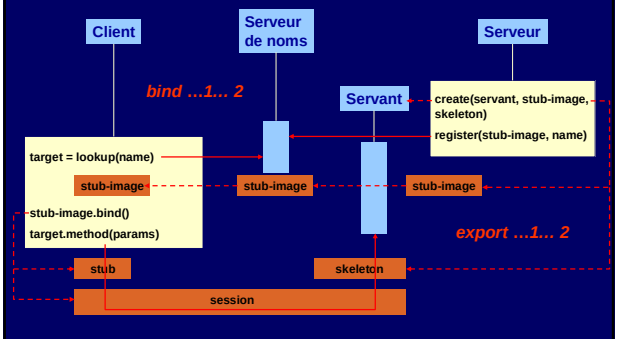
- *sockets*



Un patron pour la liaison répartie

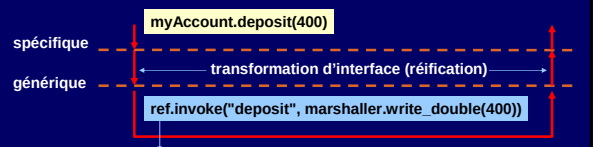
- La liaison est établie en 2 phases
- Côté serveur (*export*)
  - "publication" de l'objet à lier (identification)
  - préparation de certains éléments de la liaison
- Côté client (*bind*)
  - établissement de la liaison par création et assemblage des constituants de l'objet de liaison
- Exemple
  - La connexion par sockets peut être décrite en ces termes
    - *accept = export*
    - *connect = bind*

Liaison dans un ORB (1)



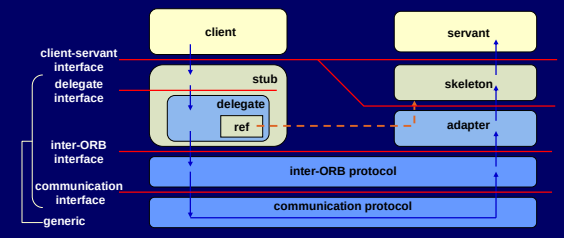
Structure d'un appel distant (1)

- L'interface "de bout en bout" est spécifique
  - Interface d'un objet défini par l'application
  - Exprimée dans un IDL, pour faciliter la portabilité
- Les interfaces intermédiaires (internes à l'ORB) ont intérêt à être génériques
  - Pour faciliter les échanges de composants d'ORB
  - Pour faciliter l'interopérabilité entre ORBs



ref = référence à l'objet servant *myAccount*

Structure d'un ORB



Transformation d'interface
côté client, dans la souche (vers le "délégué", ou souche générique)
côté serveur, dans un adaptateur

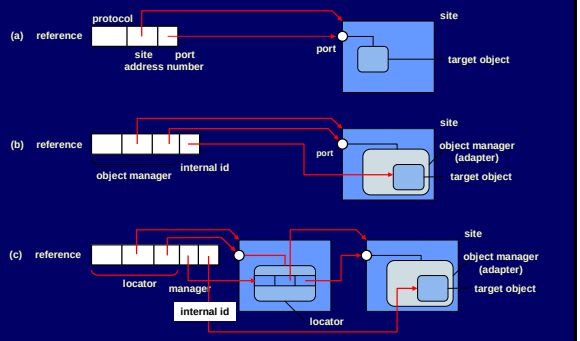
Fonctions de l'adaptateur

- Gestion des objets servants
  - Référentiel des implémentations dans CORBA
- Gestion des références d'objets
  - Créer une référence pour un objet (à la création de l'objet)
  - Trouver un objet, connaissant sa référence
- Gestion des activités côté serveur
  - Activer un objet (lui associer un *thread* pour son exécution)
- Exemple : le POA (*Portable Object Adapter*) de CORBA (OMG)
  - Permet d'isoler les politiques de gestion d'objets
    - Persistance, politique d'activation, format de références, etc.

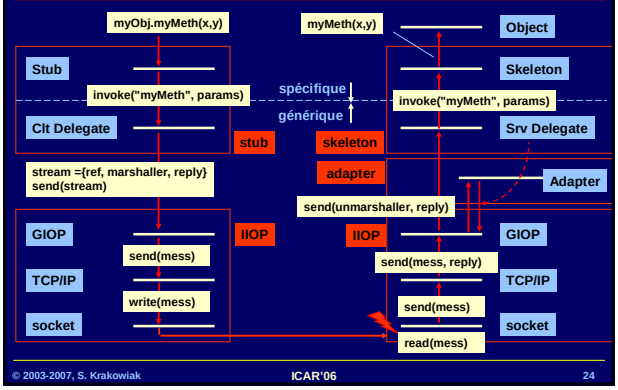
Une interface générique pour les ORB

- GIOP : *General Inter-ORB Protocol*
  - Définit une interface et un protocole générique pour l'appel d'objets distants sur une couche de transport
    - Représentation commune de données (*Common Data Representation*, CDR)
    - Format standard de référence d'objet (*Interoperable Object Reference*, IOR)
    - Format des messages
    - Contraintes sur la couche de transport
- IIOP : *Internet Inter-ORB Protocol*
  - La réalisation "standard" de GIOP
    - GIOP sur TCP/IP

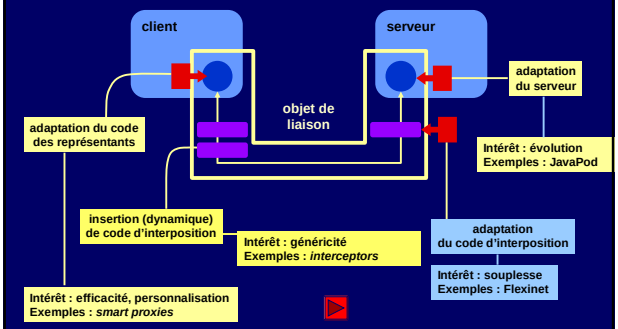
Format d'une référence



Structure d'un appel distant (2)



Adaptation de la liaison



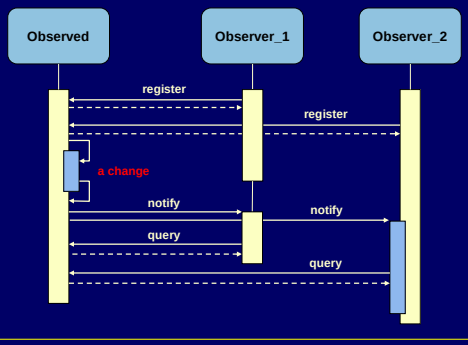
Coordination

- Définitions
  - Méthodes et outils permettant à un ensemble d'entités de coopérer à une tâche commune
  - Modèle de coordination, définit :
    - les entités coopérantes (processus, activités, "agents", …)
    - le support (médium) de coordination : véhicule de l'interaction
    - les règles de coordination : primitives, patrons d'interaction
- Domaine d'application
  - Couplage faible (évolution indépendante des entités)
  - Structure très dynamique (les entités peuvent rejoindre/quitter le système à tout instant)
  - Hétérogénéité (système, environnement, administration)
  - Condition requise : capacité de croissance

Observer, patron de base pour la coordination

- Contexte
  - Des objets "observés", dont l'état (visible) évolue au cours du temps
  - Des objets "observateurs"
- Problème
  - Permettre aux observateurs d'être informés de l'évolution des objets observés
  - Propriétés souhaitables
    - Requérir un effort minimal de la part des observateurs
    - Garantir l'indépendance mutuelle des observateurs
    - Permettre l'évolution dynamique (arrivée-départ des observateurs et observés)
  - Contraintes
    - Passage à l'échelle
- Solution
  - Les observateurs enregistrent leur intérêt auprès des observés
  - Les observés notifient aux observateurs enregistrés les événements pertinents, de manière asynchrone

Observer



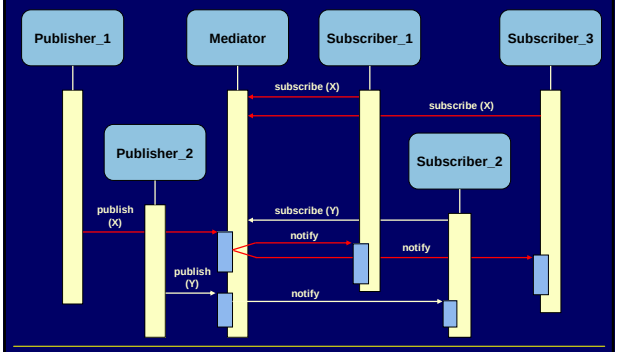
Autres patrons pour la coordination

- Les limitations de *Observer*…
  - Forte charge sur les objets observés (gèrent les observateurs et répondent aux consultations)
  - Manque de sélectivité du schéma notification-consultation (l'observateur reçoit toutes les notifications de changement)
- … et deux réponses
  - *Publish-Subscribe*
    - Deux "rôles" : abonné (*subscriber*) et émetteur (*publisher*)
    - Une entité d'intermédiation (médiateur)
    - Abonnement par sujet ou par contenu
  - Espace partagé
    - Le médium de coordination est un ensemble de tuples
    - Opérations : déposer, consulter avec filtrage (destructivement ou non)

Publish/Subscribe

- Contexte
  - Ensemble d'entités devant se coordonner par émission d'événemens et réaction à ces événements (autre formulation de *Observer*)
- Problème
  - Propriétés souhaitables
    - Comme *Observer* (indépendance, évolution dynamique)
    - Pas de rôle prédéfini
    - Sélectivité sur la nature des événements
  - Contraintes
    - Passage à l'échelle
    - Propriétés diverses : tolérance aux fautes, transactions, persistance, ordre
- Solution
  - Deux "rôles" : abonné (*subscriber*) et émetteur (*publisher*)
  - Une entité d'intermédiation (médiateur)
  - Abonnement par sujet (statique) ou par contenu (dynamique)

Publish/Subscribe



Coordination : réalisations

- *Message-Oriented Middleware* (MoM)
  - Regroupe *Publish-Subscribe* et *Message Queues*
  - Abonnement par sujet : nombreuses réalisations industrielles
    - Tibco, Websphere, ... ScalAgent (JORAM) -> cf. atelier
    - Un standard pour l'interface : JMS (*Java Messaging System*)
  - Abonnement par contenu : prototypes de recherche
    - Gryphon (IBM Research)
    - Siena (Univ. Colorado)
- Espace partagé
  - Modèle : Linda (espace de tuples), projection dans divers langages
  - Réalisation : Jini (Sun)
    - Utilisation : découverte de ressources