NSY102
Conception de logiciels Intranet
Introduction

Cnam Paris
jean-michel Douin, douin au cnam point fr
10 février 2015

Sommaire

- Introduction
- Généralités
- Tendances
  - historique
  - API & Intergiciel
- Patrons et Patrons architecturaux
- Les canevas
  - Structure d ’un ORB (Object Request Broker)
  - Fonctionnalités attendues
  - Conteneur de composants
- Architectures

Principale bibliographie utilisée

- [Sch06]Le site de douglas Schmidt
  - http://www.cs.wustl.edu/~schmidt/patterns.html
  - ce site http://www.cs.wustl.edu/~schmidt/POSA/
- [Kra06]
  - La présentation de S. Krakowiak faite à l 'école d 'été ICAR 2006

    http://sardes.inrialpes.fr/ecole/2006/ http://www2.lifl.fr/icar/Chapters/Intro/intro.html
- [SSRB00]
  - les travaux de douglas Schmidt : http://www.cs.wustl.edu/~schmidt/POSA/
- [VKZ05]
  - le site de Uwe Zdun : http://www.infosys.tuwien.ac.at/Staff/zdun/
  - ce livre : http://www.infosys.tuwien.ac.at/Staff/zdun/remotingPatterns/index.html
  - ce support http://www.infosys.tuwien.ac.at/Staff/zdun/teaching/evs/evs.pdf
  - informations générales http://www.edlin.org/cs/patterns.html

Introduction

- **Systèmes distribués, généralités**
  - **Où ?**
  - **Pourquoi ?**
  - **Contraintes/Objectifs**
  - **Attention**

Où ?

- **Internet**
  - **(protocoles HTTP, FTP, SMTP, SNMP, telnet ….)**
- **Télécommunications**
  - **RNIS *(Réseau Numérique à Intégration de services)*, PABX** (*Private Branch Exchange*)
- **B2B collaboration**
  - **XML*/ Web services***
- **Transactions financières**
  - **SWIFT** *(Society for Worldwide Interbank Financial Telecommunications)*
- **Systèmes embarqués**
  - **AUTOSAR** *(AUTomotive Open System ARchitecture)*
- **Applications**
  - **Grilles** *(ProActive)*, **P2P**
- **...**

Pourquoi ?

- **Meilleures performances, adaptabilité**
  - **(répartition de la charge)**
- **Tolérance aux pannes**
- **Services et clients dynamiques**
  - **imprévisibles**
- **Maintenance et déploiement**
- **Sécurité**
- **Business integration EAI**
  - **(Enterprise Application Integration)**

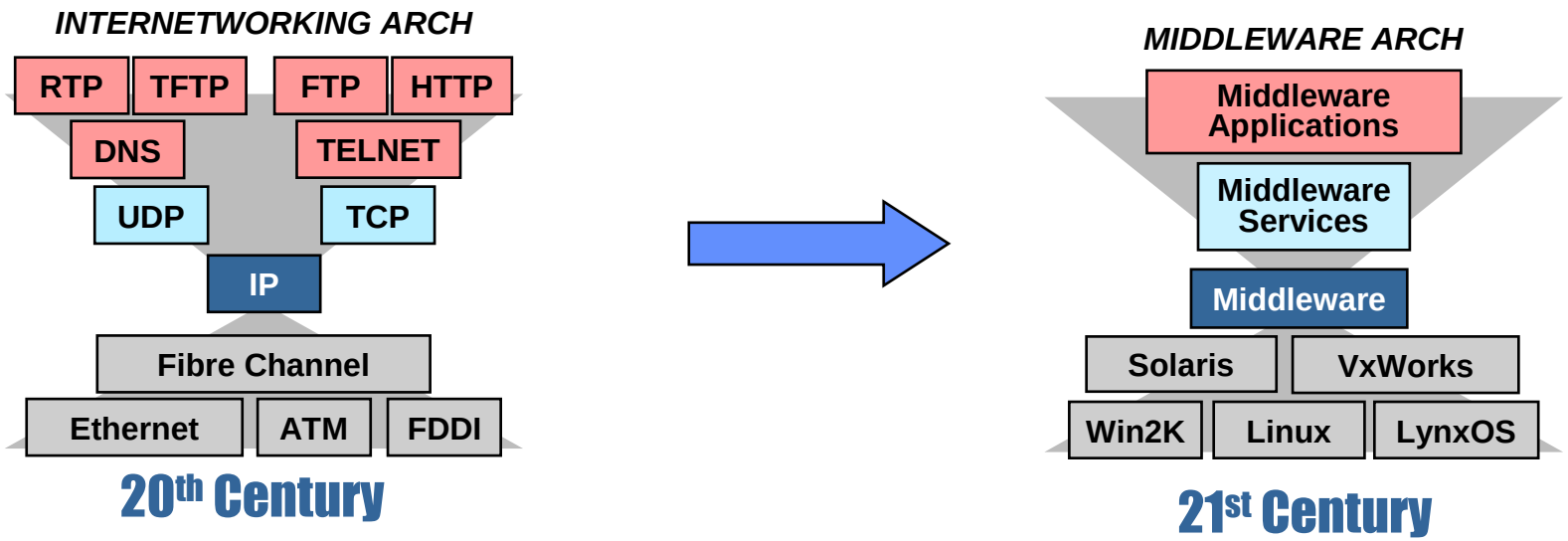
Contraintes/Objectifs/Défis

- **Temps de réponse**
  - **respect de contraintes temporelles**
- **Temps de réponse fluctuant**
  - **Deux appels distants n 'ont pas les mêmes temps de réponse**
- **Concurrence**
  - **un « vrai » parallélisme**
- **Répartition de charge *(Scalability)***
  - **En fonction de la charge, connue à l 'exécution**
- **Panne partielle**
  - **Continuité de fonctionnement avec une partie du système en panne**

Attention

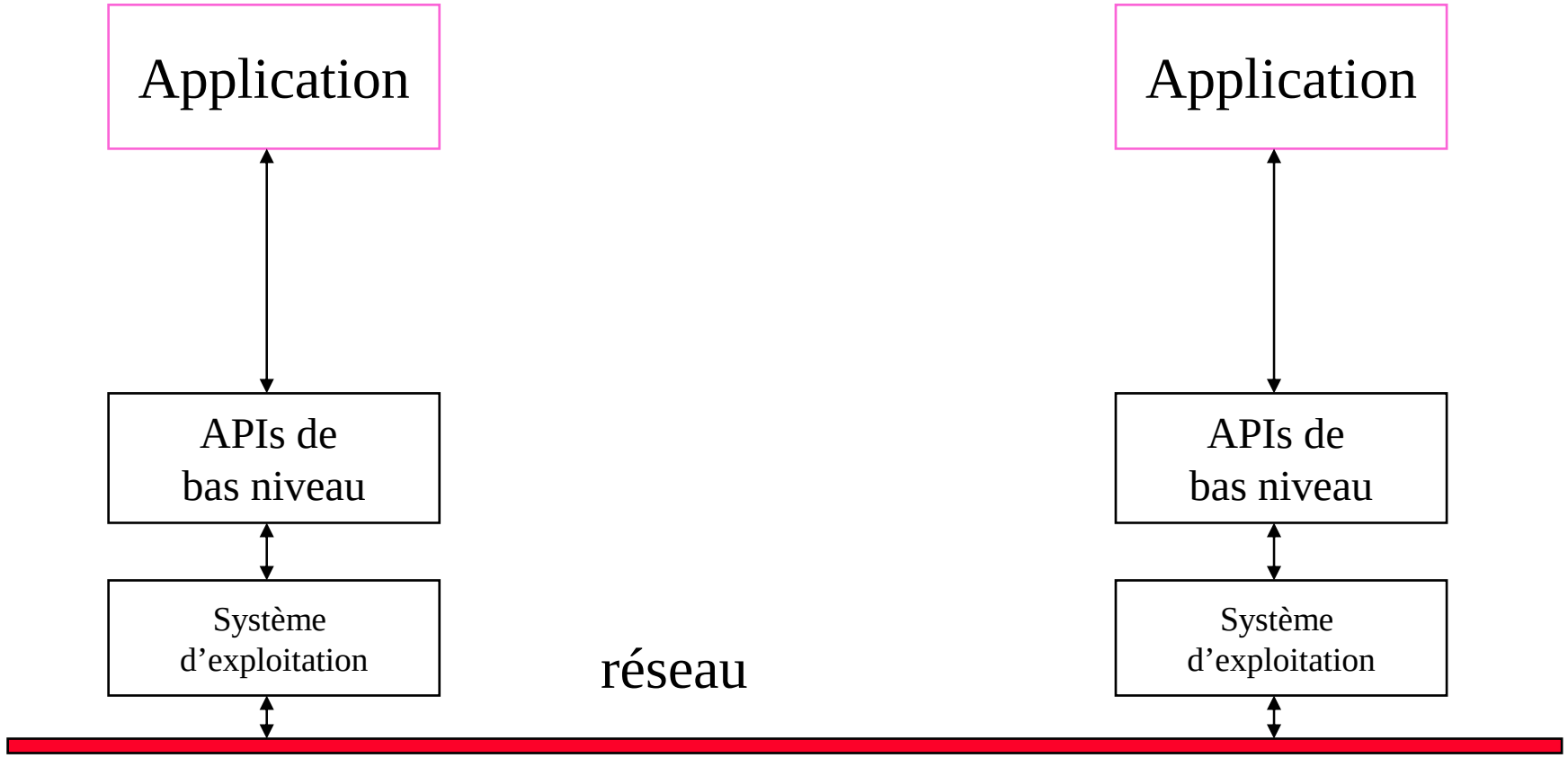
- **les ennuis potentiels croissent**
  - **Complexité, concurrence, temps de réponse etc….**
  - **Outils de conception ?**
  - **Catalogue de modèles de conception ?**
    - **Solutions réutilisables ?**

Tendances



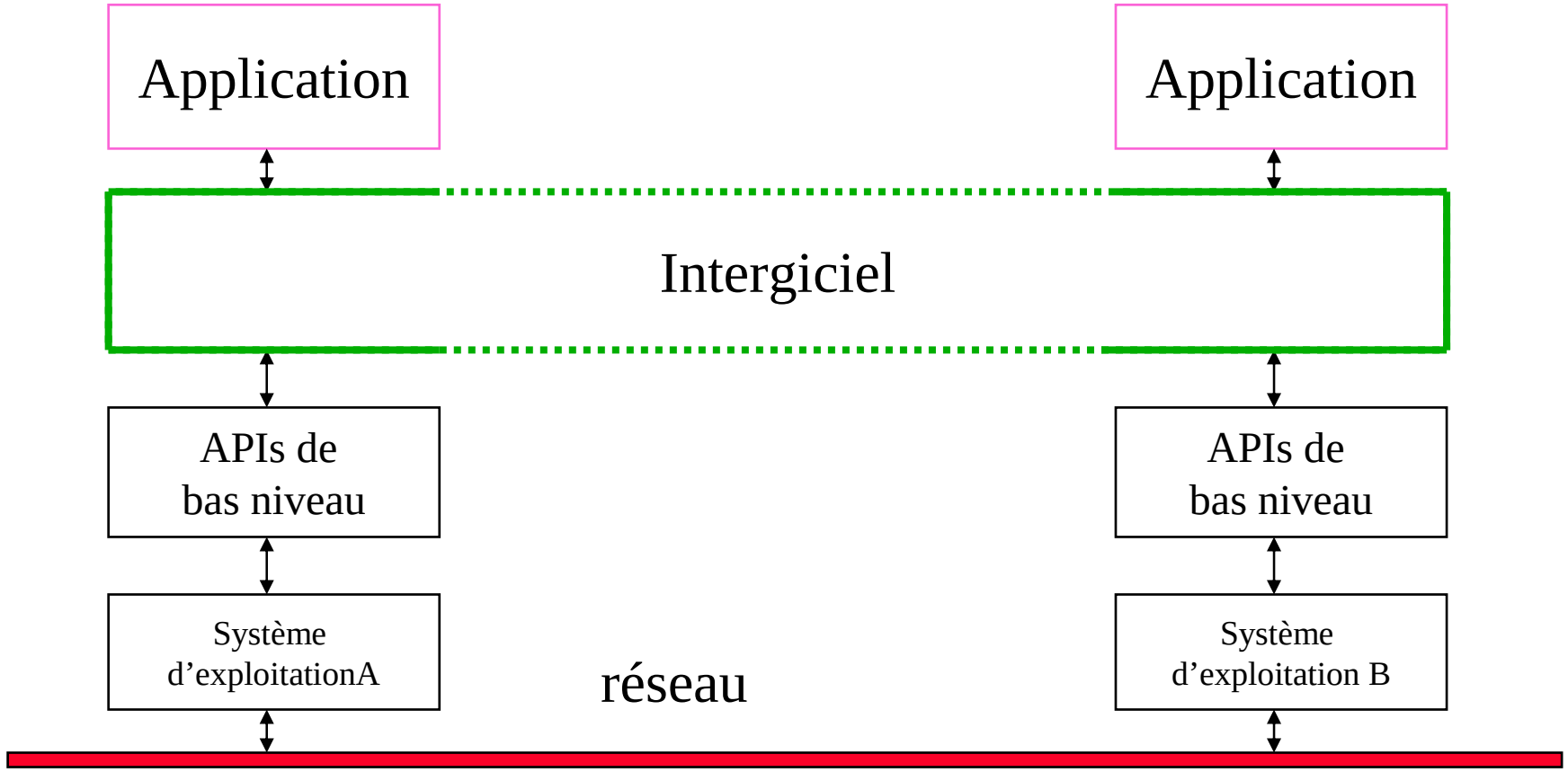
- **[Sch06]**

Avant



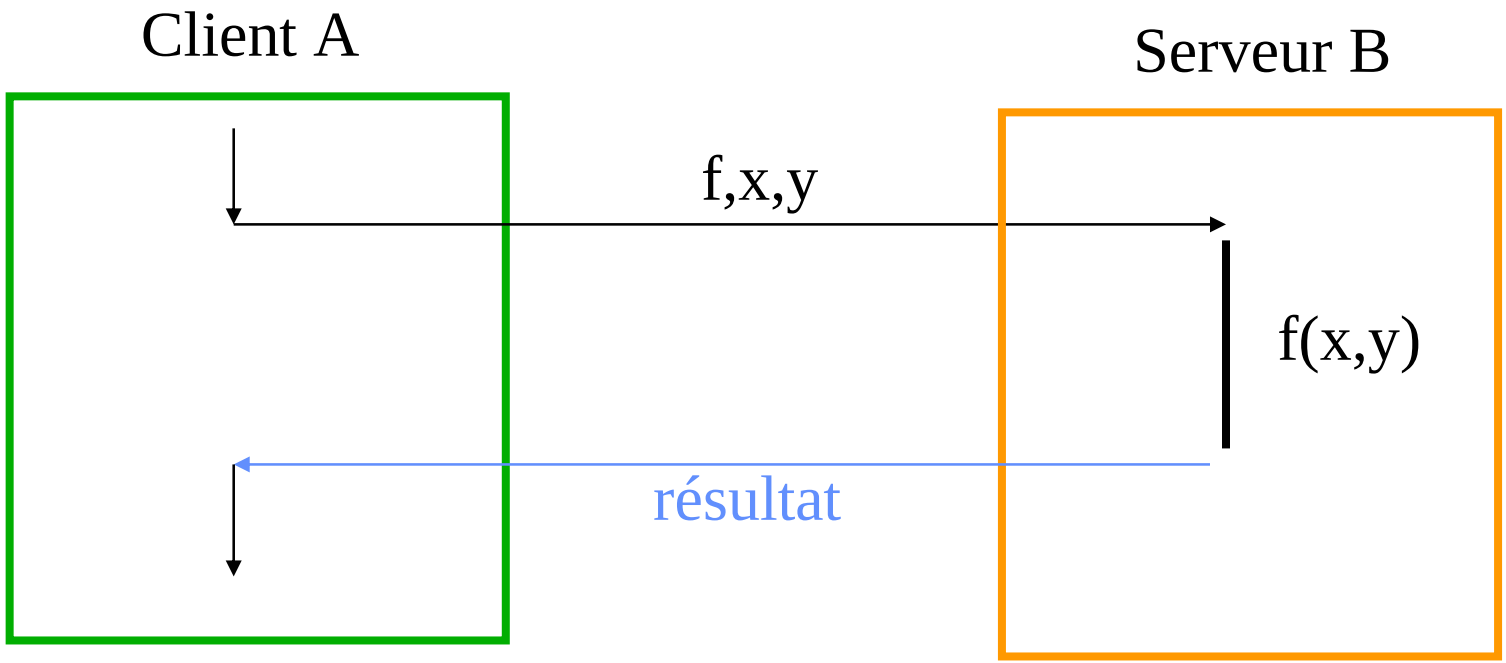
- **Évolution et maintenance difficiles**
- **Sources d'erreurs…**
- **Comment s'abstraire de la communication ?**

Tendances



- **Un logiciel de communication : intergiciel *(middleware)***

Un exemple simple d 'intergiciel : RPC



- **Appel distant de la procédure int résultat = f(x,y)**
  - **emballage *(marshalling)*, déballage *(unmarshalling)* des paramètres**
  - **réaction aux défaillances côté serveur, le client est prévenu**

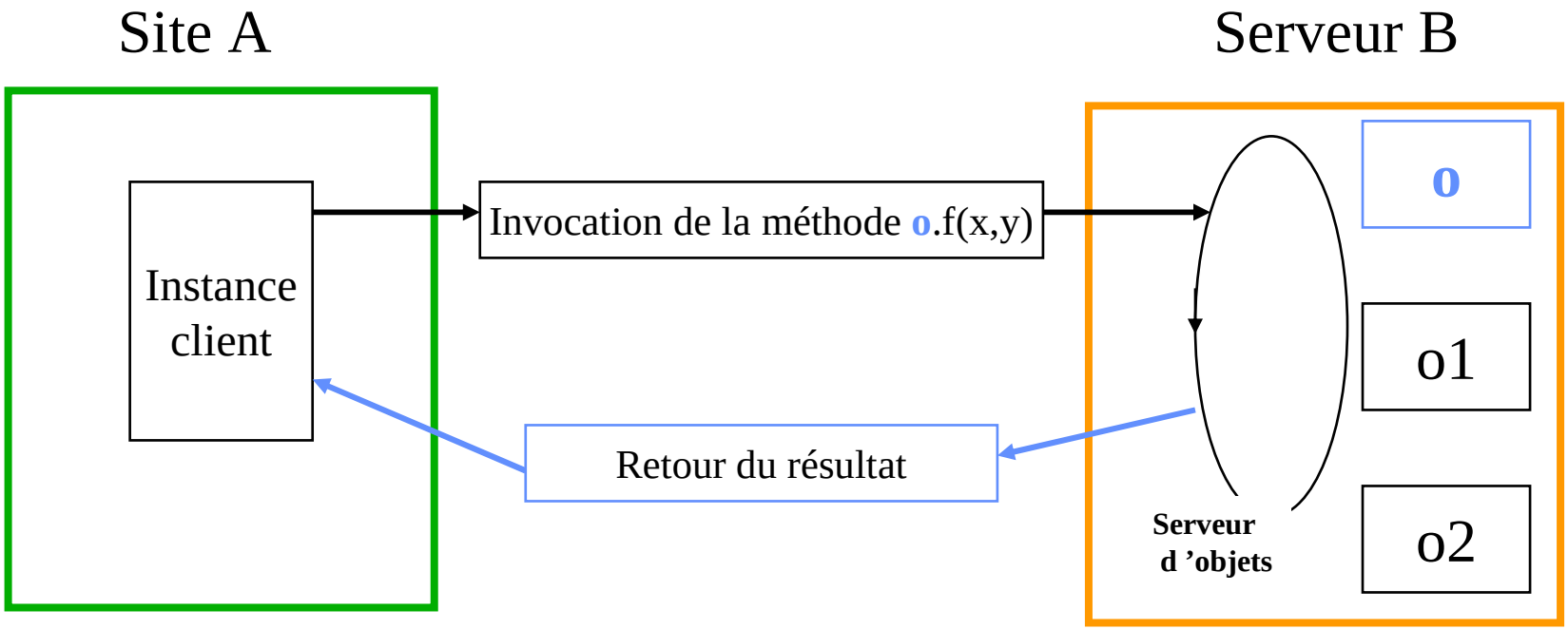
OO-RPC

- **Contexte « Objet »**

- **Côté client**
  - **Appels de méthodes distantes,**
  - **Identifiant/référence distante de l ’objet.**

- **Côté serveur**
  - **Serveur/servant d ’Objets,**
  - **Sélection de l ’objet,**
  - **Méthodes à exécuter, propagation éventuelle des erreurs.**

Intergiciel OO-RPC



- **emballage** *(marshalling),* **déballage** *(unmarshalling)* **des paramètres**
- **réaction aux défaillances, exceptions**
- **voir** [VKZ05]

Intergiciel OO-RPC

- **Le client connaît l ’adresse du serveur**
  - **localisé au plus tard, au moment de l ’appel.**
- **Un service est enregistré auprès d ’un annuaire**
  - **le nom de la méthode, la version, son descripteur.**
- **L ’ annuaire est connu de « tous »**
  - **stratégie de sécurité.**
- **Le client**
  - **interroge l ’annuaire,**
  - **obtient une référence sur le service distant,**
  - **« emballe » les paramètres, « exécute » le service distant.**
- **Le serveur**
  - **sélectionne l ’objet/service associé à la référence,**
  - **« déballe » les paramètres,**
  - **déclenche la méthode et retourne (éventuellement) un résultat**
    - **Se charge de propager l’exception**

Fonctions attendues de l 'intergiciel

- **de [Kra06]**

- **Proposer une API de haut Niveau**
- **Masquer l 'hétérogénéité**
- **Rendre la répartition invisible**
- **Faciliter la programmation répartie**

Tendances

- **Patrons** *(pattern)*
  - **Conception**
  - **Architecturaux**
  - **...**
- **Canevas (***framework***)**
- **Services et composant**
- **Interfaces et contrat**

Patron [Kra06]

- **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 ")

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

 ICAR'06 19

- **Itérateur, composite, observateur/observé, …**

Canevas [Kra06]

- **Définition**
  - Un canevas est un “squelette” de programme qui peut être réutilisé (et adapté) pour une famille d’applications
  - Il met en œuvre 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

- **.NET, J2EE, OSGi, Android, MIDP, JCRE ...**

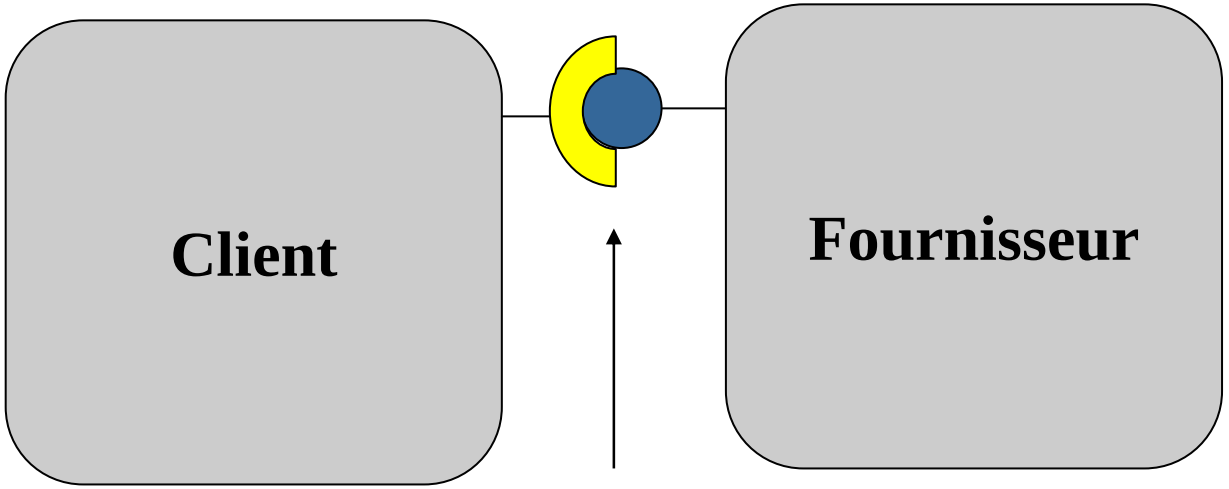
Services et interfaces

- **Extrait de [Kra06]**
- **Un système est un ensemble de composants qui interagissent**

What is a Service?
A service is a contractually defined behavior that can be implemented and provided by any component for use by any component, based solely on the contract.

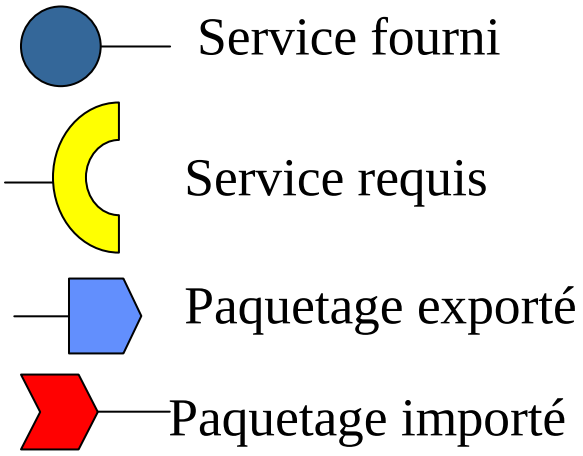
- **Un service est un comportement défini par un contrat, qui peut être implémenté et fourni par un composant afin d 'être utilisé par un autre composant sur la base exclusive du contrat**
  (*http://www.openwings.org/download/specs/ServiceOrientedIntroduction.pdf)*
- **Un service est accessible via une interface**
- **Une interface décrit l 'interaction entre le client et le fournisseur du service**

Interfaces, une notation [Sch06]



Nécessaire conformité

- **interface / service**
- **Le contrat doit être conforme**
  - **Appelant : le client**
  - **Appelé le détenteur/fournisseur du service**



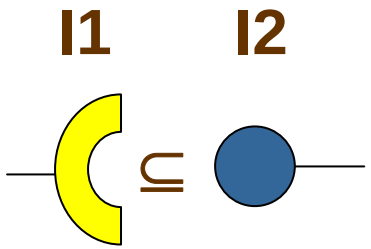
Conformité/contrat

- **Syntaxique**
- **Sémantique**
- **Synchronisation**
- **Performances, ….**

Conformité

- **décrit ici**
  - **www.irisa.fr/triskell/publis/1999/Beugnard99.pdf**
  - **www.i3s.unice.fr/~mh/RR/2002/RR-02.49-R.ROUSSEAU.pdf**

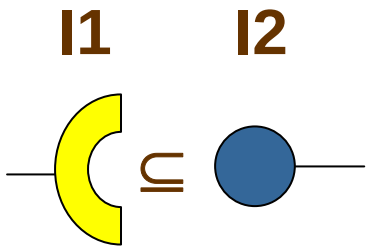
- **Syntaxique**
  - **forme des opérations, types des paramètres, (résolu statiquement)**

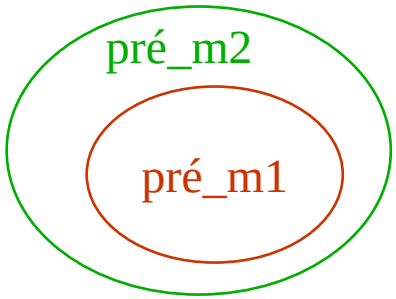


  - **soit : l 'interface I1 proposant la méthode m1 et I2 la méthode m2**
  - **T(X) type de X, T1 est un sous type de T2 noté T1 ⊆ T2**
    - **T(I1)⊆T(I2) si I2 a au moins le même nombre de méthodes que I1**
    - **et pour toutes les méthodes**
      - **m1, m2 ont le même nombre de paramètres et valeurs de retour**
      - **pour les valeurs de retour r1 de m1, r2 de m2 T(r2) ⊆ T(r1)**
      - **pour les paramètres x1..xn de m1 et y1..yn de m2 T(x1) ⊆ T(y1)**

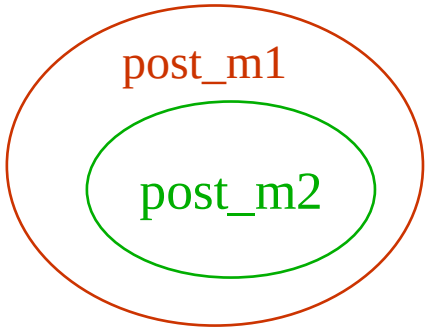
Conformité sémantique

- **Sémantique**
  - **Par exemple avec les pré et post assertions [Hoare en 1969]**
  - **De l ’interface I1**
    - **{pré_m1} m1 {post_m1}**
  - **De l ’interface I2**
    - **{pré_m2} m2 {post_m2}**
  - **conforme si**
    - **{pré_m1} ==> {pré_m2} et {post_m2} ==> {post_m1}**





A l ’appel



Au retour

Conformités de synchronisations et QOS

- **Synchronisation**
  - **Contraintes sur l 'ordre et la concurrence des exécutions**
  - **Cycle de vie à respecter**

- **Qualité de Service, *Service level Agreement***
  - **Disponibilité**
  - **Performances**
  - **Sécurité**

- **voir http://ibis.in.tum.de/staff/paschke/rbsla/index.htm**

Contract Representation and Management for Service Level Agreements

Interface ou langage commun

- **Un service est accessible via une interface**
- **Une interface décrit l ’interaction entre le client et le fournisseur du service**

IDL comme **Interface** Definition Language

- **IDL / CORBA**

```
module Counter{
  interface Count{
     attribute long sum;
     long increment();
  };
};
```

- **et Corba/Java avec idlj**
  - **http://java.sun.com/j2se/1.5.0/docs/guide/rmi-iiop/toJavaPortableUG.html**

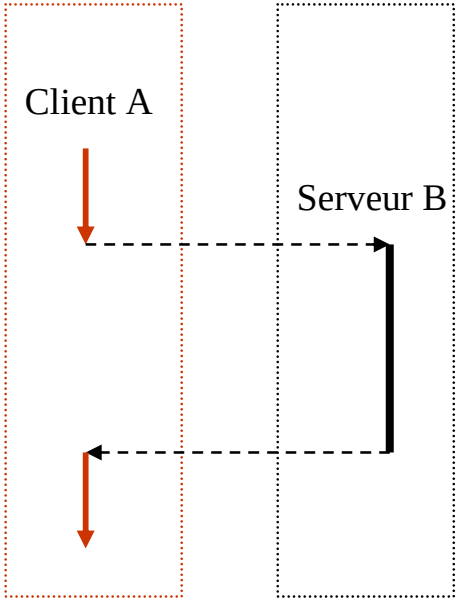
- **en Java/RMI**

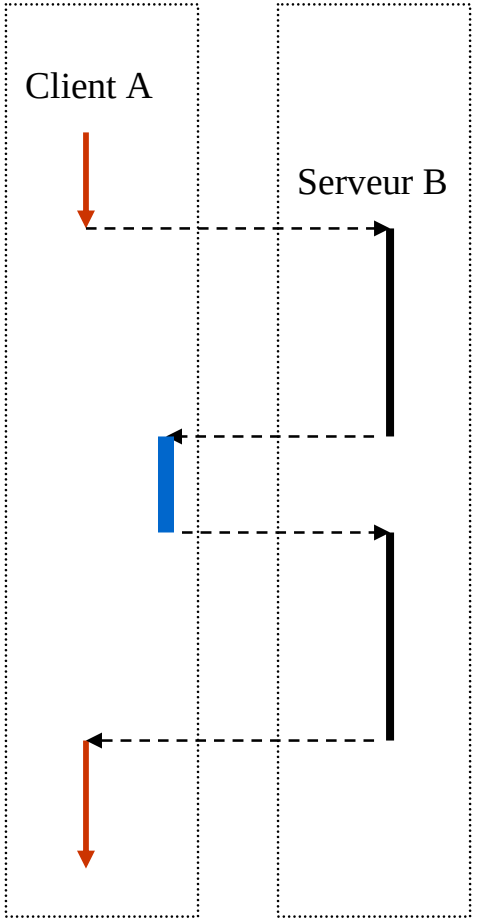
```
public interface Count extends Remote{

  public long increment() throws RemoteException;
}
```

Schémas types d 'interaction

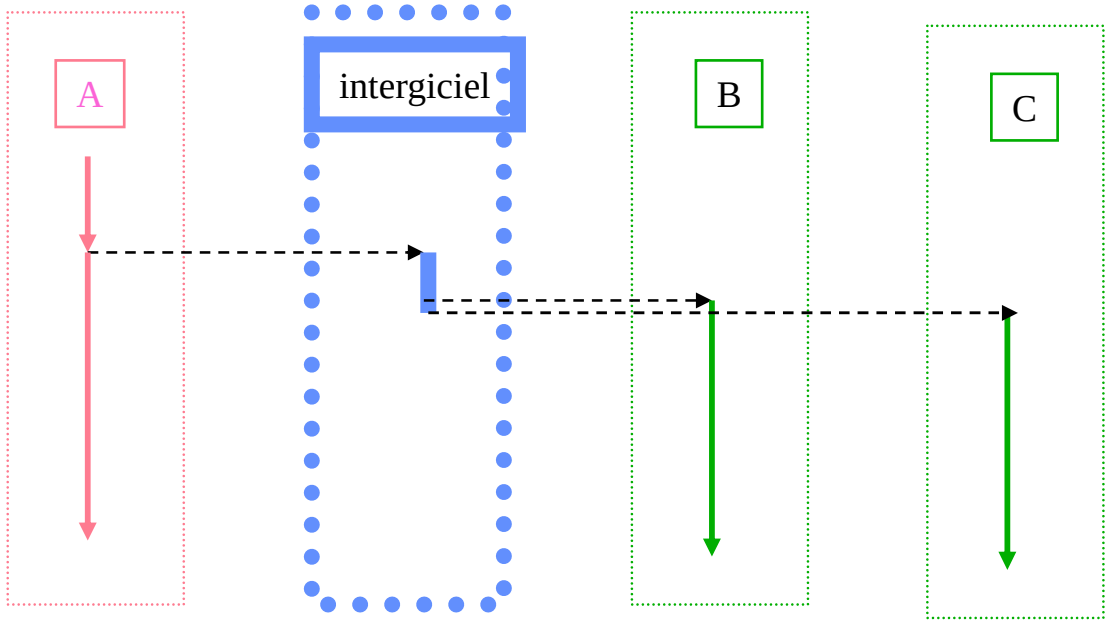
- **Appel synchrone, l 'exemple de RPC**
- **Appel synchrone avec un** *callback* **(*un thread de A*)**





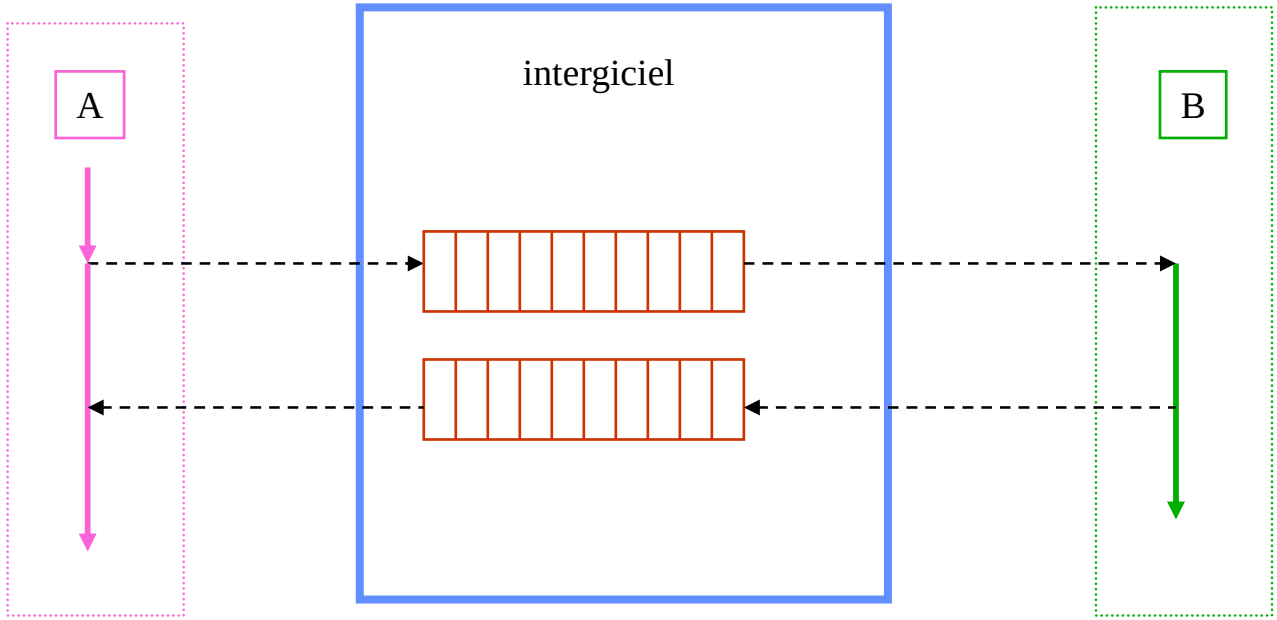
Schémas (2)

- **Appels asynchrones**
  - **événement, notification**



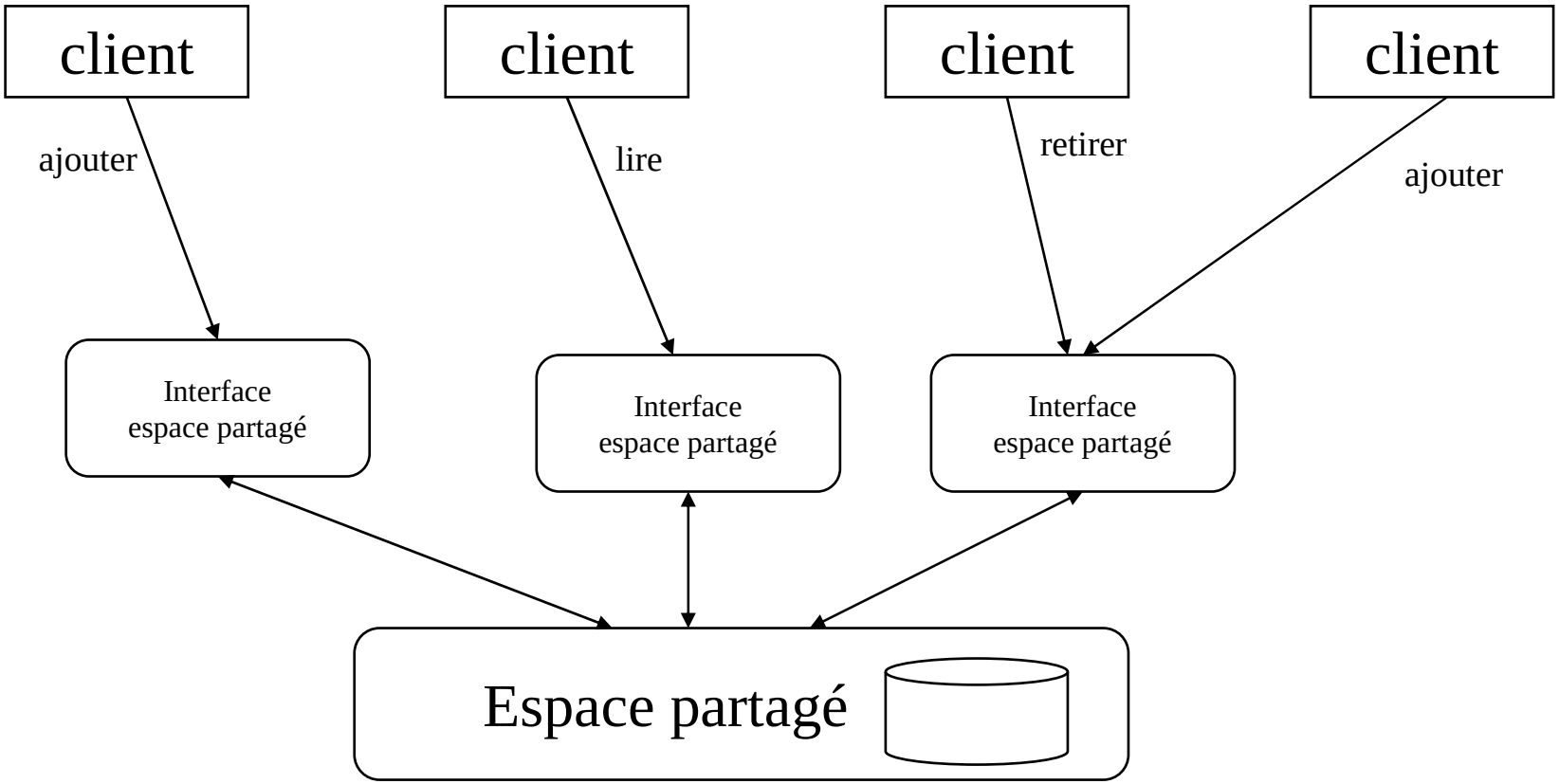
Schémas (3)

- **Asynchrones, avec mémorisation**



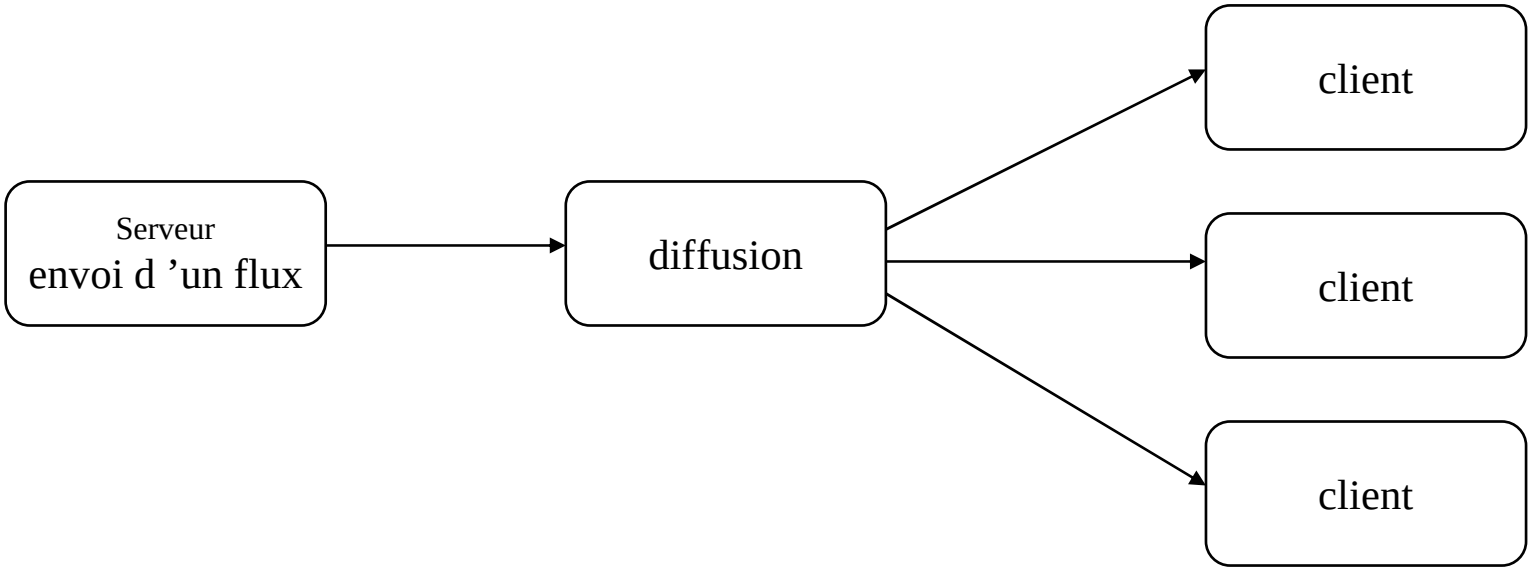
Schémas (4)

- **Coordination par objets partagés**
  - **espace de tuples**
  - **Système de gestion de base de données**
  - **Transactions distribuées**



Schémas (5)

- Flux *Streaming* **(audio/video)**
  - **http://www.inrialpes.fr/planete/people/roca/rhdm02/slides/10Mai_G.Privat_Middleware_Multimedia-RHDM02.pdf**



Patrons/Patterns pour le logiciel

- **Origine C. Alexander un architecte**
  - **1977, un langage de patrons pour l 'architecture 250 patrons**
- **Abstraction dans la conception du logiciel**
  - **[GoF95] la bande des 4 : Gamma, Helm, Johnson et Vlissides**
    - **23 patrons/patterns**
- **Une communauté**
  - **PLoP Pattern Languages of Programs**
    - **http://hillside.net**

Patrons & canevas

- **Patrons**
  - **Définition**
  - **Exemples**
- **Canevas**
  - **Définition**
  - **Exemples**

Patrons

- **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 ")

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

 ICAR'06 19

- **http://www.cmcrossroads.com/bradapp/docs/patterns-intro.html**
- http://www.edlin.org/cs/patterns.html

Patron défini par J. Coplien

- *Un pattern est une **règle** en trois parties exprimant une relation entre un **contexte**, un **problème** et une **solution** (**Alexander**)*

- **Summary by Jim Coplien:**

  Each pattern is a three-part rule, which expresses a relation between a certain context, a certain system of forces which occurs repeatedly in that context, and a certain software configuration which allows these forces to resolve themselves.

Définition d 'un patron

- **Contexte**
- **Problème**
- **Solution**

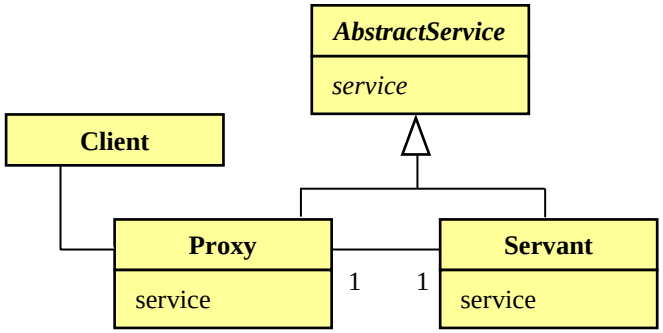
- **Patterns and software :**
  - **Essential Concepts and Terminology par Brad Appleton http://www.cmcrossroads.com/bradapp/docs/patterns-intro.html**

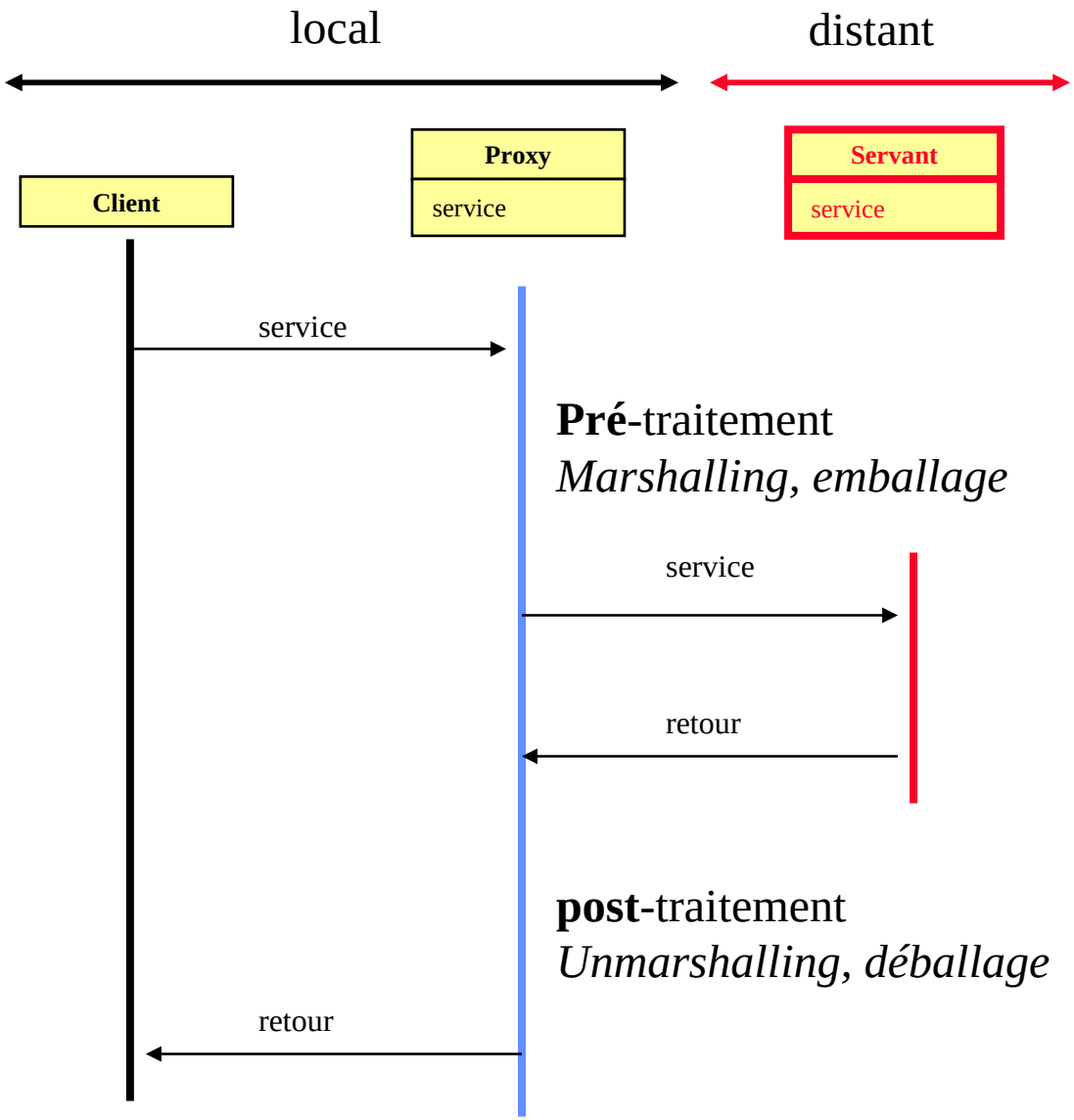
- **Différentes catégories**
  - **Conception (Gof)**
  - **Architecturaux(POSA/GoV, POSA2 [Sch06])**
  - **Organisationnels (Coplien www.ambysoft.com/processPatternsPage.html)**
  - **Pédagogiques(http://www.pedagogicalpatterns.org/)**
  - **....**

Patron Proxy

- **Ou Client Proxy**
  - **Contexte**
    - **un ensemble d ’objets répartis, un client accède à des services distants**
  - **Problème**
    - **Abstraire le client de toute la communication**
      - **du servant comme la transmission des paramètres et du retour**
  - **Solution**
    - **un représentant local du servant est sur le site du client**
    - **ce représentant possède la même interface que le servant**

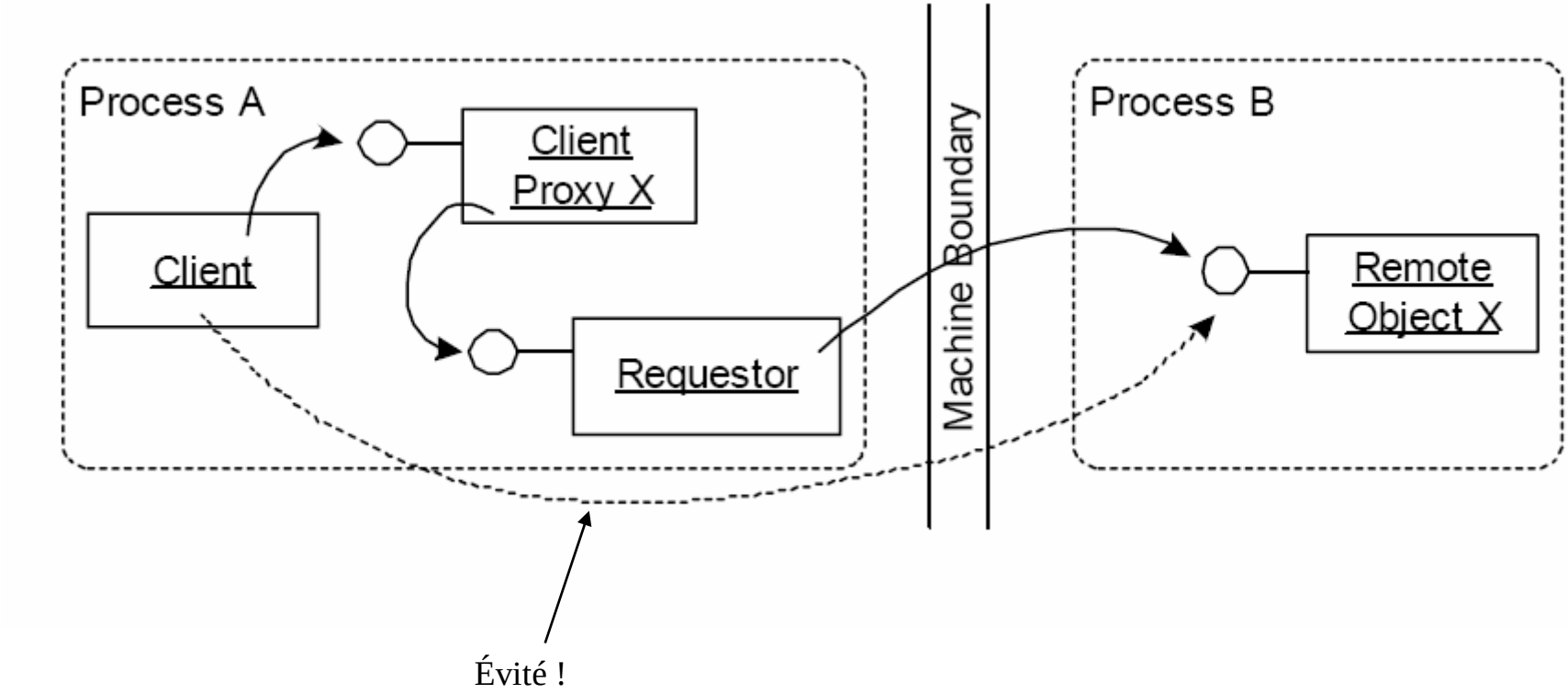
description « UML »





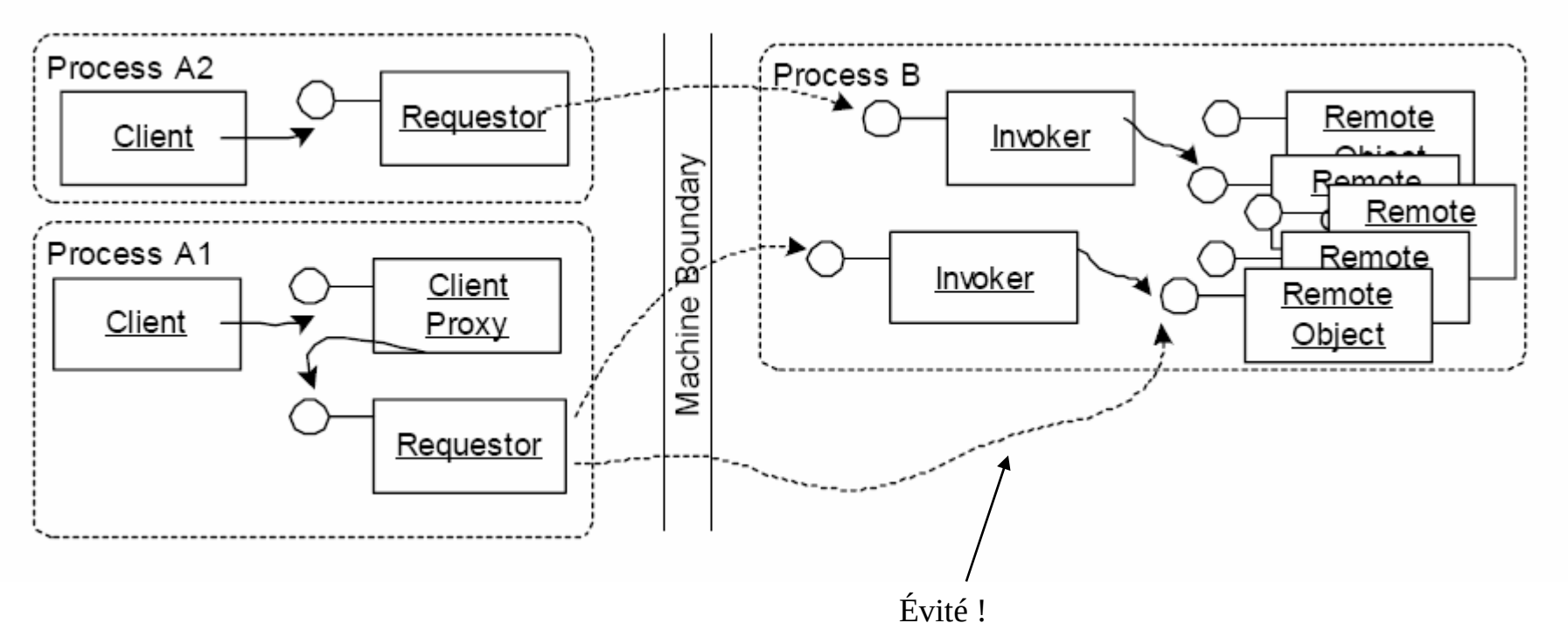
- **« Transparent » pour le client**

Proxy selon [VKZ05]



- **Extrait de http://www.infosys.tuwien.ac.at/Staff/zdun/teaching/evs/evs.pdf**

Invoker côté serveur selon [VKZ05]

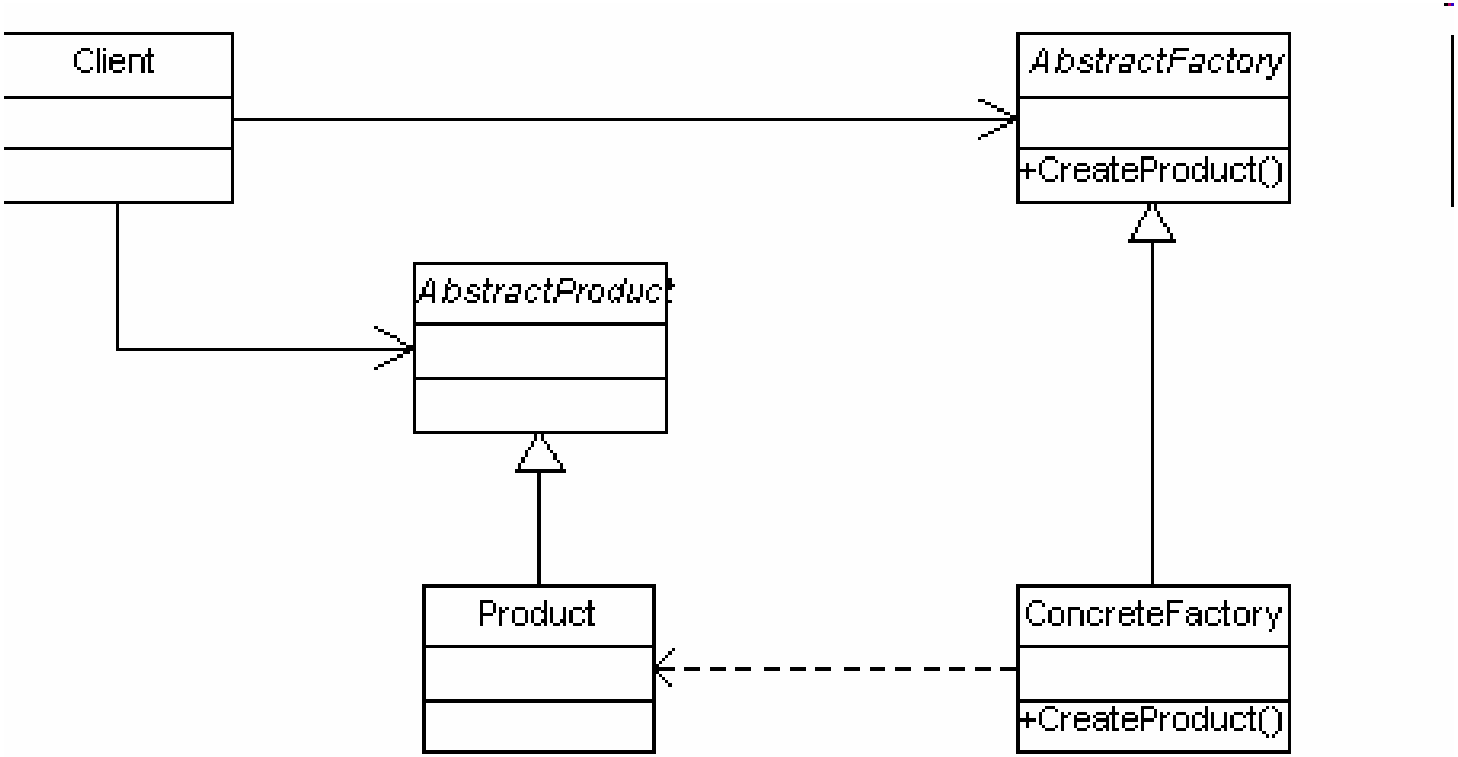


- **Extrait de http://www.infosys.tuwien.ac.at/Staff/zdun/teaching/evs/evs.pdf**

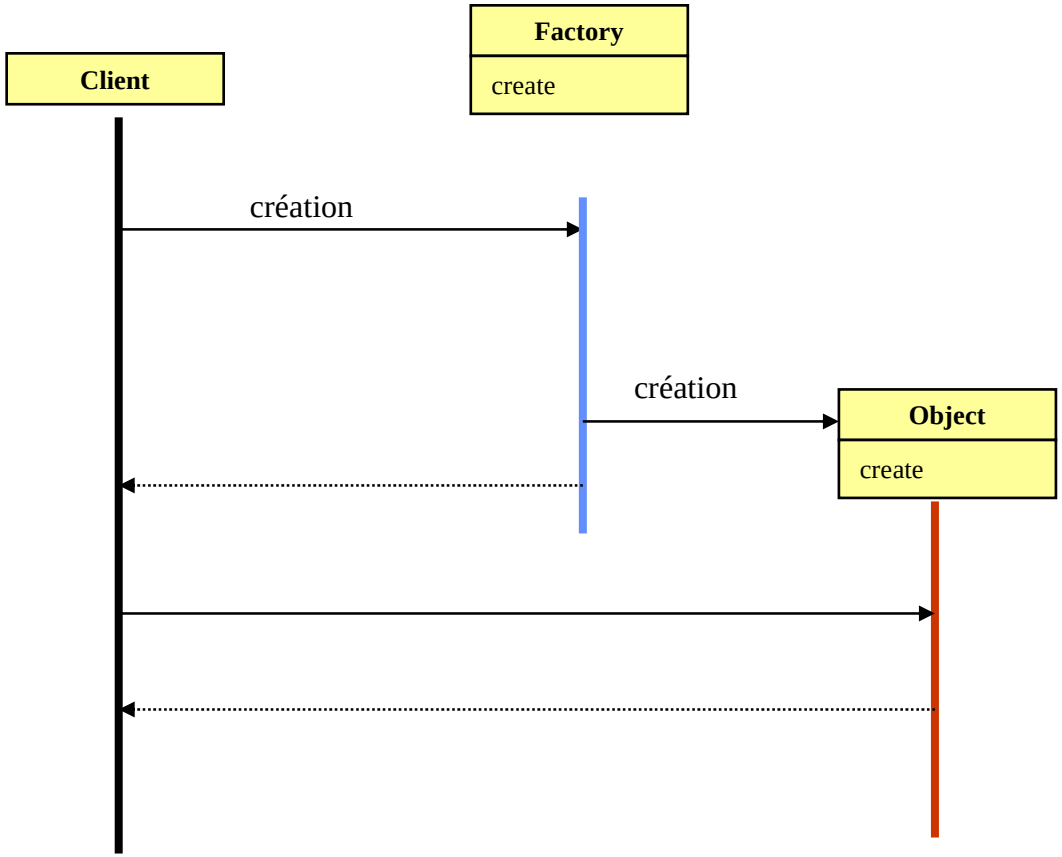
Fabrique/Factory

- **Factory**
  - **Contexte**
    - **un ensemble d ’objets répartis**
  - **Problème**
    - **Créer dynamiquement des instances**
    - **Création d ’instances paramétrée**
    - **Choix effectué à l ’exécution**
  - **Solution**
    - **Abstract Factory : une interface, la création est déléguée à des fabriques concrètes**

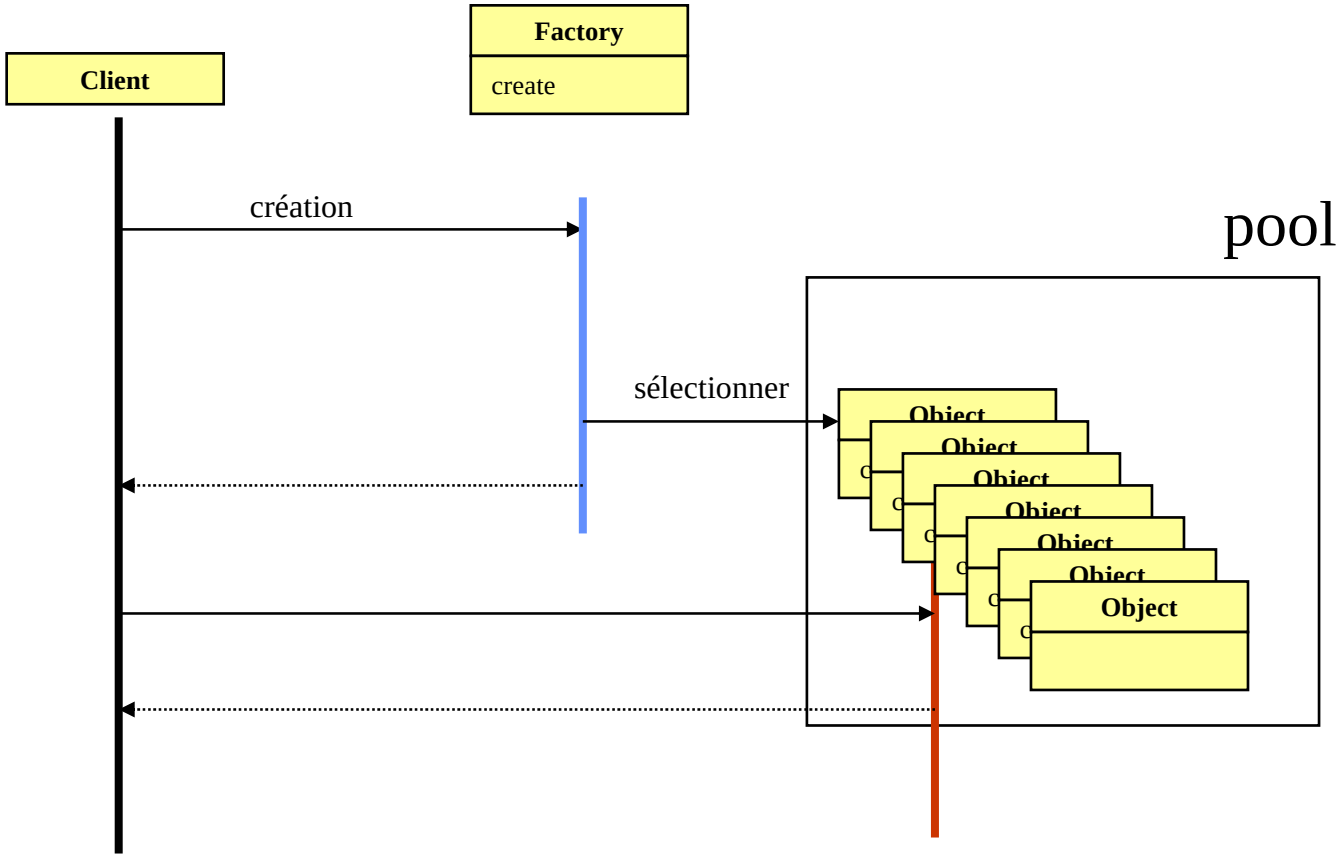
Patron Factory UML l 'original



Fabrique



Fabrique choix d 'un pool

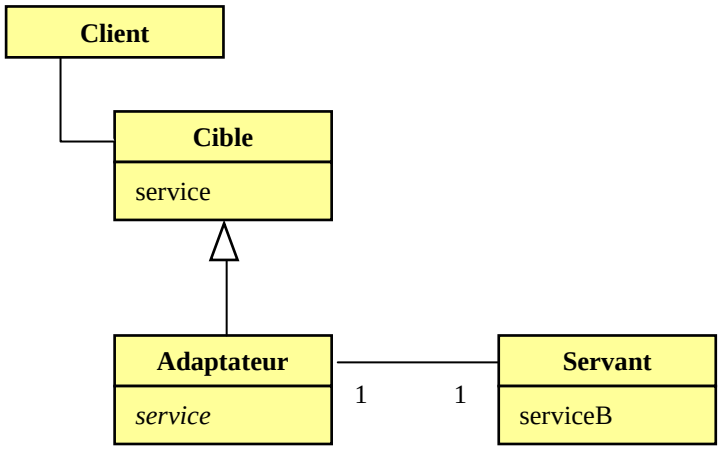


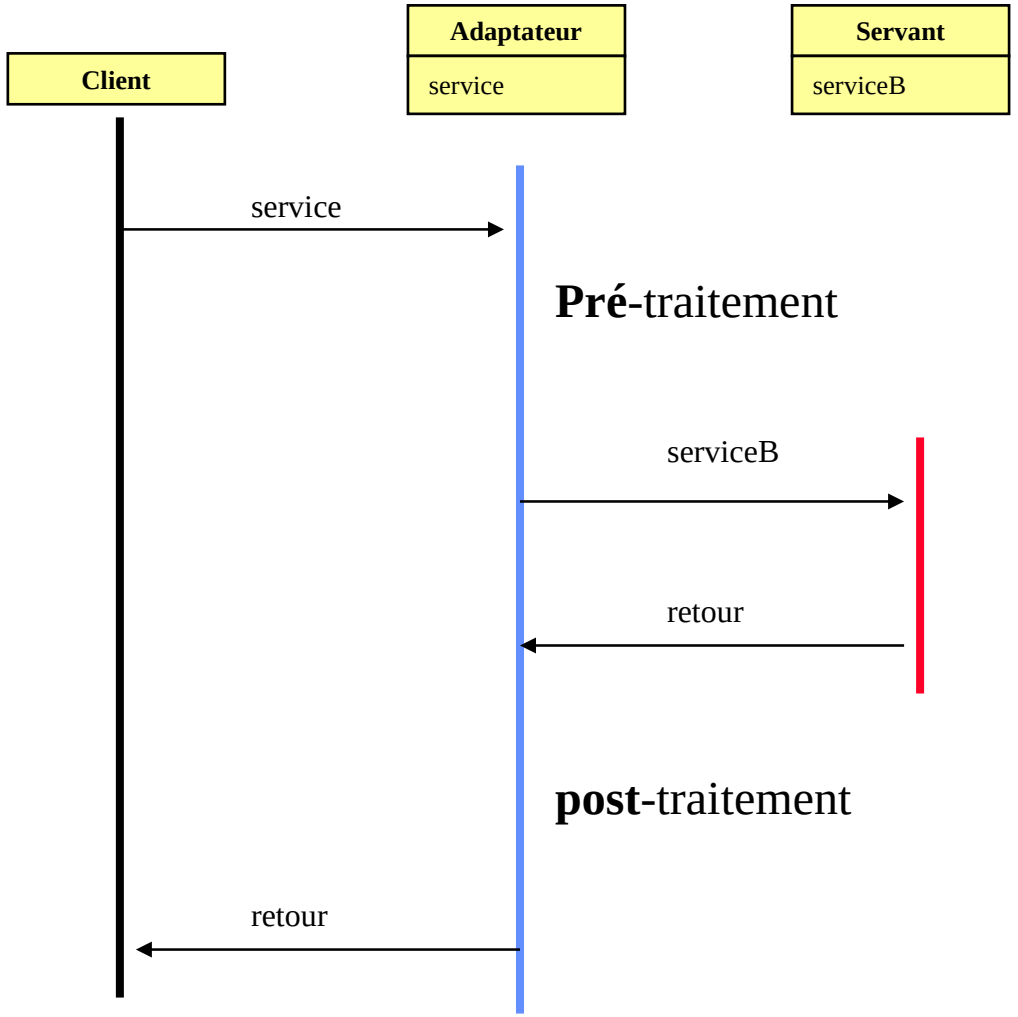
- **Un exemple**

Adaptateur

- **Adaptateur/ Wrapper**
  - **Contexte**
    - **Services requis par le client, services fournis par le servant**
  - **Problème**
    - **Adéquation Services requis / Services fournis**
    - **Réutilisable**
  - **Solution**
    - **Interception des demandes de services destinés au servant**
    - **Mise en conformité des paramètres**

description « UML »





Discussion

- **Adaptateur/Proxy**

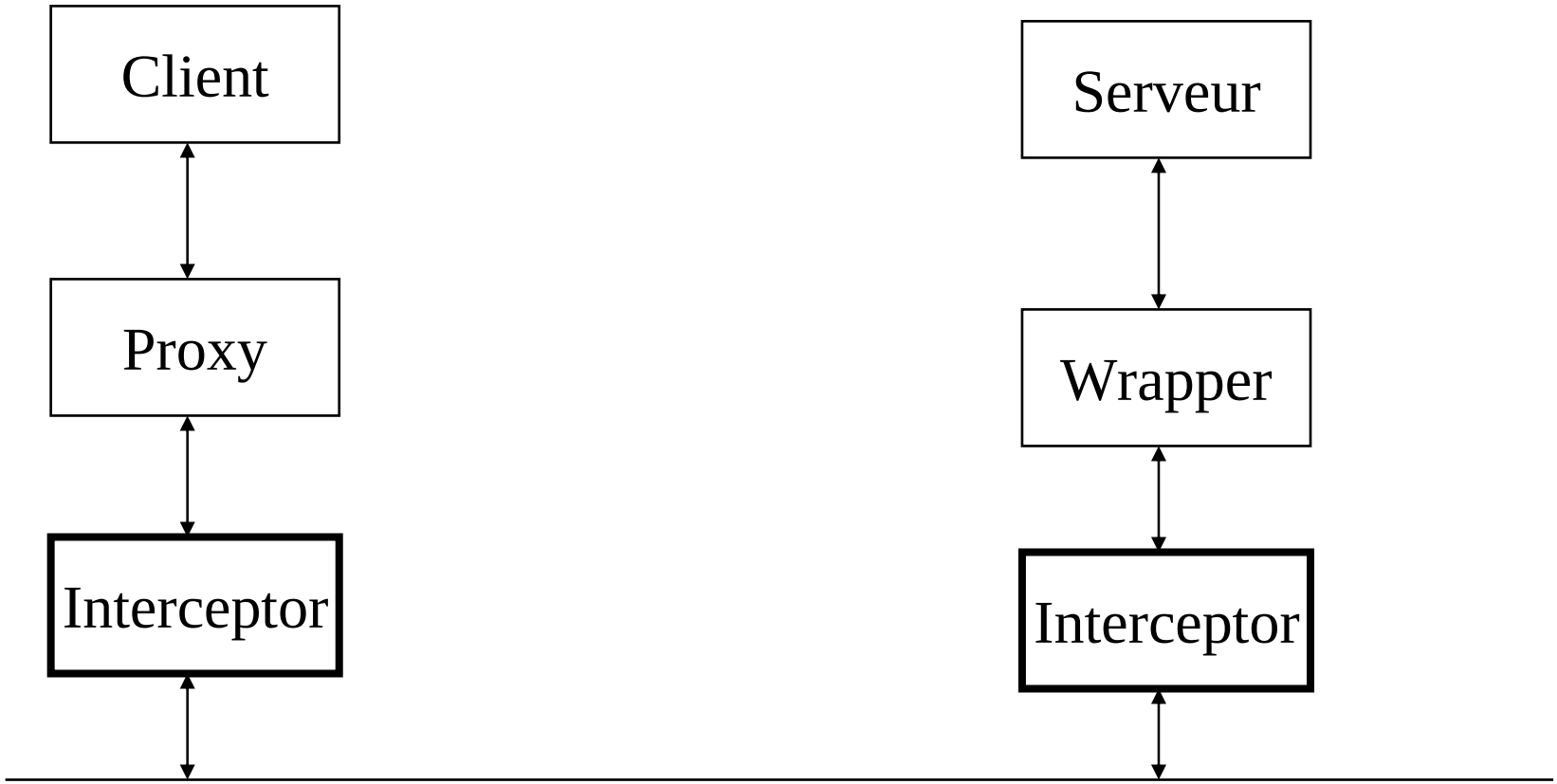
Interceptor

- **Interceptor**
  - **Contexte**
    - **Fourniture de services**
  - **Problème**
    - **Transformer le service**
    - **Client et servant demeurent inchangés**
  - **Solution**
    - **Création dynamique d ’objets d ’interposition**
    - **Interception des appels et des retours**
    - *similaire au patron chaîne de responsabilités cf suite du cours*

Discussion

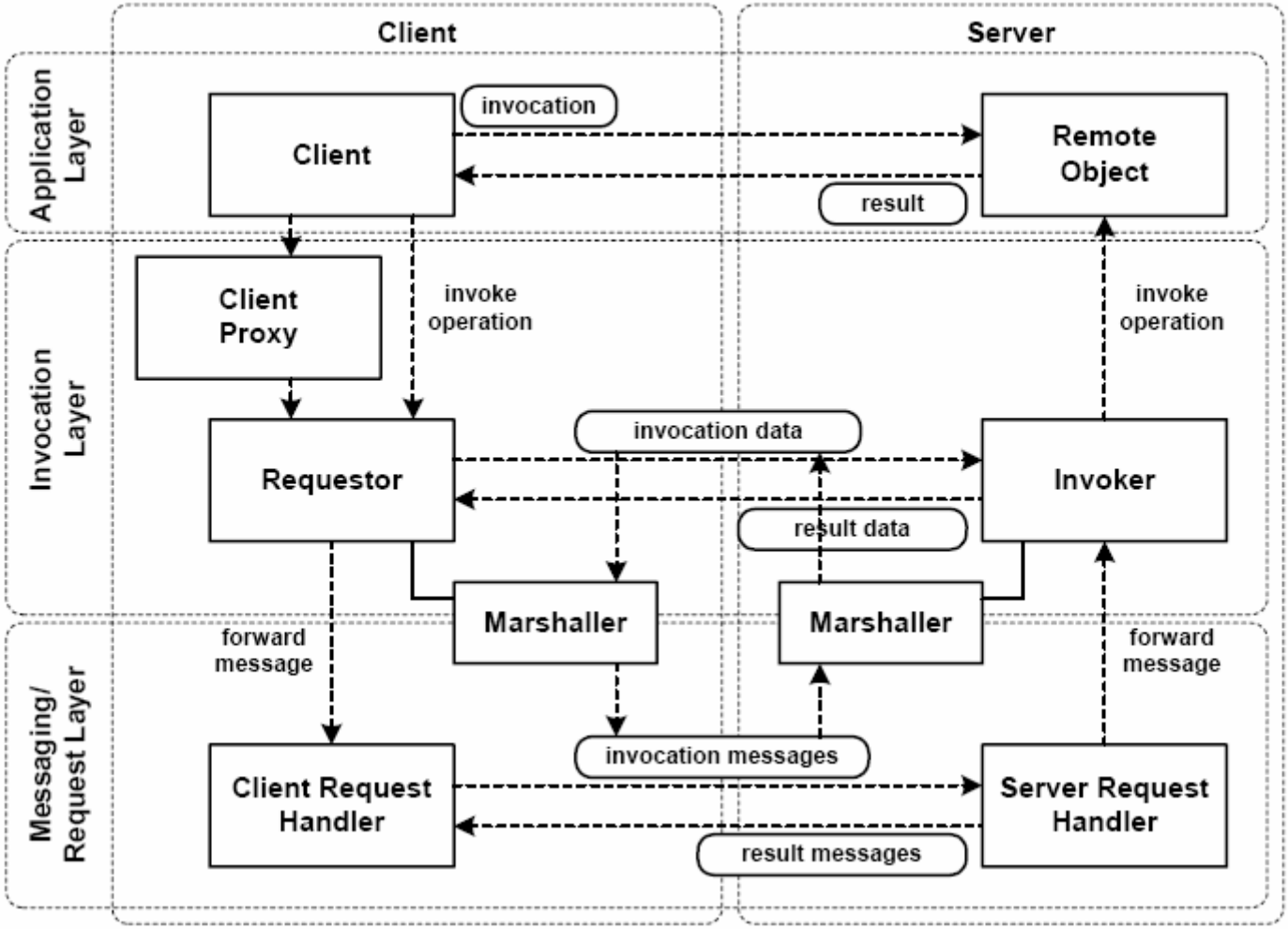
- **Adaptateur/Proxy**
- **Adaptateur/Interceptor**
- **Interceptor/Proxy**

Intergiciel



- **Intergiciel comme « assemblage/composition » de Patrons**

Intergiciel/interaction des Patrons



- **extrait de Uwe Zdun**
  - **http://www.infosys.tuwien.ac.at/Staff/zdun/teaching/evs/evs.pdf**

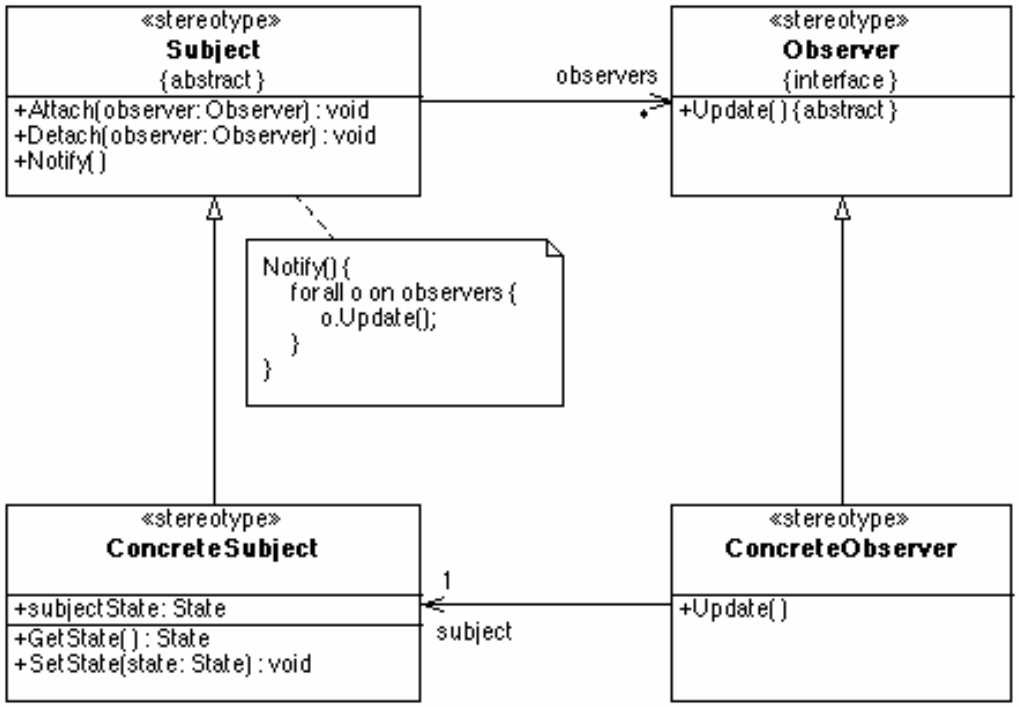
MOM

- **Message Oriented Middleware**
  - **Patron Observer/Publish-Subscribe**
  - **Files de messages persistantes**
  - **Par abonnement**
    - **JMS (Java Messaging Service)**

Pattern Observateur

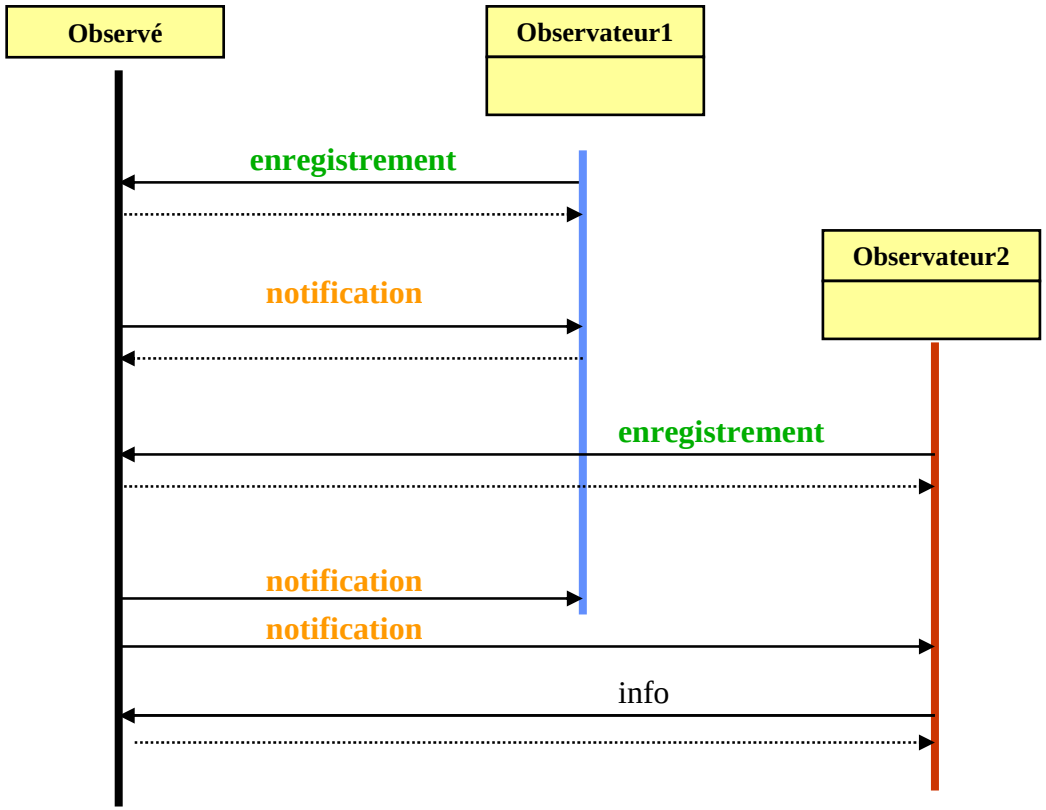
- **Observateur/Observé**
  - **Contexte**
    - **Des objets sont observés par d 'autres**
  - **Problème**
    - **Les observateurs doivent être notifiés par les observés**
    - **Évolution dynamique**
  - **Solution**
    - **Inscription des observateurs auprès des Observés**
    - **Notification synchrone/asynchrone des observateurs**
    - **Filtrage des notifications**

Pattern Observateur



- **Observateur/Observé**

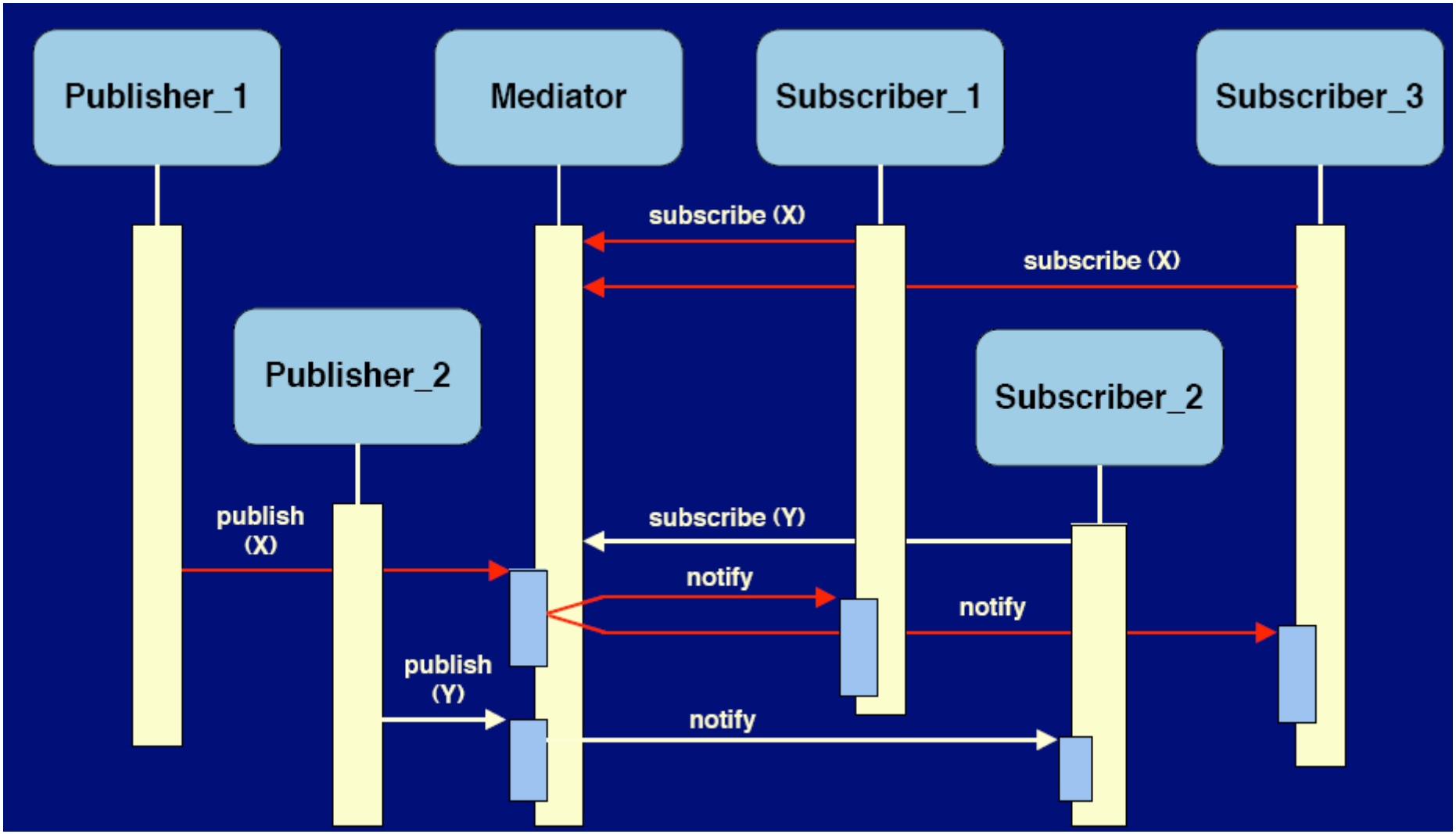
Observateur/Observé



Critiques/limites

- **Observateur/Observé**
  - Les observateurs sont réveillés à chaque changement d ’état de l ’observé
  - Filtrage nécessaire, coté émission
  - mise en œuvre en applications réparties
- **Publish/subscribe**
  - abonnement avec filtrage
- **Espace Partagé**
  - médium de coordination

Publish/subscribe



- **Extrait de [Kra06]**

MOM

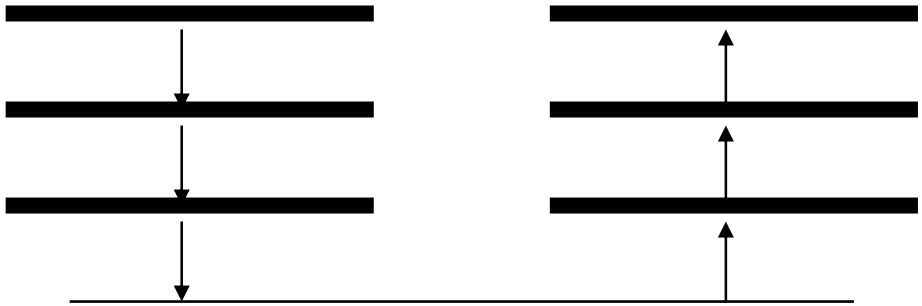
- **IBM WebSphere**
  - http://www.ibm.com/software/ts/mqseries/
- **Sun / spécification JMS**
  - http://www.sun.com/software/products/message_queue/index.xml
- **Microsoft**
  - http://www.microsoft.com/windowsserver2003/technologies/msmq/default.mspx
- **ObjectWeb (Open Source)**
  - http://www.objectweb.org/

Patrons Architecturaux

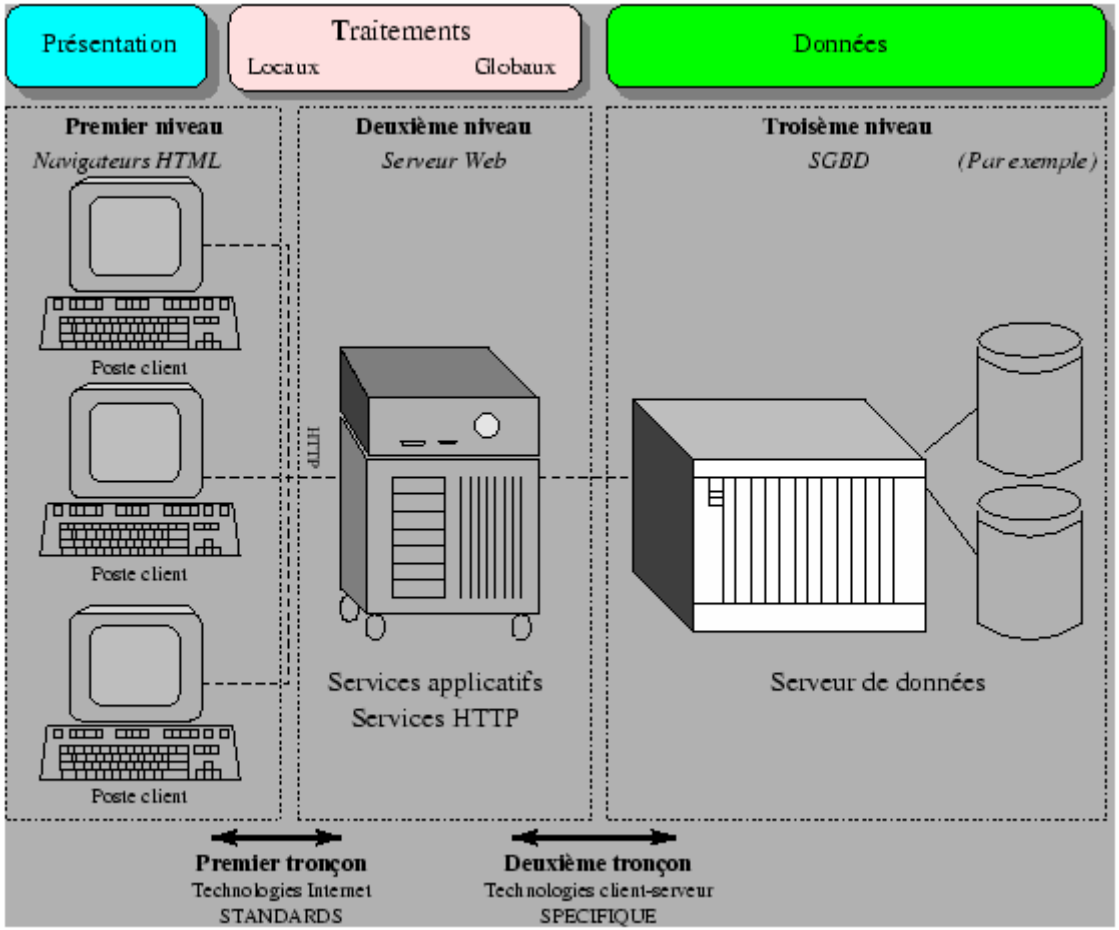
- **Architectures en couches**
- **Architectures « n-tier », multi-étages**
- **Canevas**

En couches

- **Hiérarchique**
  - **Chaque couche est vue comme une machine abstraite**
  - **Les instructions sont définies par leur interface**



Multi-niveaux (n-tiers)



- **Plusieurs niveaux**
  - **Présentation | Traitement | Données**
  - **http://remi.leblond.free.fr/probatoire/probatoire.html**

A typical…selon microsoft

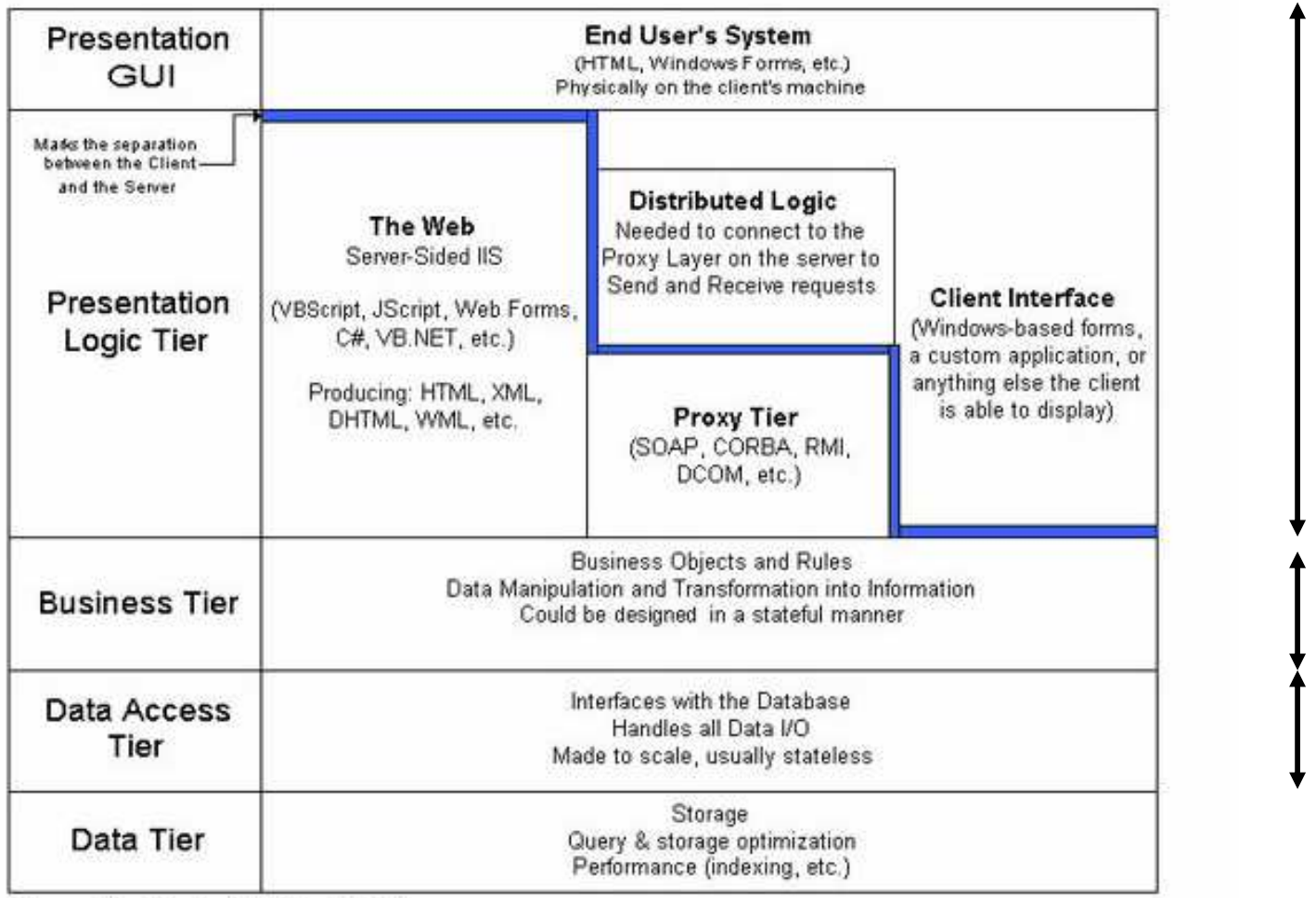


Figure 1.1 A Typical N-Tier Model

- **Extrait de http://www.15seconds.com/issue/011023.htm**

Canevas/ Framework

- **Définition**
  - Un canevas est un “squelette” de programme qui peut être réutilisé (et adapté) pour une famille d’applications
  - Il met en œuvre 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

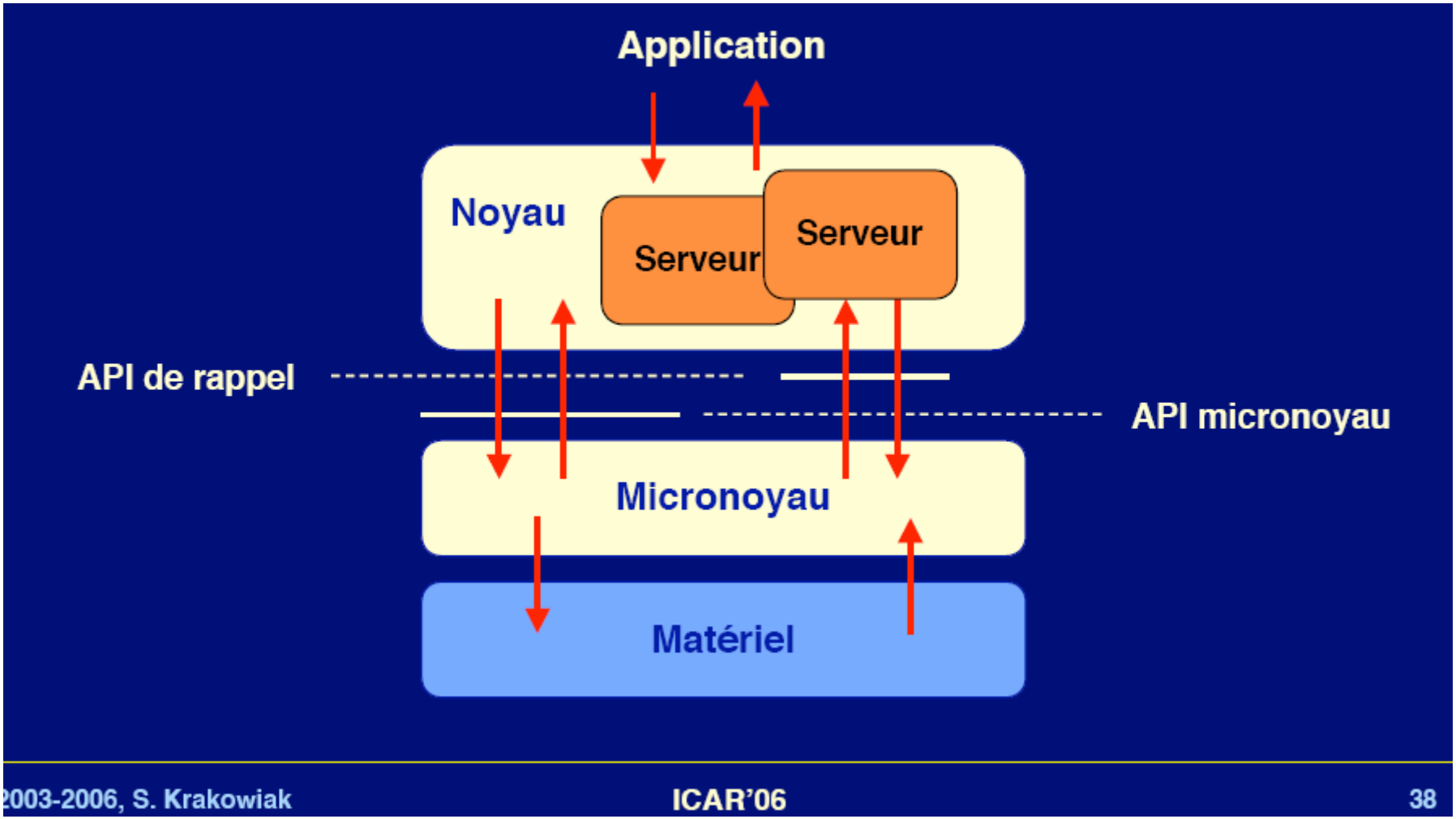
- **Extrait de [Kra06]**

Canevas

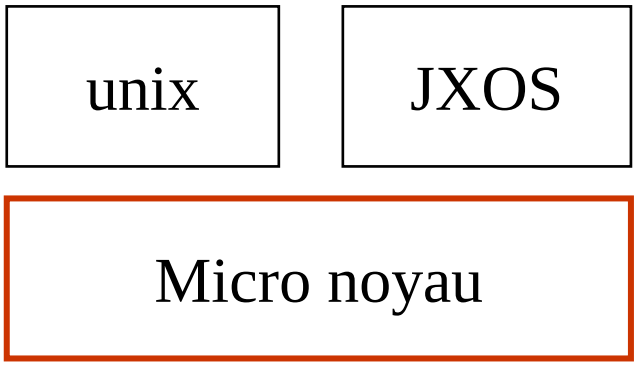
- **Contexte**
  - Applications réparties
- **Problème**
  - Mises à jour
  - Déploiement
  - Cohérence
  - réutilisation
- **Solution**
  - Un squelette de programme
  - Architecture de classes (dans un langage à objets)
    - adaptée aux contraintes de l ’environnement
  - Règles d ’usage de ces classes

Canevas global [Kra06], années 80

- **à l ’aide d ’un micro noyau**

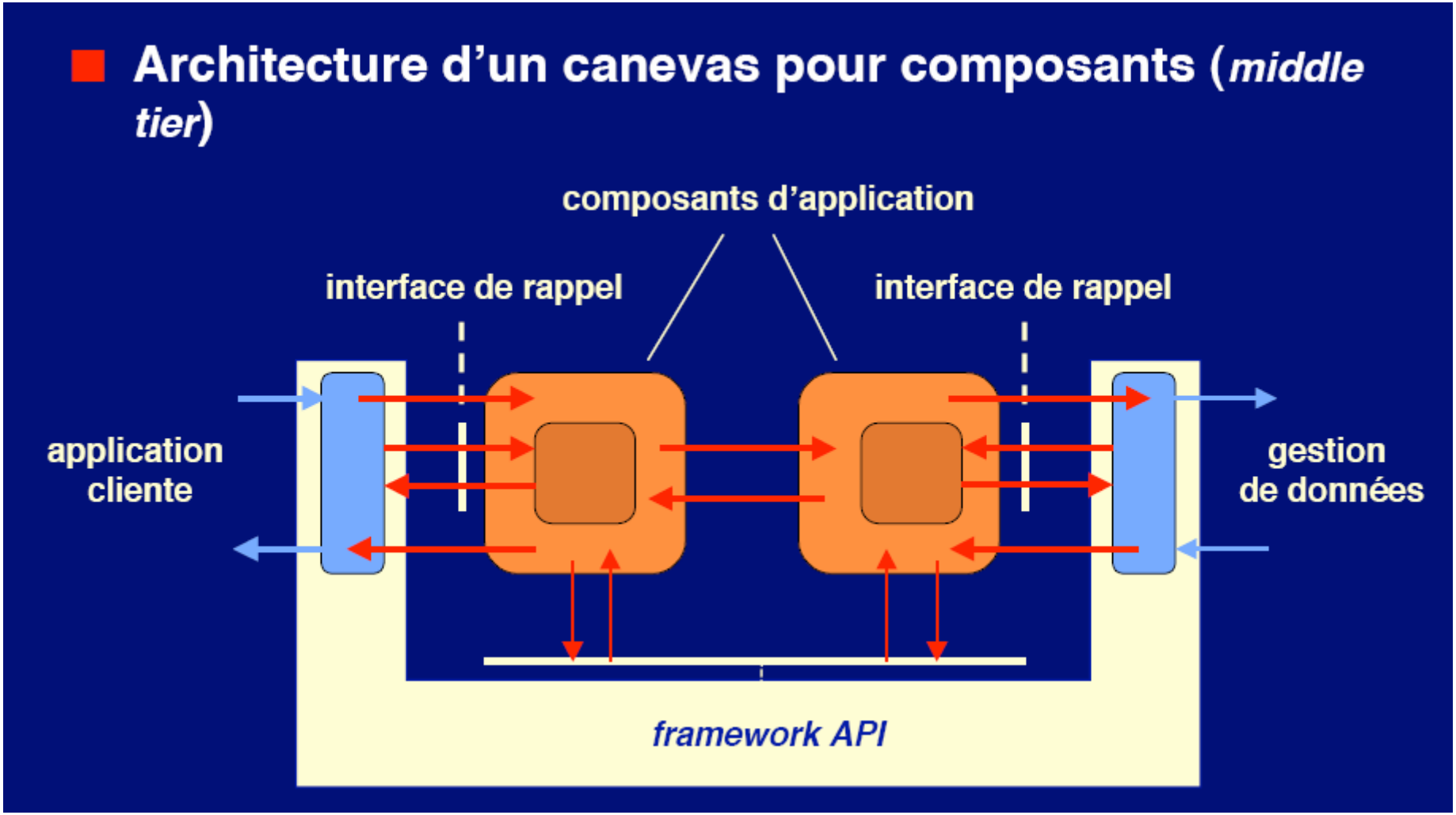


Exemples



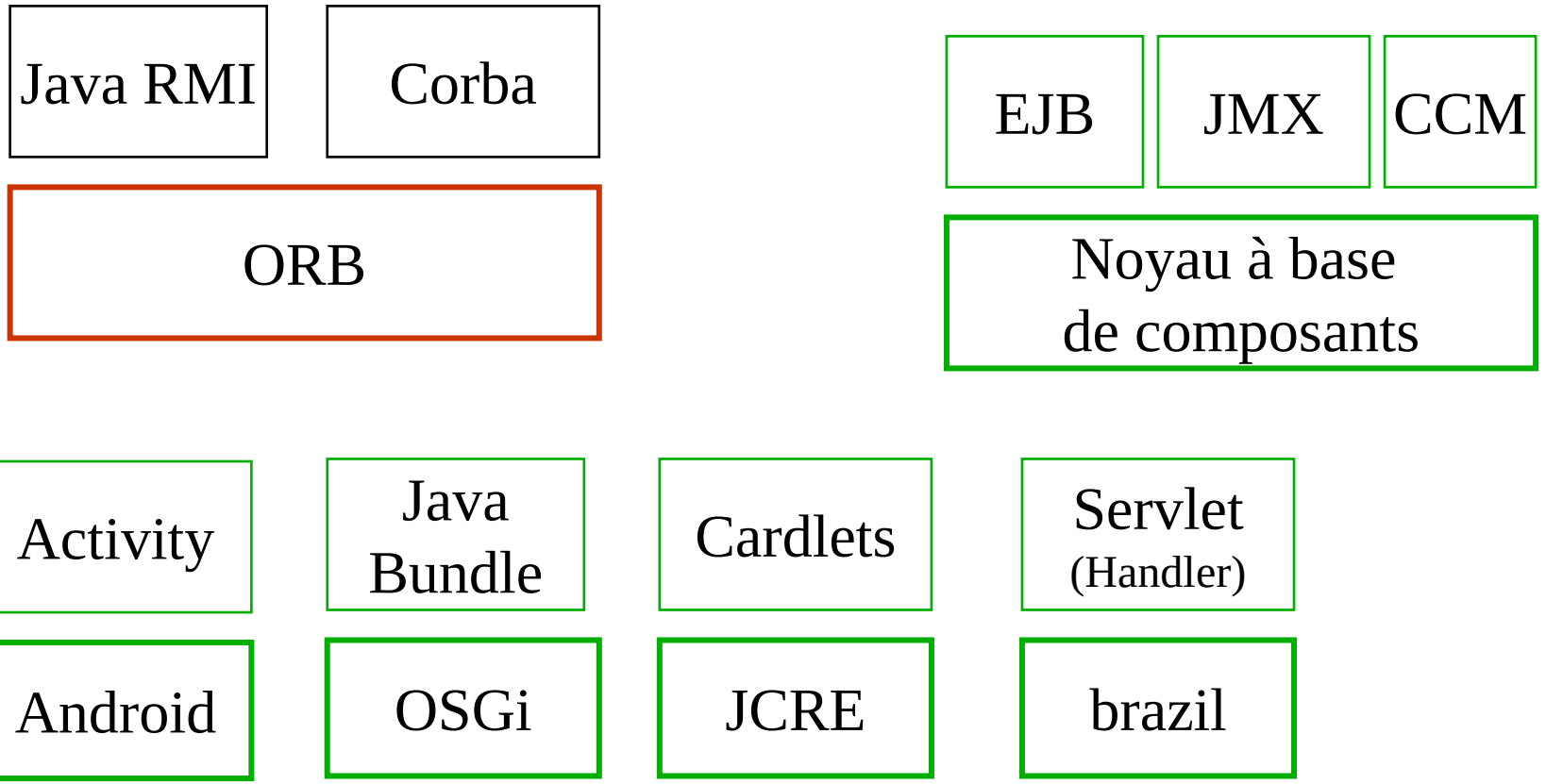
- **Le micro noyau prend en charge les ressources matérielles**
- **http://jxos.free.fr** *(javaOS utilisé en NSY208_209 en 2005)*

à base de composants



- **Extrait de [Kra06]**

Exemples de canevas



- **ORB Object Request Broker**
  JCRE Java Card Run Time,
  OSGi Open Services Gateway initiative
  brazil un serveur web de sun www.experimentalstuff.com
  CCM Corba Component Model
  EJB Enterprise Java Bean JMX Java Management eXtension

NSY102

- **OOM Object Oriented Middleware**
  - **Java RMI,** *(sous-ensemble de CORBA)*
- **Architecture à base de composants**
  - **JMX, EJB, .NET**
- **MOM Message Oriented Middleware**
  - **OpenJMS Java Messaging Service**
- **Espace partagé**
  - **JINI/JavaSpaces**

Conclusion

- **Architecture**
- **Patrons**
- **Canevas**
- **Tendances**

- **NSY102**
  - **Objectifs :** appréhender les concepts et outils permettant la mise en oeuvre d'applications et de systèmes distribués sur le Web. Apprendre à identifier et utiliser les " patrons " (design patterns) adaptés à une situation de conception, ou pour la définition d'une architecture. Construire le canevas (" Framework ") adapté à un type d'architecture système