Environnement de distribution et architecture répartie

Ce support est très largement inspiré du livre et du cours de S. Krakowiak

©S. Krakowiak

Objectifs

- Typologies des applications et systèmes répartis (distribués)
- Présentation des normes utilisées dans la construction d'applications réparties
- Méthodes de conception d'applications réparties

Motivations

- La répartition est un état de fait pour un nombre important d'applications
  - Développement des réseaux (Internet, réseaux sans fil)
  - Intégration d'applications existantes initialement séparées
  - Pénétration de l'informatique dans des domaines nouveaux d'application
    - Intégration d'objets du monde réel (informatique omniprésente (ubiquitous computing))
    - Intégration entre informatique et télécommunications
- Le parallélisme est la réponse aux besoins croissants des applications
  - Puissance de traitement
  - Gestion de grandes masses de données
  - Intégration et mise en commun de ressources

Caractéristiques des systèmes répartis

- Définition d'un système réparti
  - Ensemble composé d'éléments reliés par un système de communication ; les éléments ont des fonctions de traitement (processeurs), de stockage (mémoire), de relation avec le monde extérieur (capteurs, actionneurs)
  - Les différents éléments du système ne fonctionnent pas indépendament mais collaborent à une ou plusieurs tâches communes. Conséquence : une partie au moins de l'état global du système est partagée entre plusieurs éléments (sinon, on aurait un fonctionnement indépendant)
  - De manière plus précise : toute expression de la spécification du système fait intervenir plusieurs éléments (exemple : préserver un invariant global, mettre des interfaces en correspondance, etc.)

Caractéristiques des systèmes répartis

- Propriétés souhaitées
  - Le système doit pouvoir fonctionner(au moins de façon dégradée) même en cas de défaillance de certains de ses éléments
  - Le système doit pouvoir résister à des perturbations du système de communication (perte de messages, déconnexion temporaire, performances dégradées)
  - Le système doit pouvoir résister à des attaques contre sa sécurité (violation de la confidentialité, de l'intégrité, usage indu de ressources, déni de service)
  - Le système doit pouvoir facilement s'adapter pour réagir à des changements d'environnement ou de conditions d'utilisation
  - Le système doit préserver ses performances lorsque sa taille croît (nombre d'éléments, nombre d'utilisateurs, étendue géographique)

Caractéristiques des systèmes répartis

- Difficultés
  - Propriété d'asynchronisme du système de communication (pas de borne supérieure stricte pour le temps de transmission d'un message
    - Conséquence : difficulté pour détecter les défaillances
  - Dynamisme (la composition du système change en permanence)
    - Conséquences : difficulté pour définir un état global
    - Difficulté pour administrer le système
  - Grande taille (nombre de composants, d'utilisateurs, dispersion géographique)
    - Conséquence : la capacité de croissance (scalability) est une propriété importante, mais difficile à réaliser
- Malgré ces difficultés, des grands systèmes répartis existent et sont largement utilisés
  - le DNS (Domain Name System)
  - le World Wide Web

Applications réparties

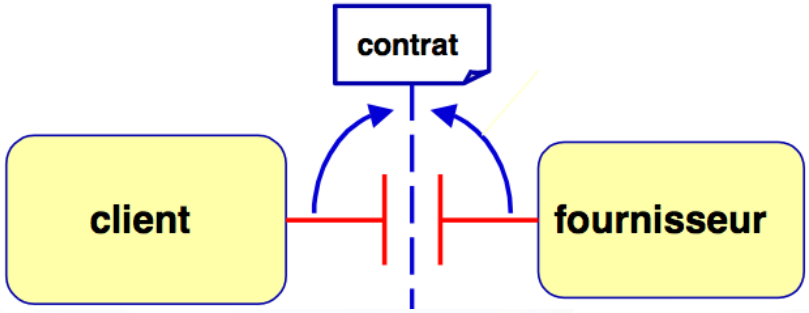
- Distinction entre "système" et "application"
  - Système : gestion des ressources communes et de l'infrastructure, lié de manière étroite au matériel sous-jacent
    - Système d'exploitation : gestion de chaque élément
    - Système de communication : échange d'information entre les éléments
    - Caractéristiques communes : cachent la complexité du matériel et des communications, fournissent des services communs de plus haut niveau d'abstraction
  - Application : réponse à un problème spécifique, fourniture de services à ses utilisateurs (qui peuvent être d'autres applications). Utilise les services généraux fournis par le système
  - La distinction n'est pas toujours évidente, car certaines applications peuvent directement travailler à bas niveau (au contact du matériel). Exemple : systèmes embarqués, réseaux de capteurs

Services et interfaces

- Définition
  - Un système est un ensemble de composants (au sens non technique du terme) qui interagissent
  - Un service est "un comportement défini par contrat, qui peut être implémenté et fourni par un composant pour être utilisé par un autre composant, sur la base exclusive du contrat [1]
- Mise en oeuvre
  - Un service est accessible via une ou plusieurs interfaces
  - Une interface décrit l'interaction entre client et fournisseur du service
    - Point de vue opérationnel : définition des opérations et structures de données qui concourent à la réalisation du service
    - Point de vue contractuel : définition du contrat entre client et fournisseur

[1] Bieber and Carpenter, Introduction to Service-Oriented Programming, http ://www.openwings.org

Définitions d'interfaces



- La fourniture d'un service met en jeu deux interfaces
  - Interface requise (côté client)
  - Interface fournie (côté fournisseur)
- Le contrat spécifie la compatibilité (conformité) entre ces deux interfaces
  - Au delà de l'interface, chaque partie est une "boîte noire" pour l'autre (principe d'encapsulation)
  - Conséquence : client ou fournisseur peuvent être remplacés du moment que le composant remplaçant respecte le contrat (est conforme)

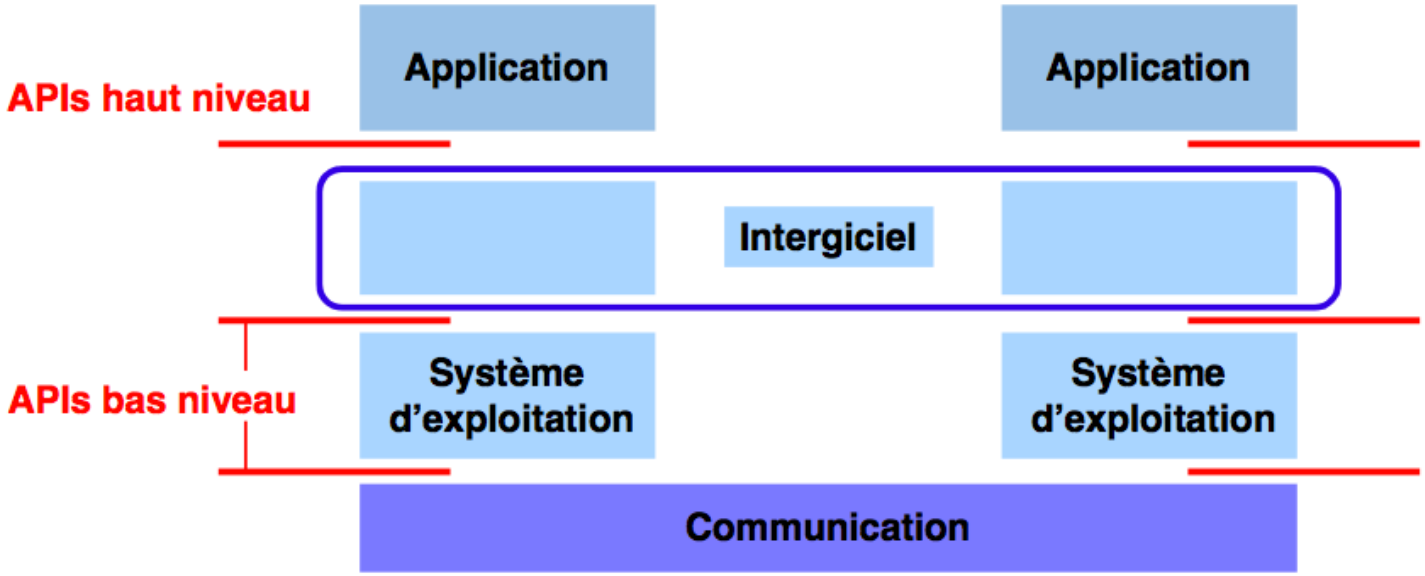
Définitions d'interfaces

- Partie "opérationnelle"
  - Interface Definition Language (IDL) : pas de standard, mais s'appuie sur un langage existant
    - IDL CORBA sur C++
    - Java et C# définissent leur propre IDL
- Partie "contractuelle"
  - Plusieurs niveaux de contrats
    - Sur la forme : spécification de types $\longrightarrow$ conformité syntaxique
    - Sur le comportement (1 méthode) : assertions $\longrightarrow$ conformité sémantique
    - Sur les interactions entre méthodes : synchronisation
    - Sur les aspects non fonctionnels (performances, etc.) : contrats de QoS

L'intergiciel (middleware)

- Motivations
  - Dans un système réparti, même l'interface fournie par les systèmes d'exploitation et de communication est encore trop complexe pour être utilisée directement par les applications.
    - Hétérogénéité
    - Complexité des mécanismes (bas niveau)
    - Nécessité de gérer (et de masquer, au moins partiellement) la répartition
- Solution
  - Introduire une couche de logiciel intermédiaire (répartie) entre les niveaux bas (systèmes et communication) et le niveau haut (applications) : c'est l'intergiciel
  - L'intergiciel joue un rôle analogue à celui d'un "super-système d'exploitation" pour un système réparti

Place et interfaces de l'intergiciel



- L'intergiciel est la couche "du milieu" (Middleware)
  - L'intergiciel est une notion récente (années 90), mais la chose existait avant le mot

Fonctions de l'intergiciel

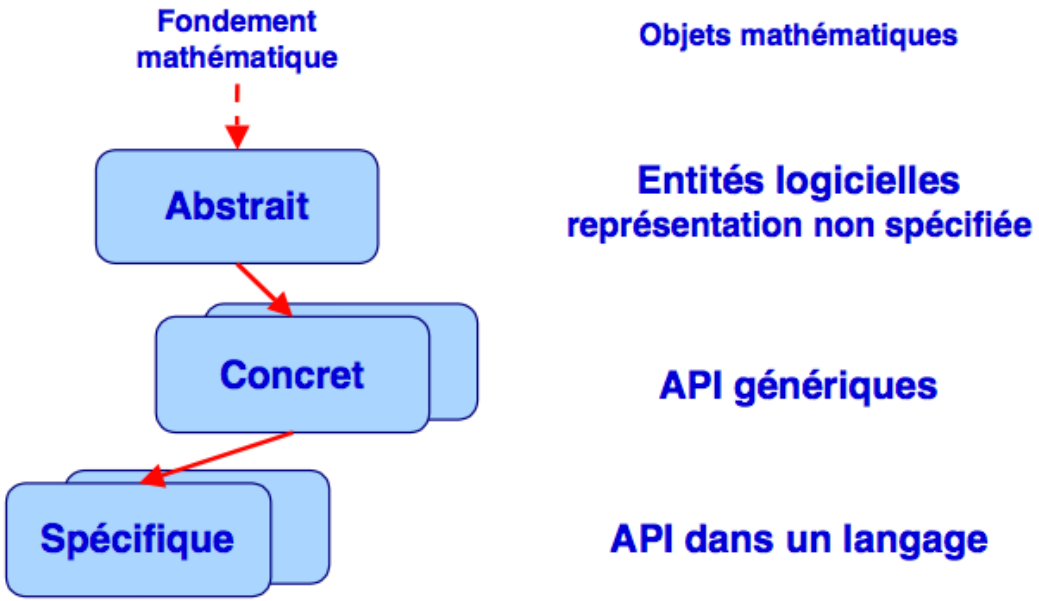
- L'intergiciel a quatre fonctions principales
  - Fournir une interface ou API (Applications Programming Interface) de haut niveau aux applications
  - Masquer l'hétérogénéité des systèmes matériels et logiciels sous-jacents
  - Rendre la répartition aussi invisible ("transparente") que possible
  - Fournir des services répartis d'usage courant
- L'intergiciel vise à faciliter la programmation répartie
  - Développement, évolution, réutilisation des applications
  - Portabilité des applications entre plates-formes
  - Interoperabilité d'applications hétérogènes

Classes d'intergiciel

- Objets répartis
  - Java RMI, CORBA, DCOM, .NET
- Composants répartis
  - Java Beans, Enterprise Java Beans, CCM
- Message-Oriented Middleware (MOM)
  - Message Queues, Publish-Subscribe
- Coordination
- Intégration d'applications
  - Web Services
- Accès aux données, persistance ' Support d'applications mobiles

Modèles d'architecture logicielle

- Définition
  - Une description d'un aspect (point de vue, fonction) de l'architecture
- Usage
  - Compréhension, explication, prévision, preuve, guide pour la réalisation
- Une hiérarchie de modèles



Modèles d'architecture logicielle

- Plusieurs critères de classification
  - Selon nature du flot de contrôle
    - Synchrone (client-serveur)
    - Asynchrone (messages, événements)
    - Mixte
  - Selon unité d'organisation
    - Objets répartis
    - Composants
  - Structure statique ou dynamique
    - Mobilité des éléments
    - Reconfiguration
- La majorité des modèles sont concrets ou spécifiques (état de l'art)

Principes de conception

- Principe directeur : séparation des préoccupations
  - Isoler les aspects indépendants (ou faiblement corrélés) et les traiter séparément
  - Examiner un problème à la fois
  - Eliminer les interférences
  - Permettre aux aspects d'évoluer indépendamment
- Mise en oeuvre
  - Encapsulation : séparer interface et réalisation (contrat commun)
  - Abstraction : décomposition en niveaux, cacher les détails non pertinents à un niveau donné
  - Séparationentrepolitiquesetmécanismes
    - Ne pas réimplémenter les mécanismes quand on change de politique
    - Ne pas "sur-spécifier" les mécanismes
  - Isolation et expression indépendante des aspects extra-fonctionnels (hors interface)

Patrons de conception

- Définition (dépasse le cadre de la conception de logiciel)
  - Ensemble de règles (définitions d'éléments , principes de composition, règles d'usage) permettant de répondre à une classe de besoins spécifiques dans un environnement donné.
- Propriétés
  - Un patron est élaboré à partir de l'expérience acquise au cours de la résolution d'une classe de problèmes apparentés" ; il capture des éléments de solution communs
  - Un patron définit des principes de conception, non des implémentations spécifiques de ces principes.
  - Un patron fournit une aide à la documentation, par ex. en définissant une terminologie, voire une description formelle ("langage de patrons")
- Références
  - E. Gamma, R. Helm, R. Johnson, J. Vlissides. Design Patterns - Elements of Reusable Object-Oriented Software, Addison-Wesley, 1995
  - F. Buschmann, R. Meunier, H. Rohnert, P. Sommerlad, M. Stal. Pattern-Oriented Software Architecture - vol. 1, Wiley 1996
  - D. Schmidt, M. Stal, H. Rohnert, F. Buschmann. Pattern-Oriented Software Architecture"- vol. 2, Wiley, 2000

Patrons de conception

- Définition d'un patron
  - Contexte : Situation qui donne lieu à un problème de conception ; doit être aussi générique que possible (mais éviter l'excès de généralité)
  - Problème : spécifications, propriétés souhaitées pour la solution ; contraintes de l'environnement
  - Solution :
    - Aspects statiques : composants, relations entre composants ; peut être décrit par diagrammes de classe ou de collaboration
    - Aspects dynamiques : comportement à l'exécution, cycle de vie (création, terminaison, etc.) ; peut être décrite par des diagrammes de séquence ou d'état
- Catégories de patrons
  - Conception : petite échelle, structures usuelles récurrentes dans un contexte particulier
  - Architecture : grande échelle, organisation structurelle, définit des sous-systèmes et leurs relations mutuelles
  - Idiomatiques : constructions propres à un langage

Source : F. Buschmann, R. Meunier, H. Rohnert, P. Sommerlad, M. Stal. Pattern-Oriented Software Architecture - vol. 1, Wiley 1996

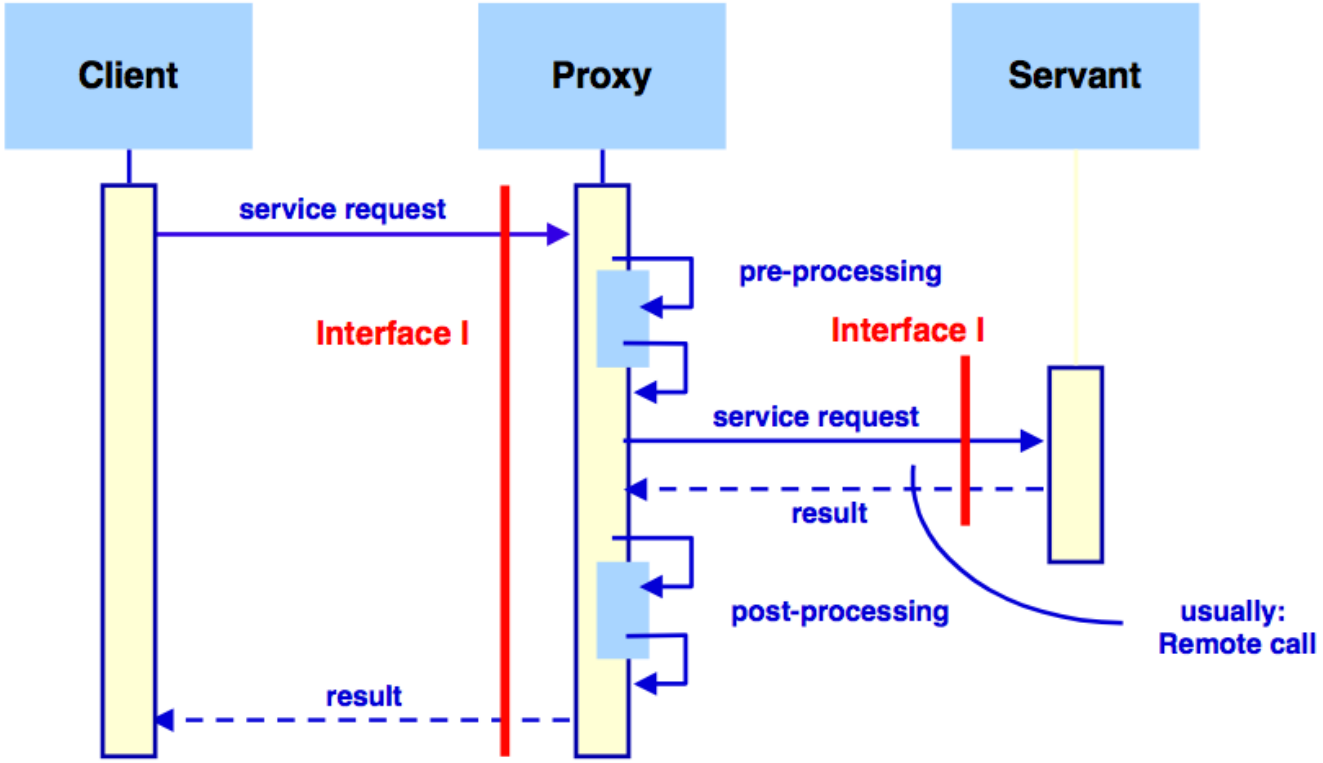
Quelques patrons de base

- Ces patrons sont d'un usage courant dans la construction d'intergiciel
  - Proxy
    - Patron de conception : représentant pour accès à distance
  - Factory
    - Patron de conception : création d'objet
  - Wrapper (Adapter)
    - Patron de conception : transformation d'interface
  - Interceptor
    - Patron d'architecture : adaptation de service
  - Observer
    - Patron de base pour l'asynchronisme

Proxy (Mandataire)

- Contexte
  - Applications constituées d'un ensemble d'objets répartis ; un client accède à des services fournis par un objet pouvant être distant (le "servant")
- Problème
  - Définir un mécanisme d'accès qui évite au client
    - Le codage "en dur" de l'emplacement du servant dans son code
    - Une connaissance détaillée des protocoles de communication
  - Propriétés souhaitables
    - Accès efficace et sûr
    - Programmation simple pour le client ; idéalement, pas de différence entre accès local et distant
  - Contraintes
    - Environnement réparti (pas d'espace unique d'adressage)
- Solutions
  - Utiliser un représentant local du servant sur le site client (isole le client du servant et du système de communication)
  - Garder la même interface pour le représentant et le servant
  - Définir une structure uniforme de représentant pour faciliter sa génération automatique

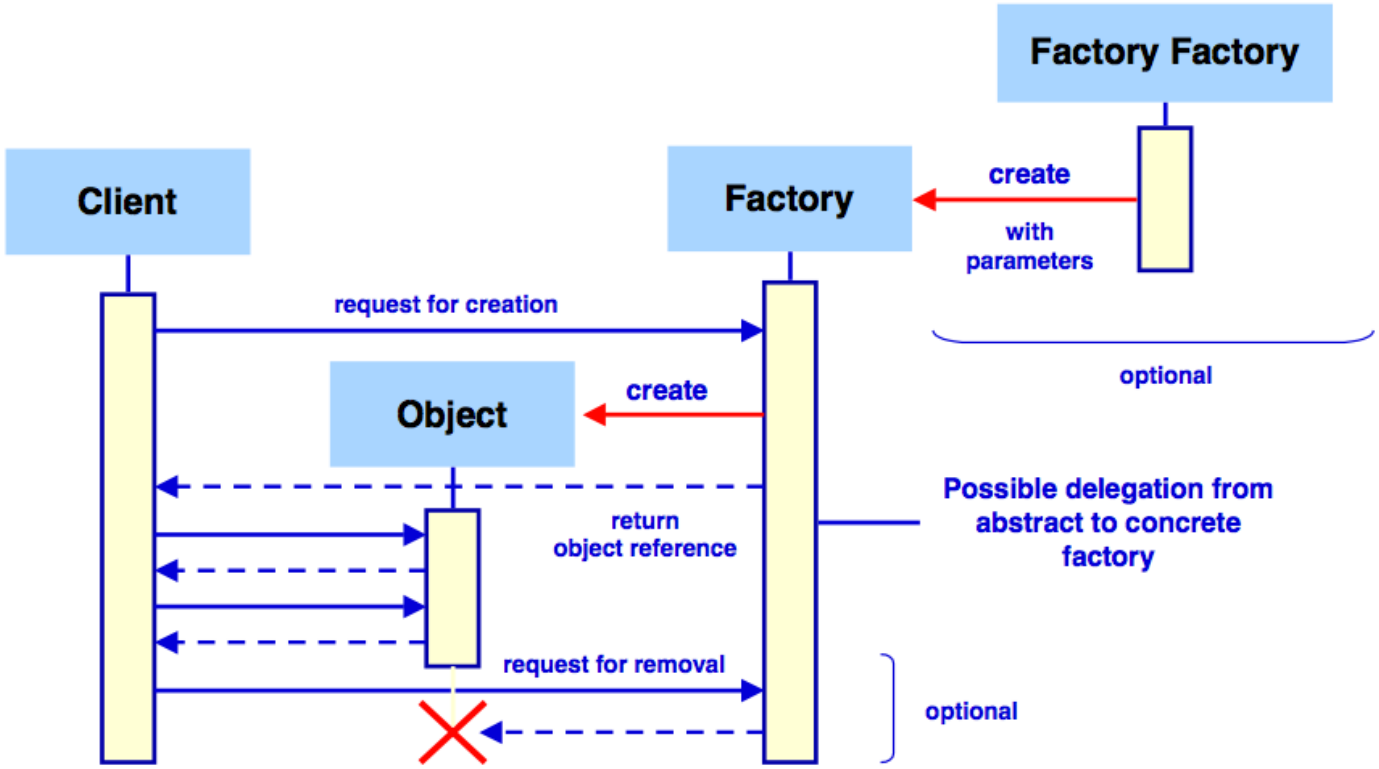
Usage de Proxy



Factory (Fabrique)

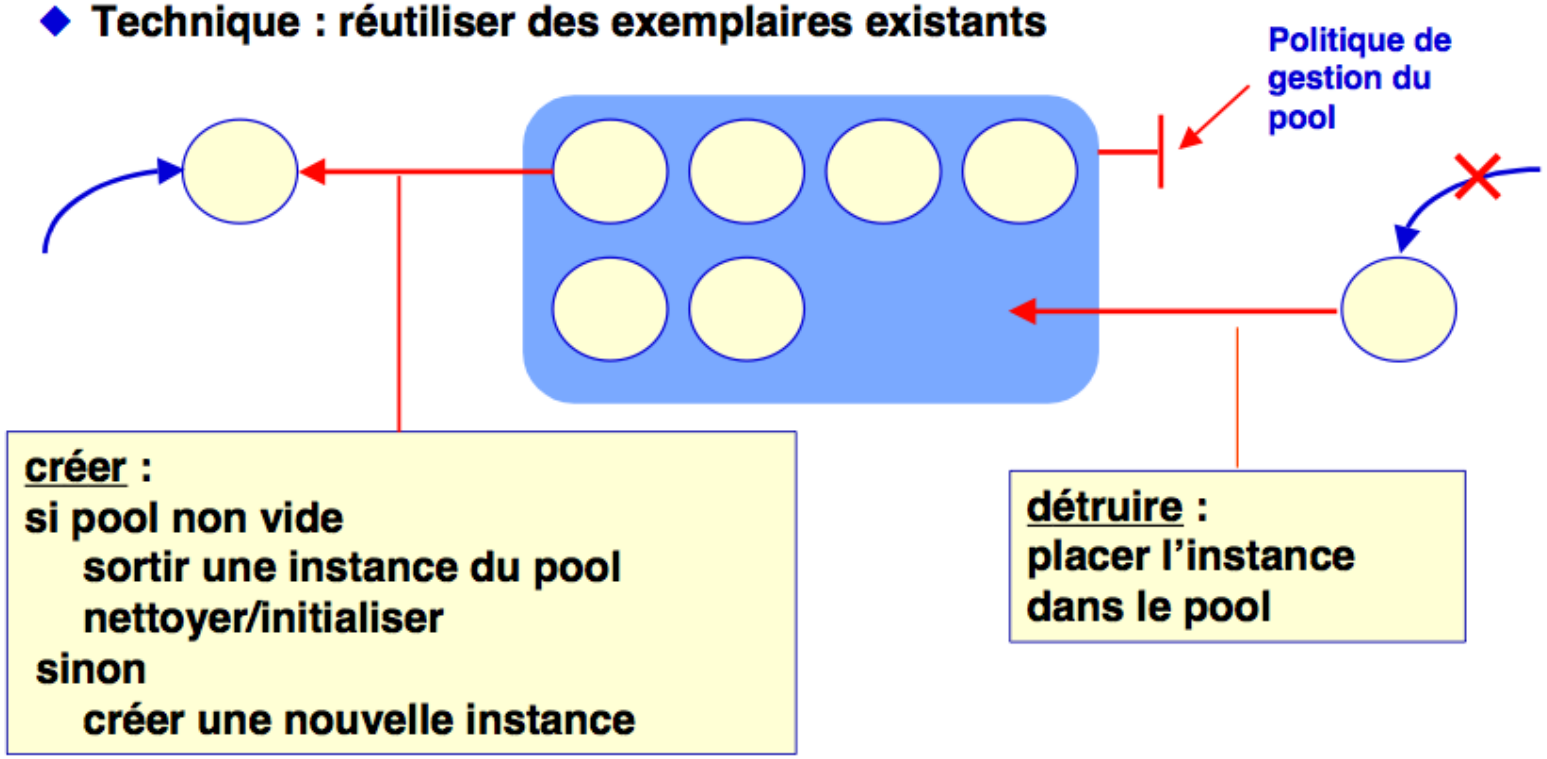
- Contexte
  - Application = ensemble d'objets en environnement réparti
- Problème
  - Créer dynamiquement des instances multiples d'une classe d'objets
  - Propriétés souhaitables
    - Les instances doivent être paramétrables
    - L'évolution doit être facile (pas de décisions "en dur")
  - Contraintes
    - Environnement réparti (pas d'espace d'adressage unique)
- Solutions
  - Abstract Factory : définit une interface et une organisation génériques pour la création d'objets ; la création effective est déléguée à des fabriques concrètes qui implémentent les méthodes de création
  - AbstractFactory peut être implémentée par Factory Methods (méthode de création redéfinie dans une sous-classe)
  - Pour plus de souplesse, on peut utiliser FactoryFactory (le mécanisme de création lui-même est paramétré)

Usage de Factory



Un complément à Factory : Pool

- Idée : réduire le coût de la gestion de ressources
  - **Technique : réutiliser des exemplaires existants**



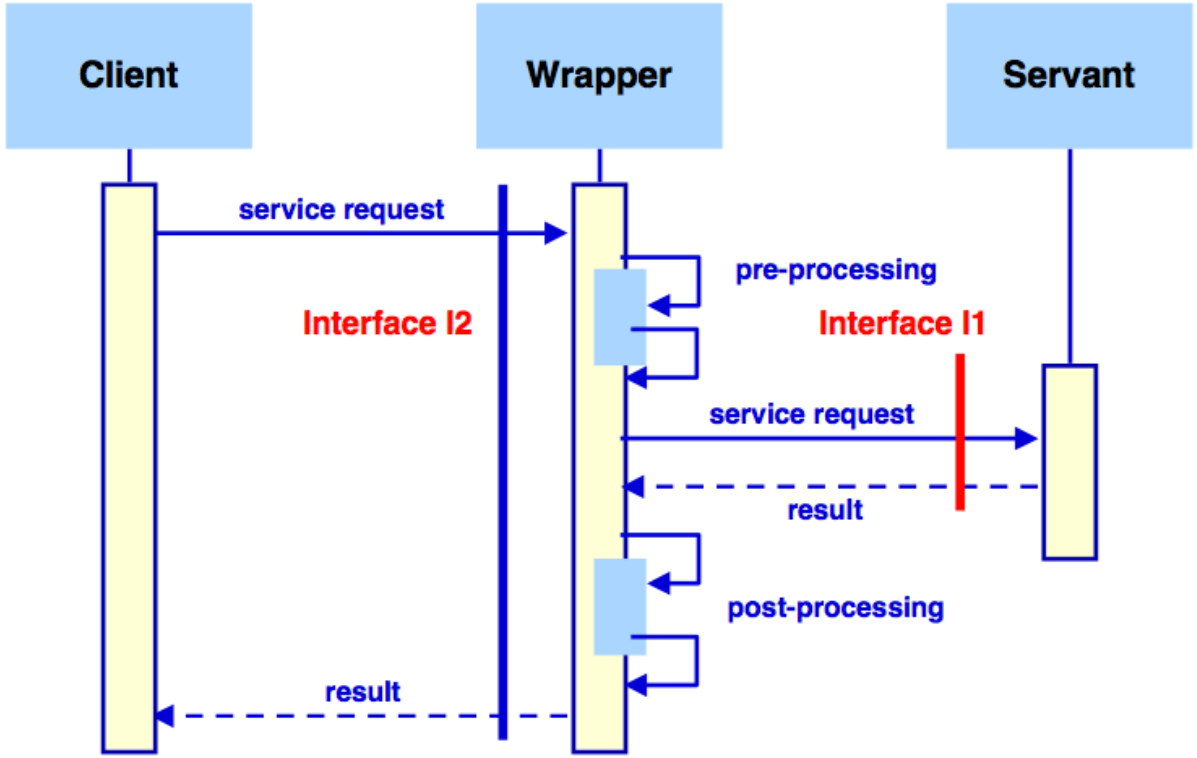
Utilisation de Pool

- Gestion de la mémoire
  - Pool de zones (plusieurs tailles possibles)
  - Evite le coût du ramasse-miettes
  - Evite les copies inutiles (chaînage de zones)
- Gestion des activités
  - Pool de threads
  - Evite le coût de la création
- Gestion de la communication
  - Pool de connexions
- Gestion des composants
  - Voir plus loin (réalisation des conteneurs)

Wrapper (ou Adapter)

- Contexte
  - Des clients demandent des services ; des servants fournissent des services ; les services sont définis par des interfaces
- Problème
  - Réutiliser un servant existant en modifiant son interface et/ou certaines de ses fonctions pour satisfaire les besoins d'un client (ou d'une classe de clients)
  - Propriétés souhaitables : doit être efficace ; doit être adaptable car les besoins peuvent changer de façon imprévisible ; doit être réutilisable (générique)
- Solutions
  - Le Wrapper isole le servant en interceptant les appels de méthodes vers l'interface de celui-ci. Chaque appel est précédé par un prologue et suivi par un épilogue dans le Wrapper
  - Les paramètres et résultats peuvent être convertis

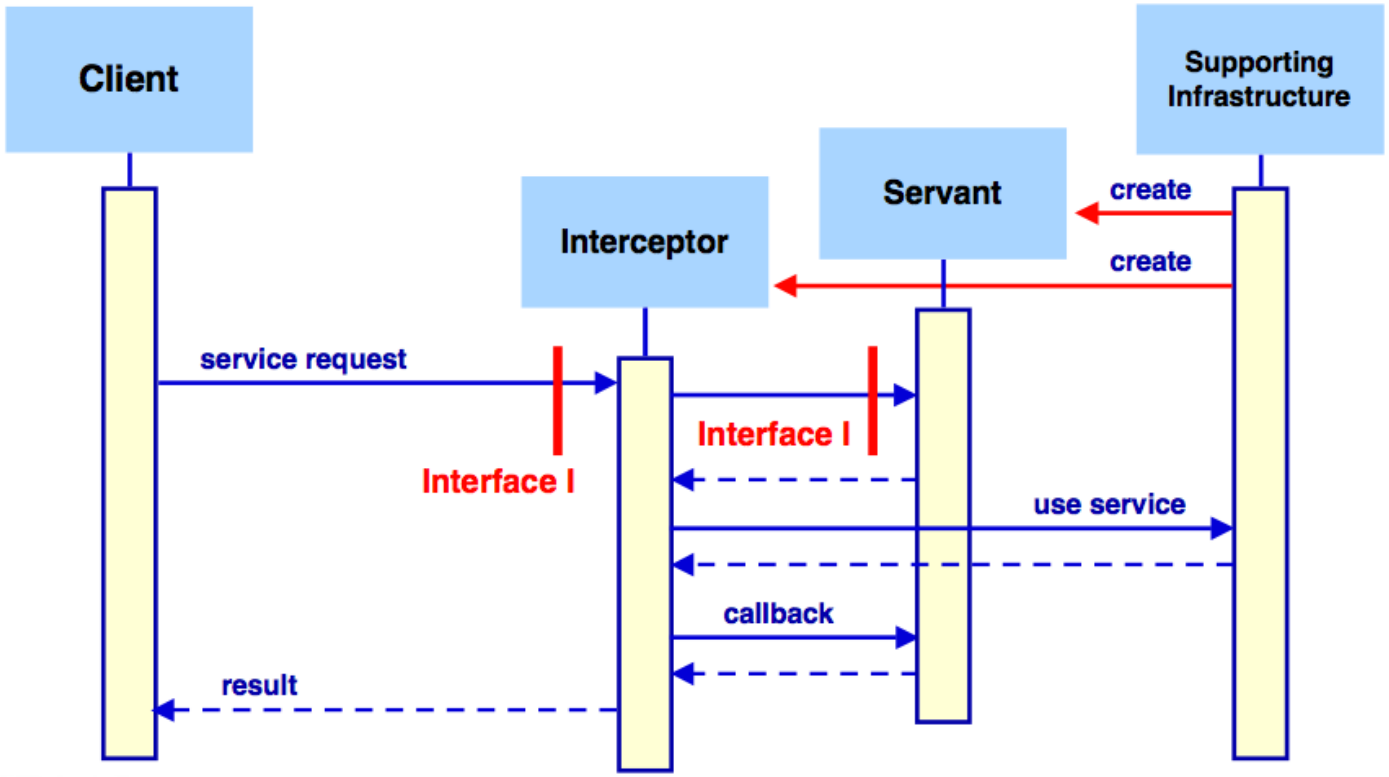
Usage de Wrapper



Interceptor (Intercepteur)

- Contexte
  - Fourniture de services (cadre général)
    - Client-serveur, pair à pair, hiérarchique
    - Uni- ou bi-directionnel, synchrone ou asynchrone
- Problème
  - Transformer le service (ajouter de nouvelles fonctions), par différents moyens
    - Interposer une nouvelle couche de traitement (cf. Wrapper)
    - Changer (conditionnellement) la destination de l'appel " Contraintes
    - Les programmes cleient et serveur ne doivent pas être modifiés
    - Les services peuvent être ajoutés ou supprimés dynamiquement
- Solutions
  - Créer des objets d'interposition (statiquement ou dynamiquement). Ces objets
    - interceptent les appels (et/ou les retours) et insèrent des traitements spécifiques, éventuellement fondés sur une analyse du contenu
    - peuvent rediriger l'appel vers une cible différente
    - peuvent utiliser des appels en retour

Usage d'Interceptor



Comparaison des patrons de base

- Wrapper vs. Proxy
  - Wrapper et Proxy ont une structure similaire
    - Proxy préserve l'interface ; Wrapper transforme l'interface
    - Proxy utilise (pas toujours) l'accès à distance ; Wrapper est en général local
- Wrapper vs. Interceptor
  - Wrapper et Interceptor ont une fonction similaire
    - Wrapper transforme l'interface
    - Interceptor transforme la fonction (peut même complètement détourner l'appel de la cible initiale)
- Proxy vs. Interceptor
  - Proxy est une forme simplifiée d'Interceptor
    - on peut rajouter un intercepteur à un proxy (smart proxy)

Canevas logiciels (Frameworks)

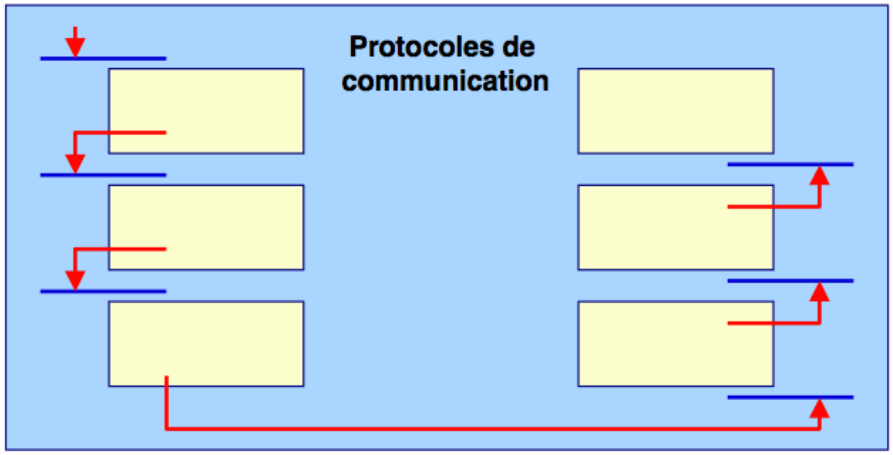
- Définition
  - Un canevas est un "squelette" de programme qui peut être réutilisé (et adapté) pour une famille d'applications
  - Il met en oeuvre un modèle (pas toujours explicite)
  - Dans les langages à objets : un canevas comprend
    - Un ensemble de classes (souvent abstraites) devant être adaptées (par ex. par surcharge) à des environnements et contraintes spécifiques
    - Un ensemble de règles d'usage pour ces classes
- Patrons et canevas
  - Ce sont deux techniques de réutilisation
  - Les patrons réutilisent un schéma de conception ; les canevas réutilisent du code
  - Un canevas implémente en général plusieurs patrons

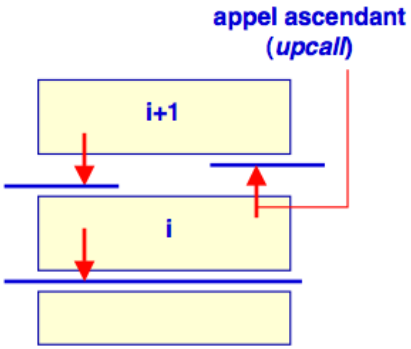
Schémas de décomposition

- Objectifs
  - Faciliter la construction
    - La structure reflète la démarche de conception
    - Les interfaces et les dépendances sont mises en évidence " Faciliter l'évolution
    - Principe d'encapsulation
    - Echange standard
- Exemples
  - Structures multi-niveaux
    - Décomposition "verticale" ou "horizontale" " Canevas pour insertion de composants

Décomposition en couches

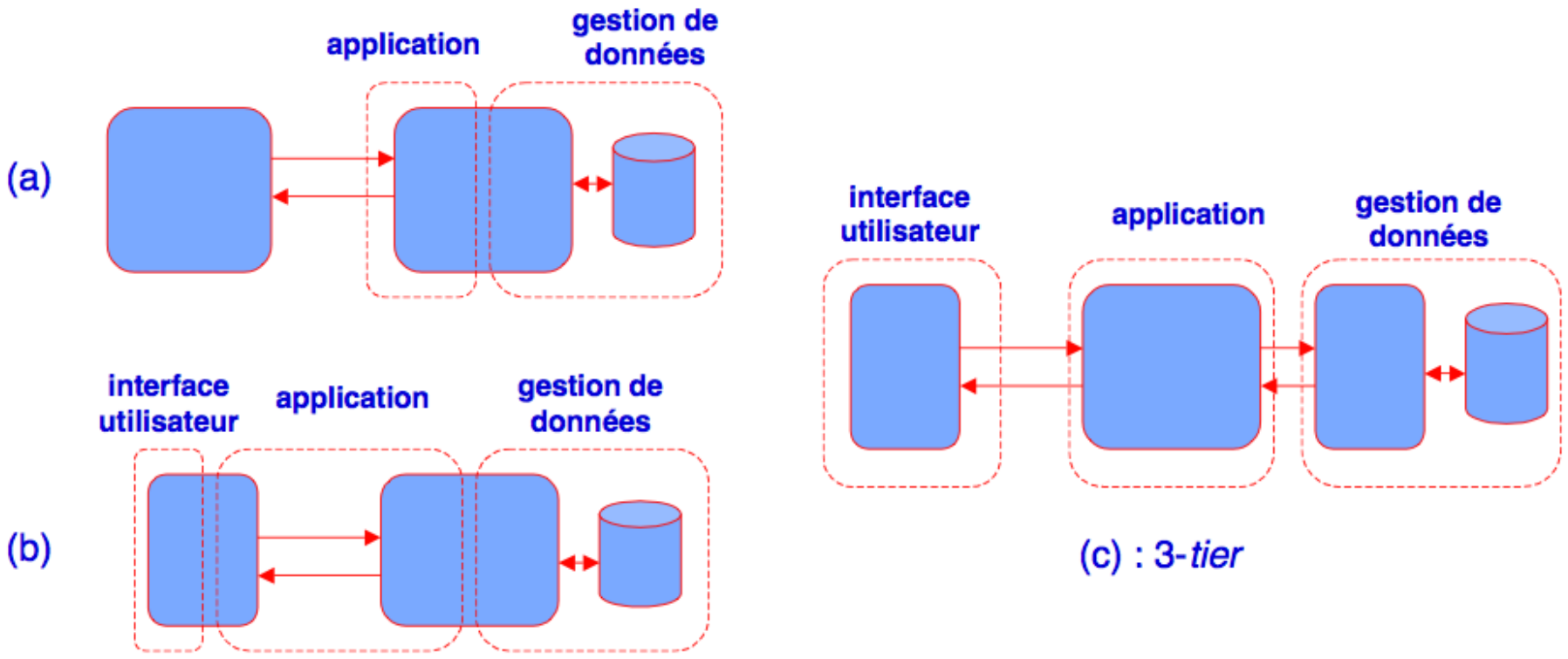
- Hiérarchie de "machines abstraites"
  - La réalisation des niveaux ¡ i est invisible au niveau i
  - Exemple : machines virtuelles (OS multiples, JVM, etc.)





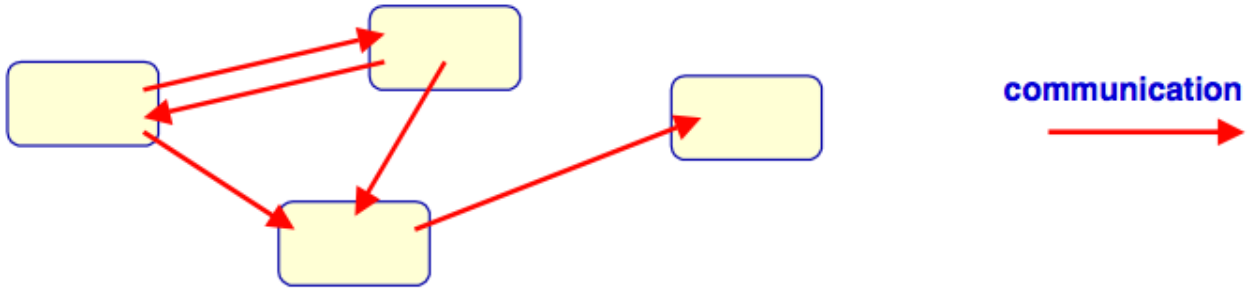
Décomposition "horizontale"

- Exemple : évolution du schéma client-serveur



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

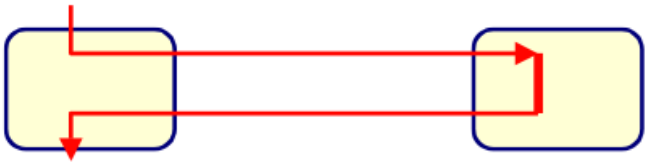
- Exemples
  - Java Remote Method Invocation (RMI) : appel d'objets distants en Java - Sun
  - Common Object Request Broker Architecture (CORBA) : support pour l'exécution d'objets répartis hétérogènes - OMG
  - DCOM, COM+ : Distributed Common Object Model - Microsoft
- Schéma commun : ORB (Object Request 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 oeuvre (réalisation, services)
- Notre objectif
  - élaborer des patrons pour les aspects ci-dessus
  - proposer un canevas pour la structure d'un ORB

Schémas d'interaction

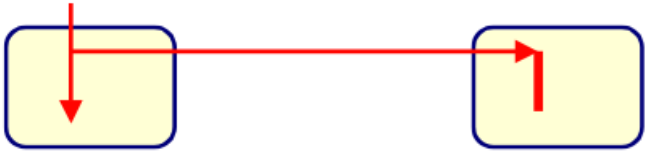
- **Synchrones**



Couplage fort

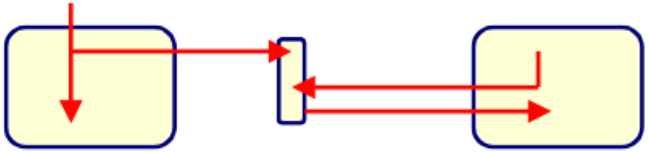
RMI, CORBA, COM, ...

- **Asynchrones**



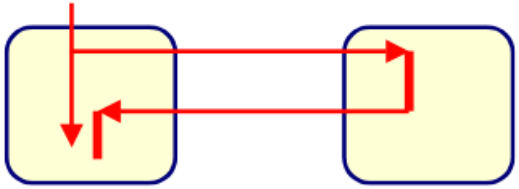
Couplage faible

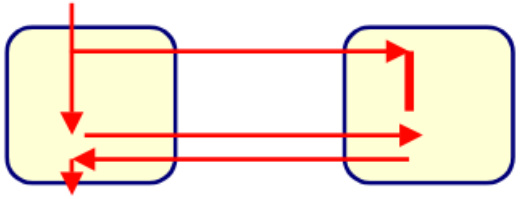
Événements



Queues de messages

- **Semi-synchrones**



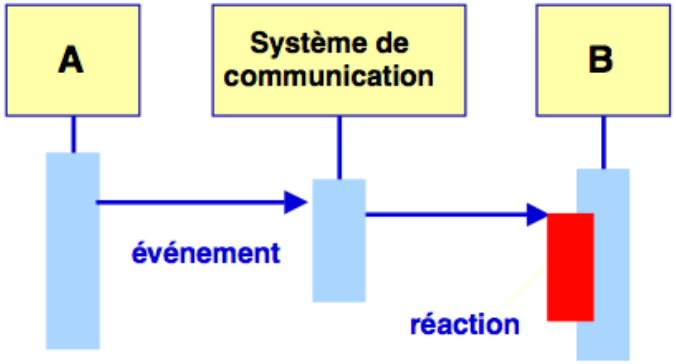


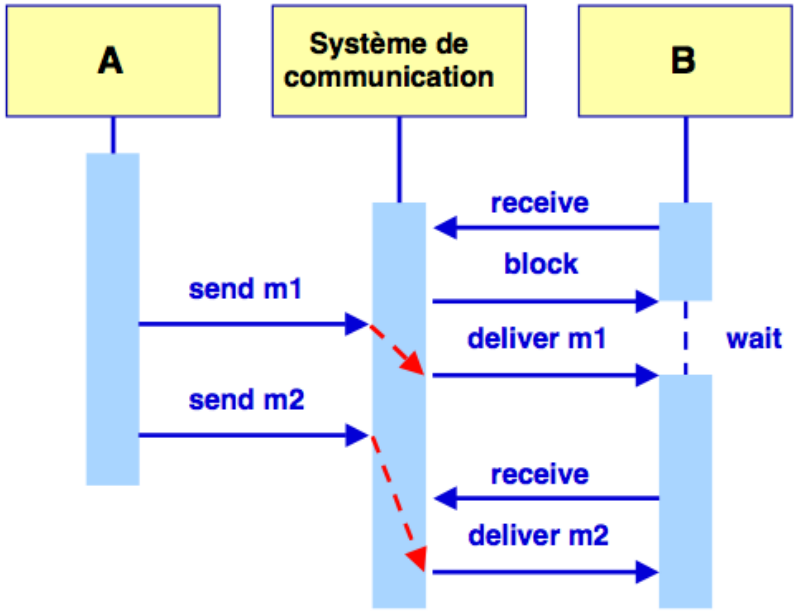
Combinaisons synchrone-asynchrone

Schémas d'interaction

Schémas asynchrones

Exécution parallèle de l'émetteur
et du récepteur
Couplage faible

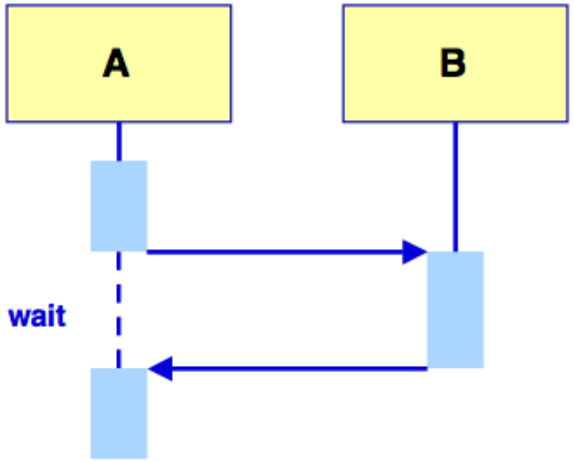




- Événement-réaction
- Messages asynchrones

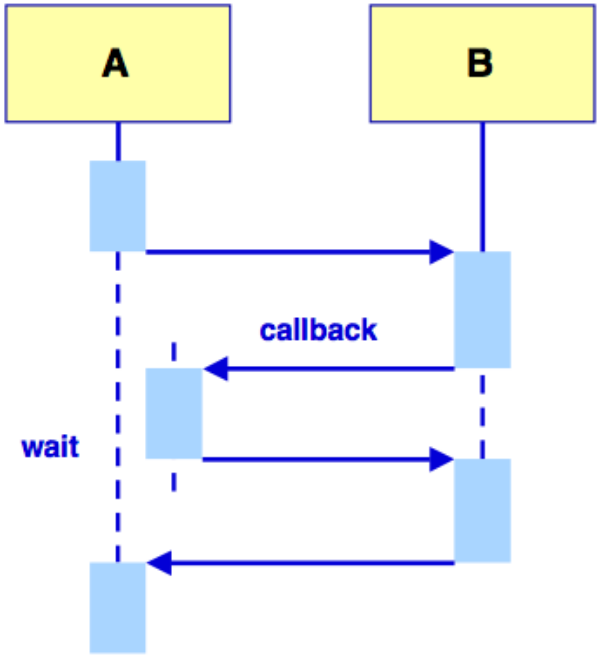
événements et messages peuvent être ou non mémorisés

Schémas d'interaction



Appel synchrone

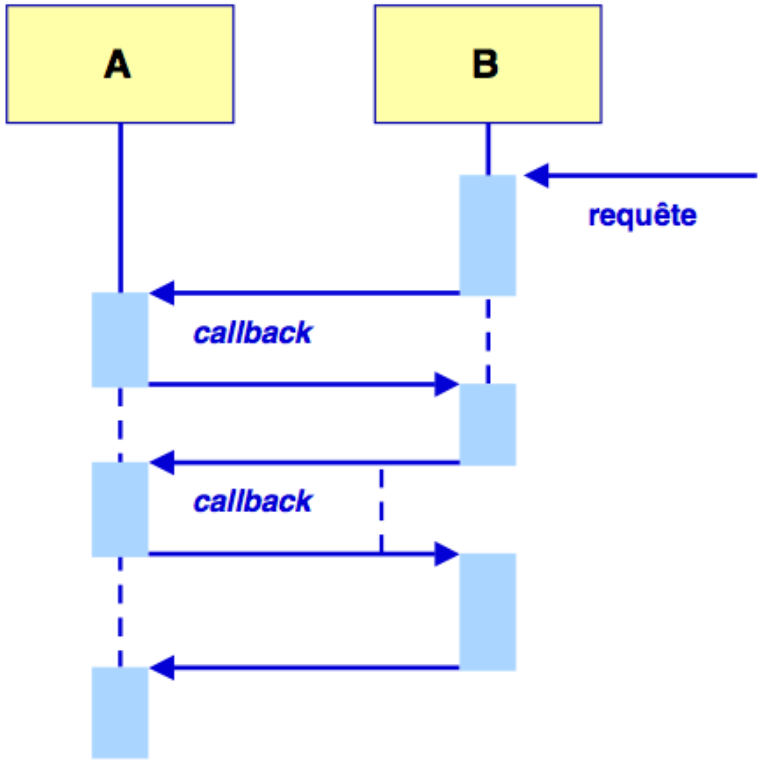
- L'émetteur (client) est bloqué en attendant le retour
- Couplage fort



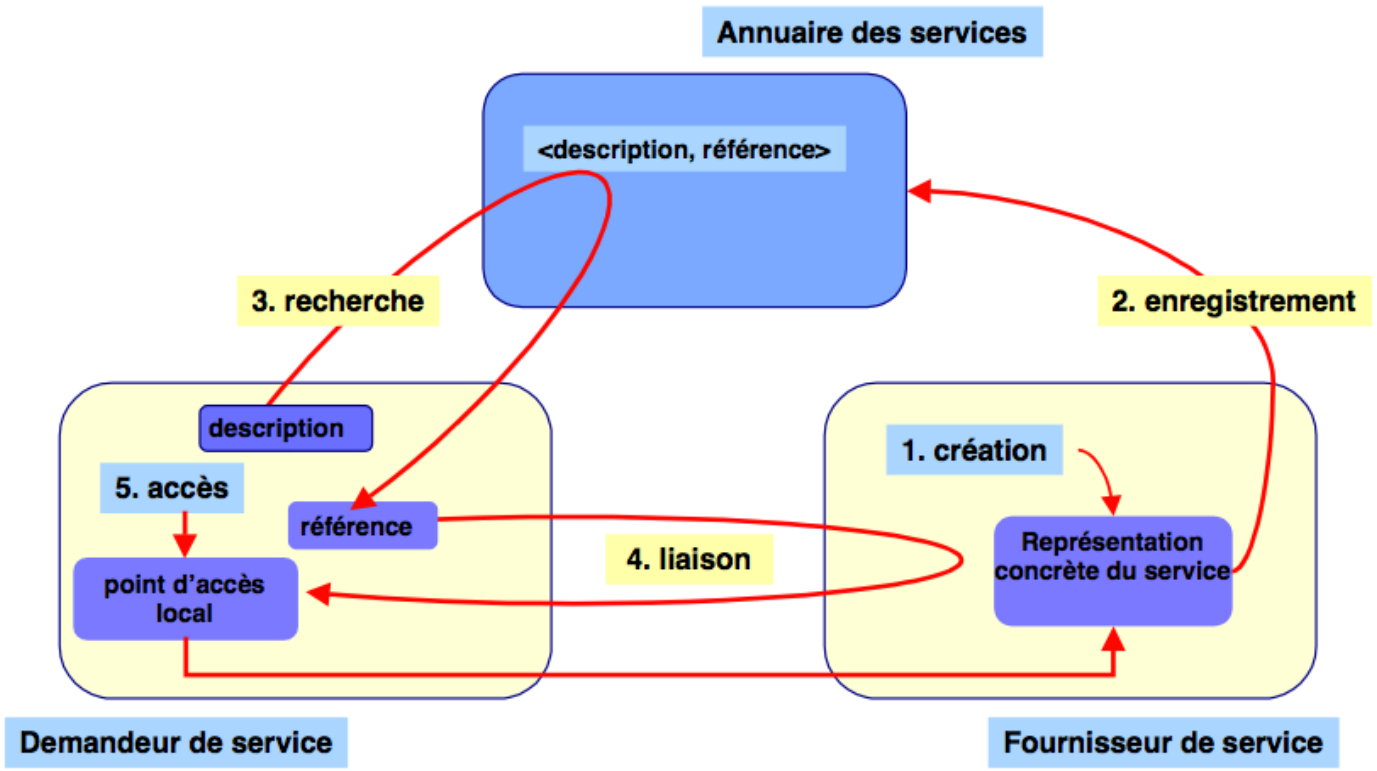
Avec appel en retour (callback)

Schémas d'interaction

- Inversion du contrôle
  - Situation où B "contrôle" A
  - La requête de service pour A est déclenchée depuis l'extérieur

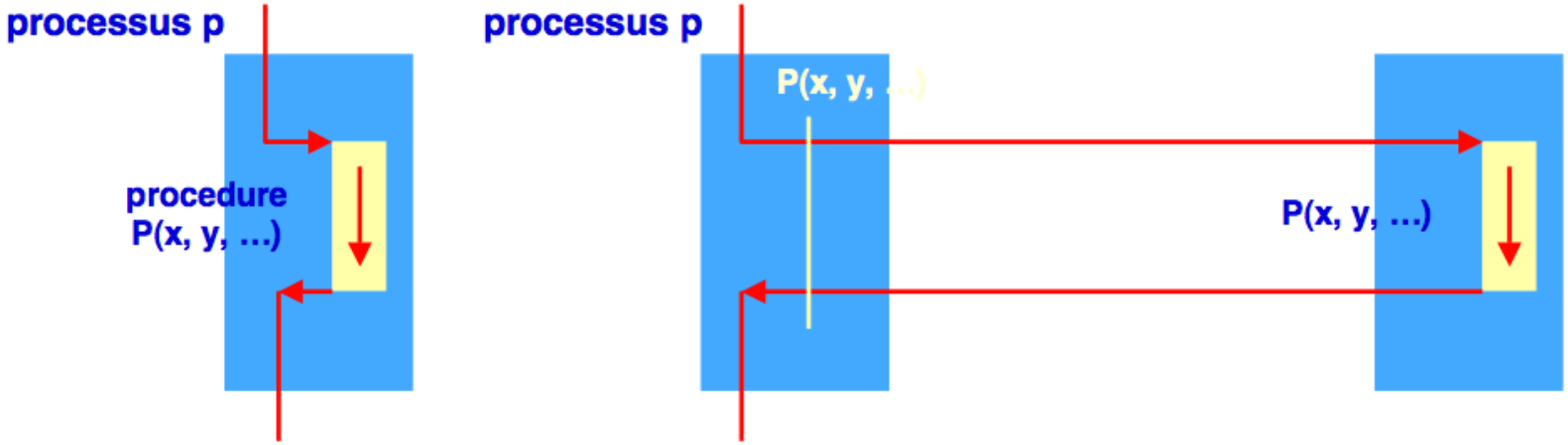


Accès à un service



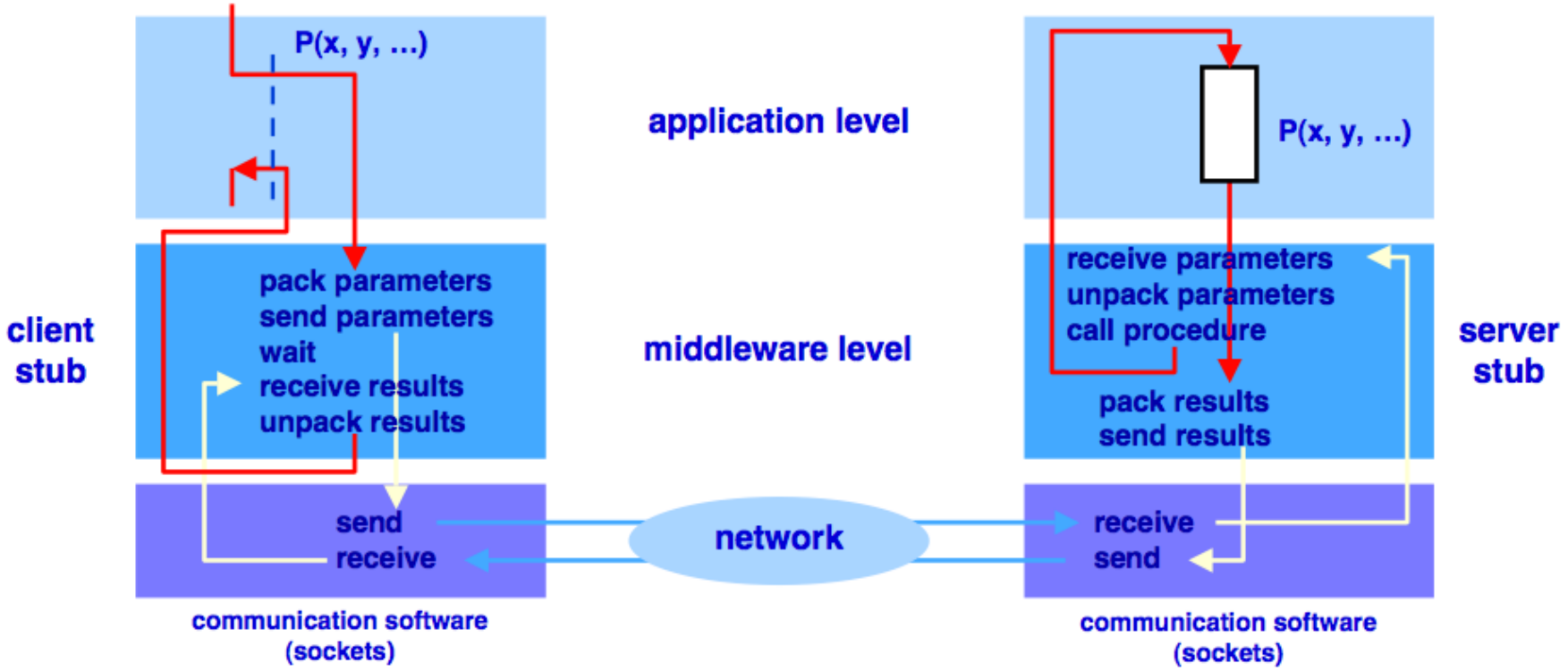
Remote Procedure Call (RPC)

- 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.

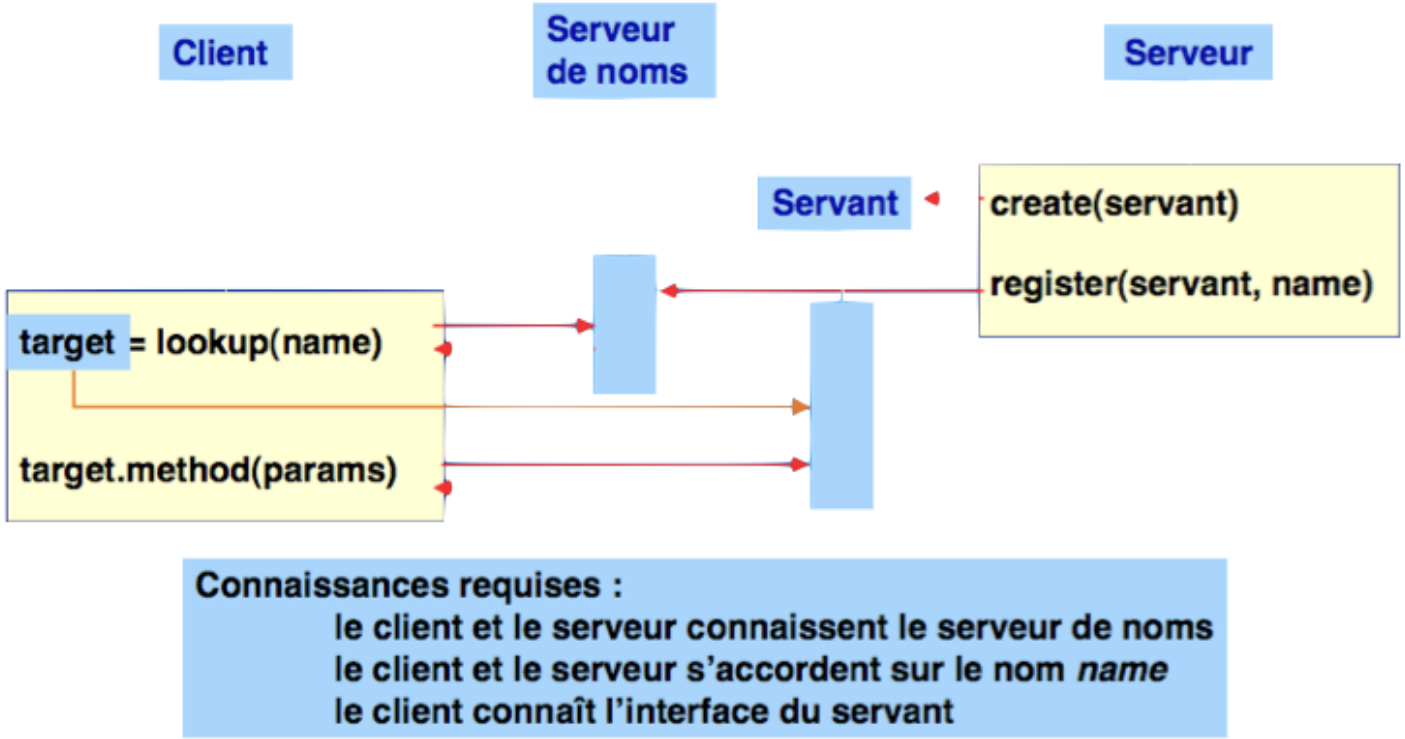
Mise en oeuvre de RPC



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
  - 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)



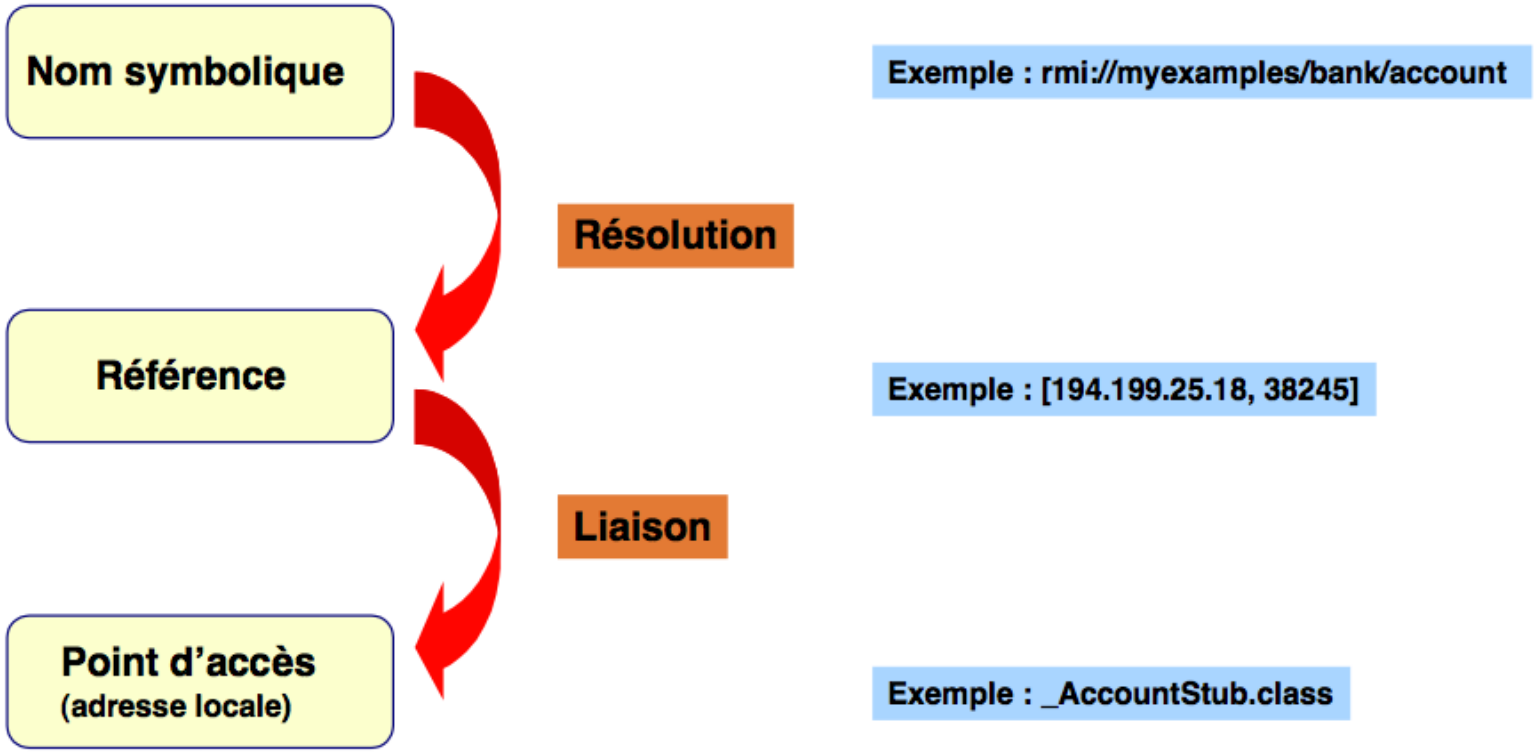
Les étapes d'un appel réparti (vues du système)

- 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 porte])
    - Mais référence # chaîne d'accès '

Liaison répartie

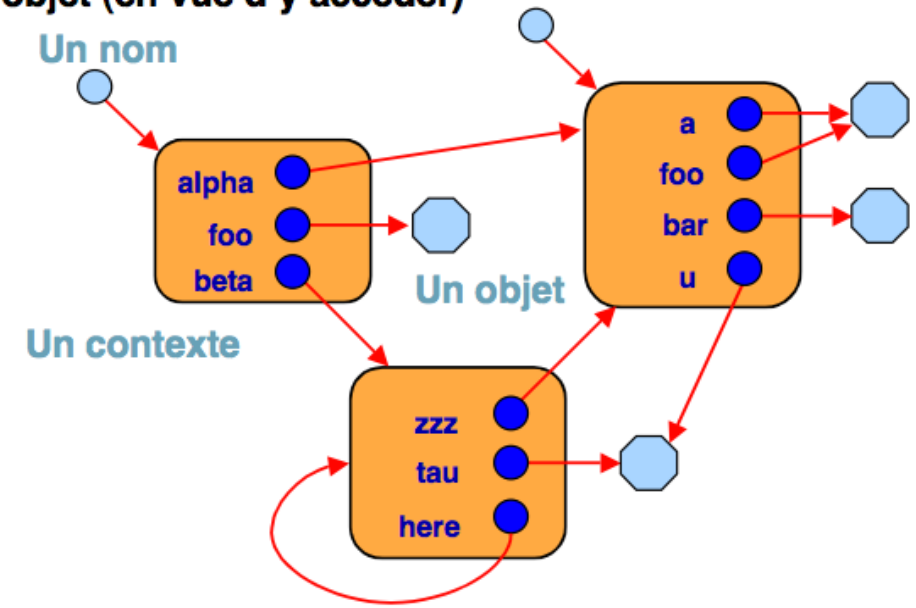


Rappel rapide sur les noms

Deux sortes de noms

- Identificateurs : distinguer un objet des autres
- Références : localiser un objet (en vue d'y accéder)

Noms contextuels

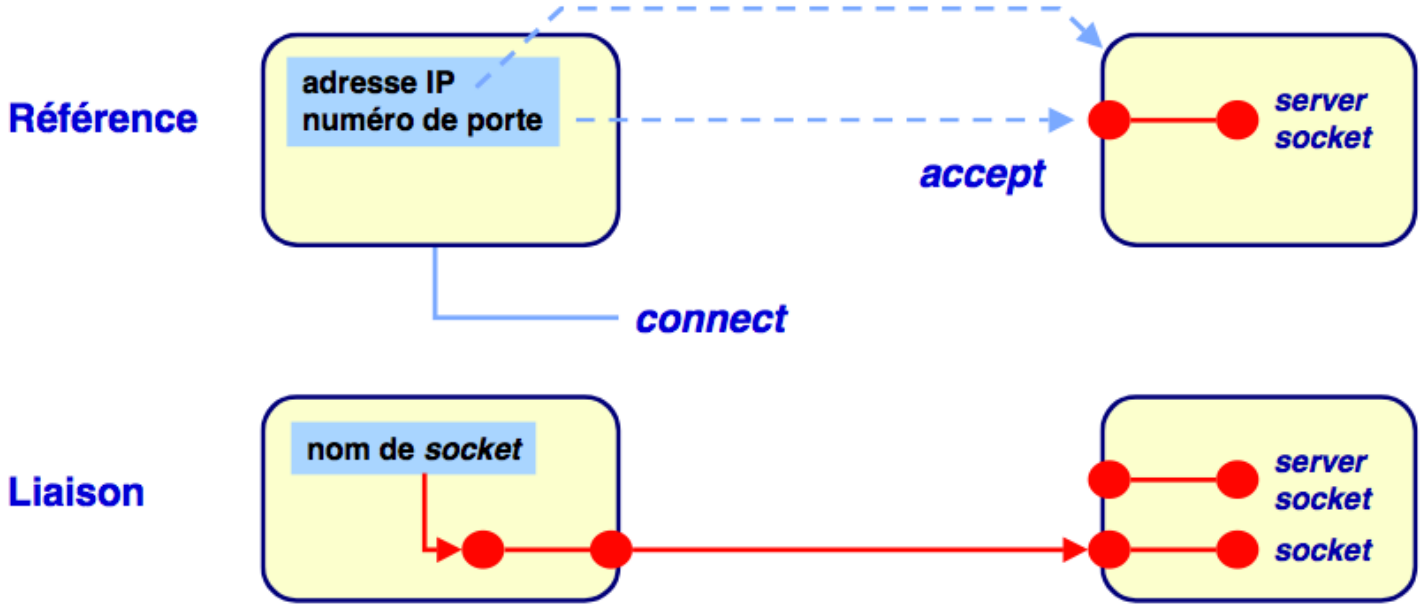


Résolution et liaison des noms

- Résoudre un nom (dans un contexte)
  - A partir du nom, trouver l'objet
  - Processus récursif
    - target = context.resolve(name) 3 issues possibles pour target ou name.resolve()
    - 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
  - A 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

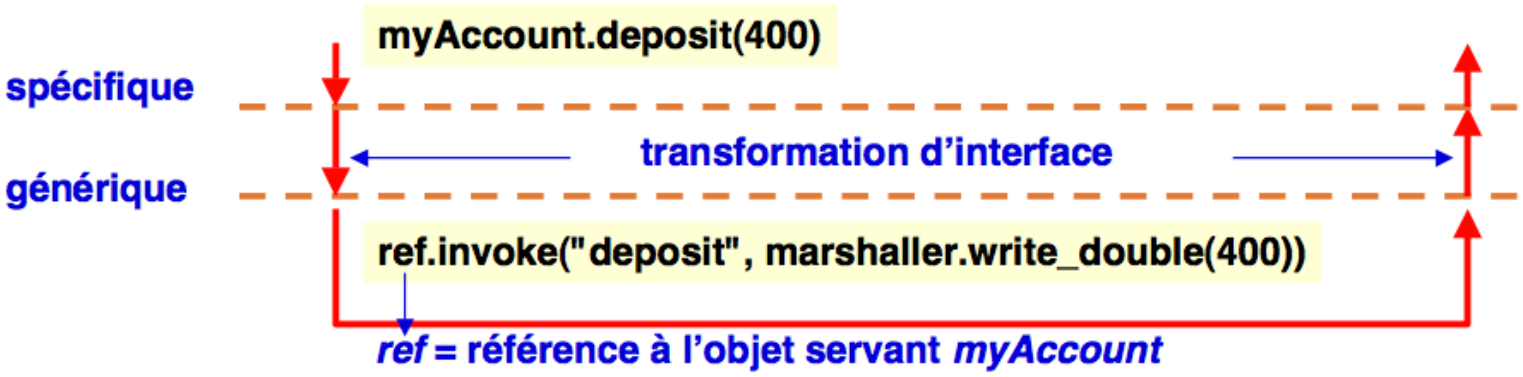


Un patron pour la liaison répartie

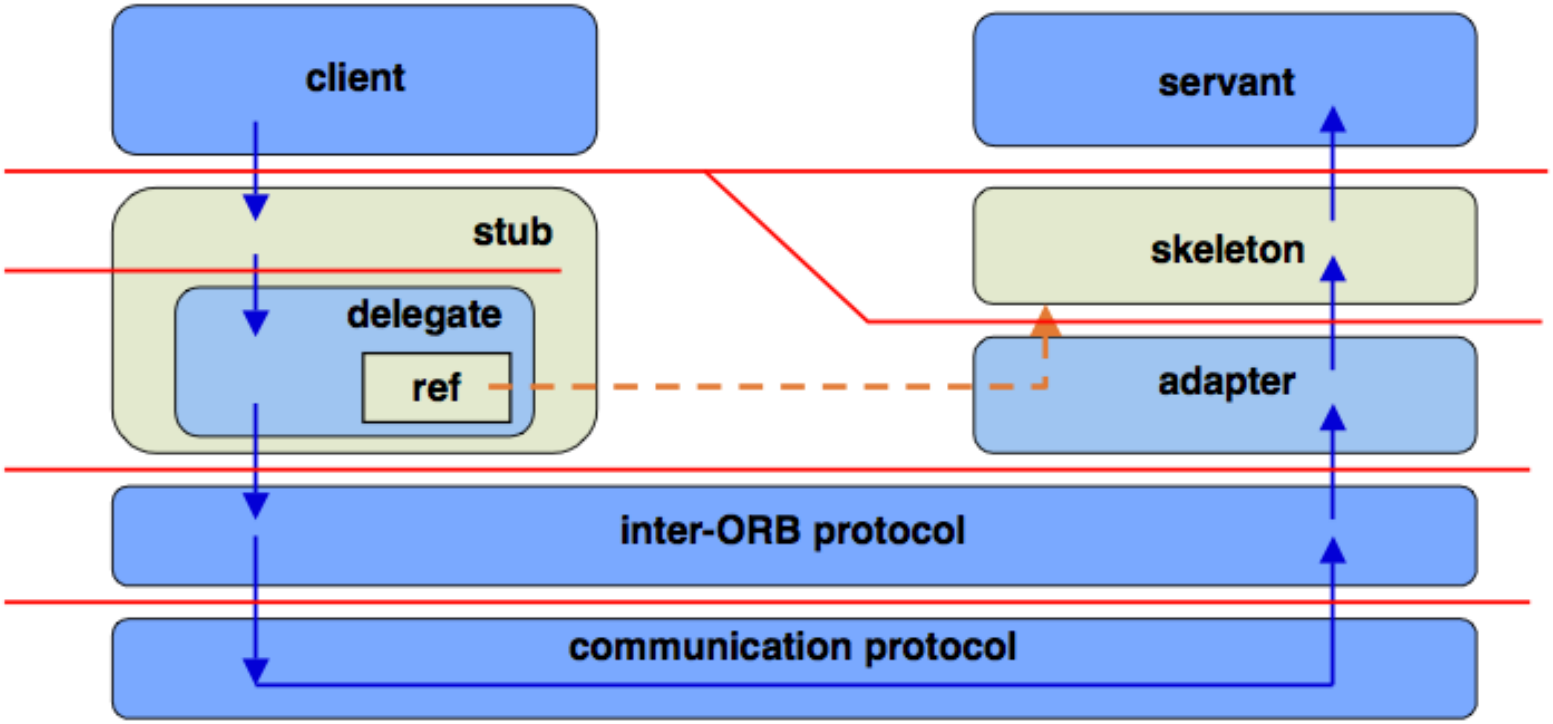
- La liaison est établie en 2 phases
- Côté serveur (export)
  - "publication" de l'objet à lier (identification)
  - préparation de certains éléments de la liaison
- Côté client (bind)
  - établissement de la liaison par création et assemblage des constituants de l'objet de liaison
- Exemple
  - La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

Structure d'un appel distant

- 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



Structure d'un appel distant



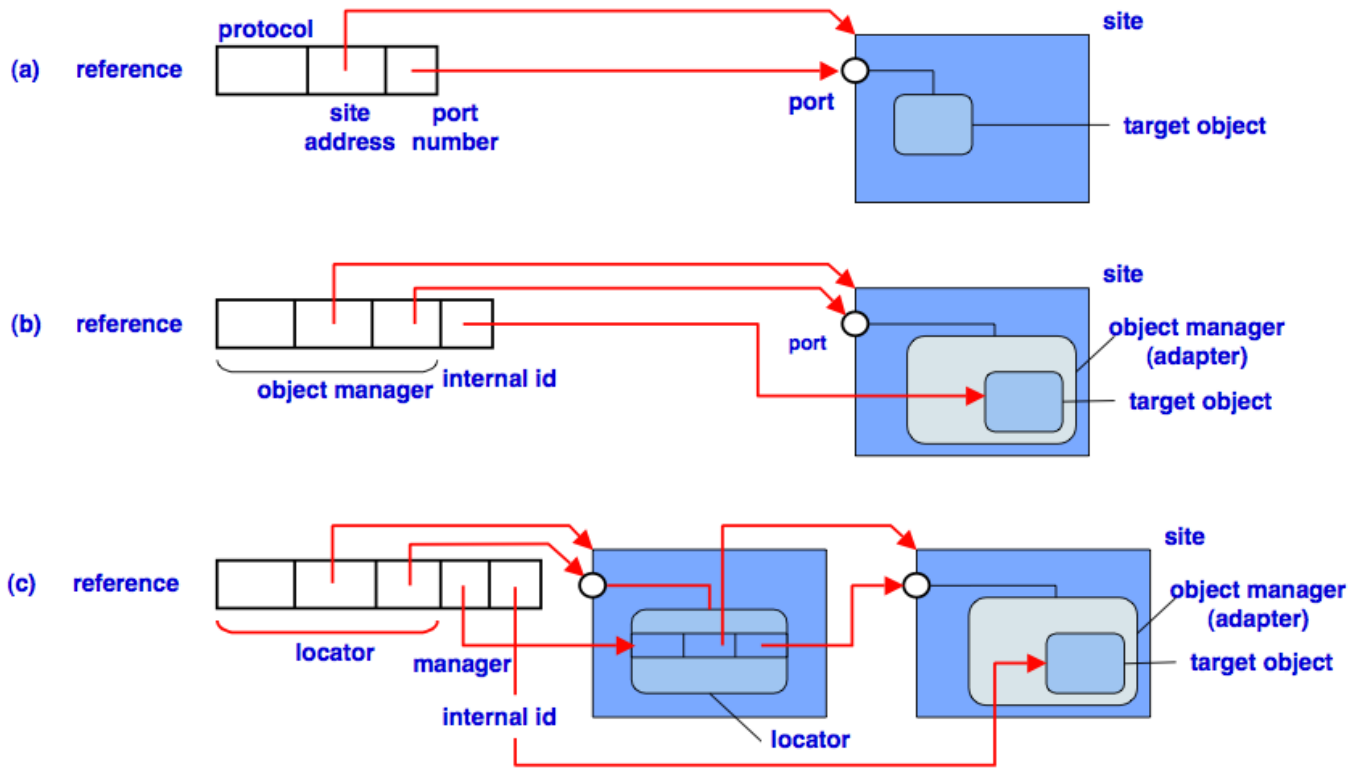
Fonctions de l'adaptateur

- 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



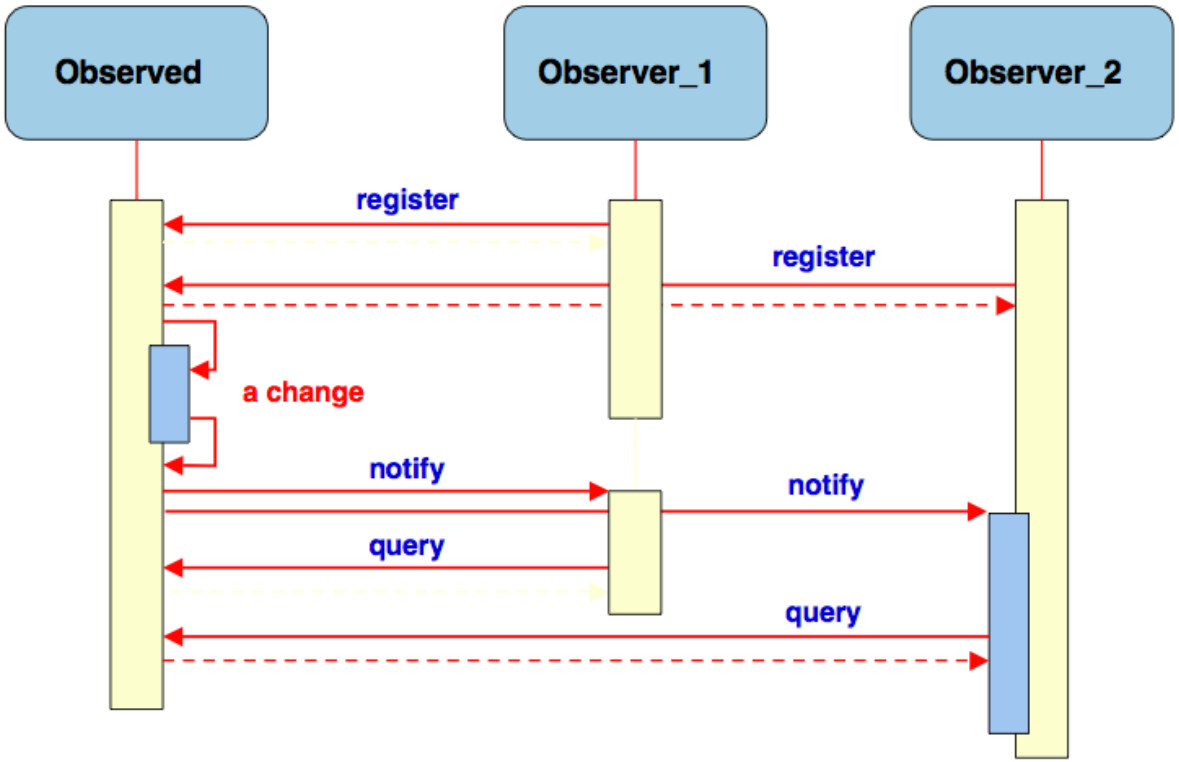
Coordination

- 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

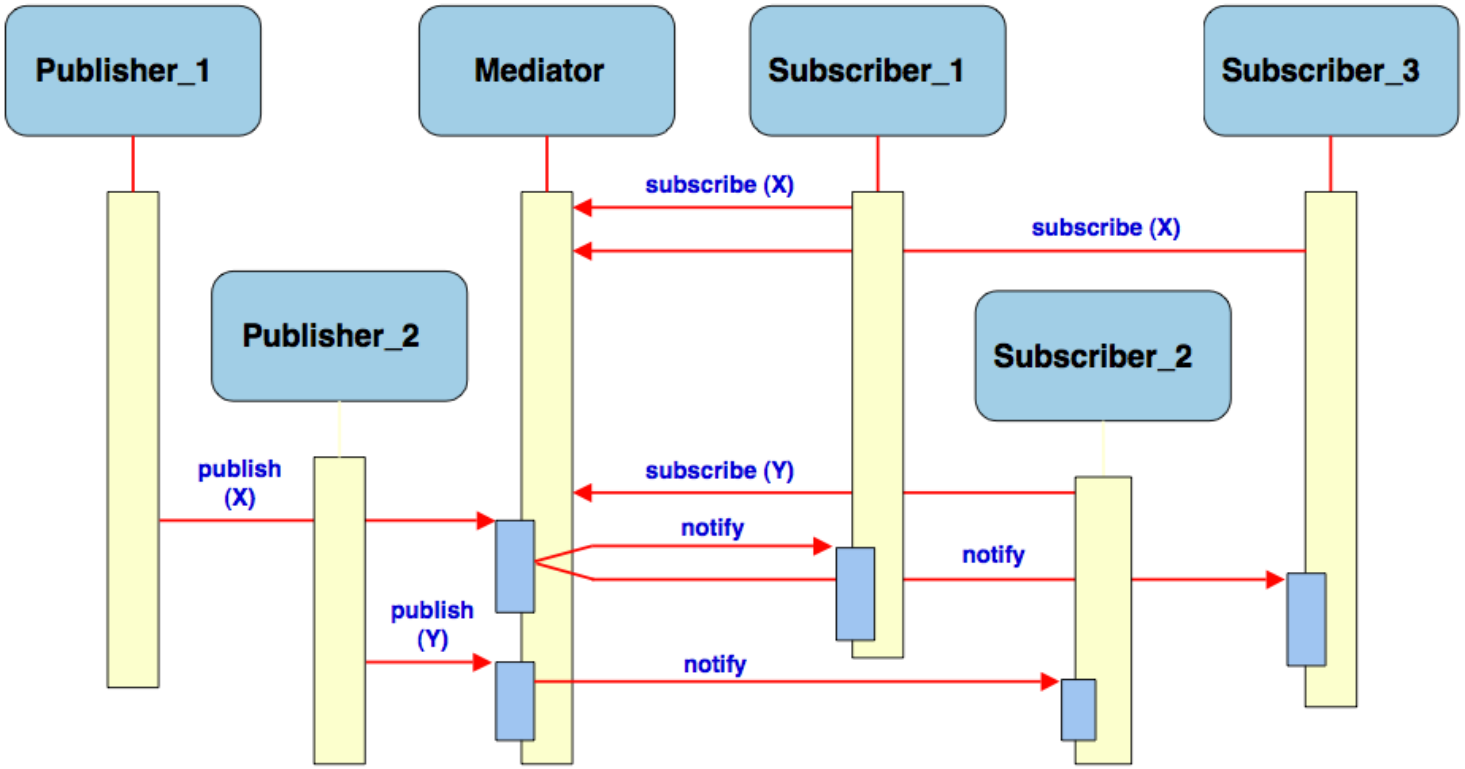
Usage d'Observer



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'événements et réaction à ces événements (autre formulation de Observer)
- Problème
  - Propriétés souhaitables
    - Comme Observer (indépendance, évolution dynamique)
    - Pas de rôle prédéfini
    - électivité 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)

Usage de Publish/Suscribe