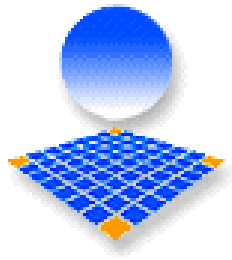
Quelques mécanismes à objets de base pour les intergiciels

Clémentine Nebut

LIRMM

Clementine.nebut@lirmm.fr



Patrons de conception

Transparents inspirés de :

http://www2.lifl.fr/icar/Chapters/Patterns/patterns.html

Proxy

- Contexte.
  - ensemble d'objets dans un environnement réparti ; communication via d'appels de méthode à distance : un client demande un service fourni par un objet éventuellement distant (le servant).
- Problème.
  - Définir un mécanisme d'accès qui :
    - n'implique pas de coder «en dur» l'emplacement du servant dans le code client,
    - ne nécessite pas une connaissance détaillée des protocoles de communication par le client.

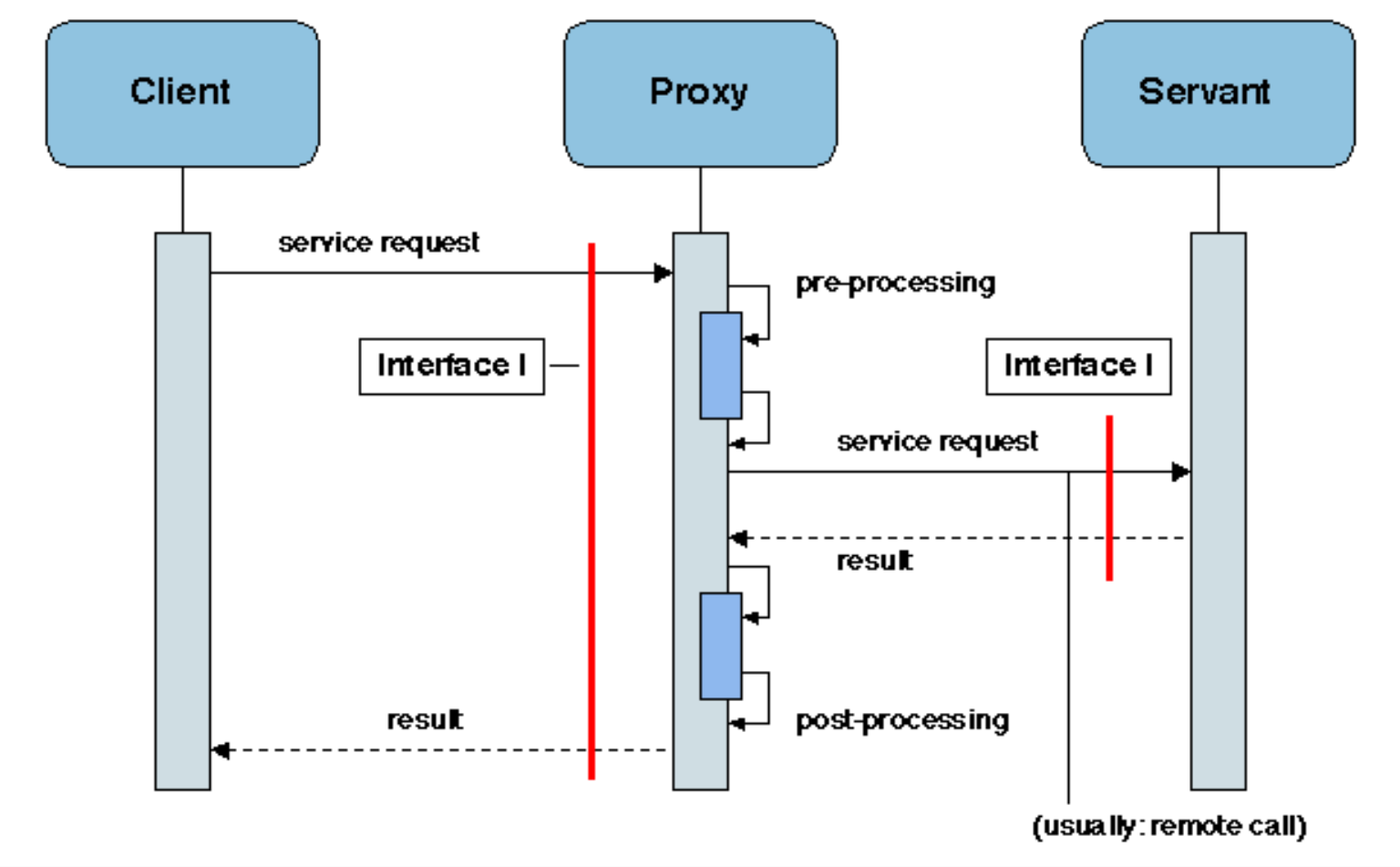
Proxy

- Propriétés souhaitées.
  - Accès efficace à l'exécution.
  - Programmation simple pour le client, idéalement transparence d'accès.
- Contraintes.
  - le client et le serveur sont dans des espaces d'adressage différents.

Proxy

- Solution.
  - Utiliser un représentant local du serveur sur le site du client, qui a exactement la même interface que le servant.
  - L'information relative au système de communication et à la localisation du servant est cachée dans le mandataire (invisible au client).

Proxy


Client
Proxy
Servant
service request
Interface I
pre-processing
Interface I
service request
result
result
post-processing
(usually: remote call)

Factory

- Contexte.
  - ensemble d'objets dans un environnement réparti (la notion d'objet dans ce contexte peut être très générale, et n'est pas limitée au domaine strict de la programmation par objets).
- Problème.
  - pouvoir créer dynamiquement des familles d'objets apparentés (par ex. des instances d'une même classe),
  - en permettant de reporter certaines décisions jusqu'à la phase d'exécution (par exemple le choix d'une classe concrète pour réaliser une interface donnée).

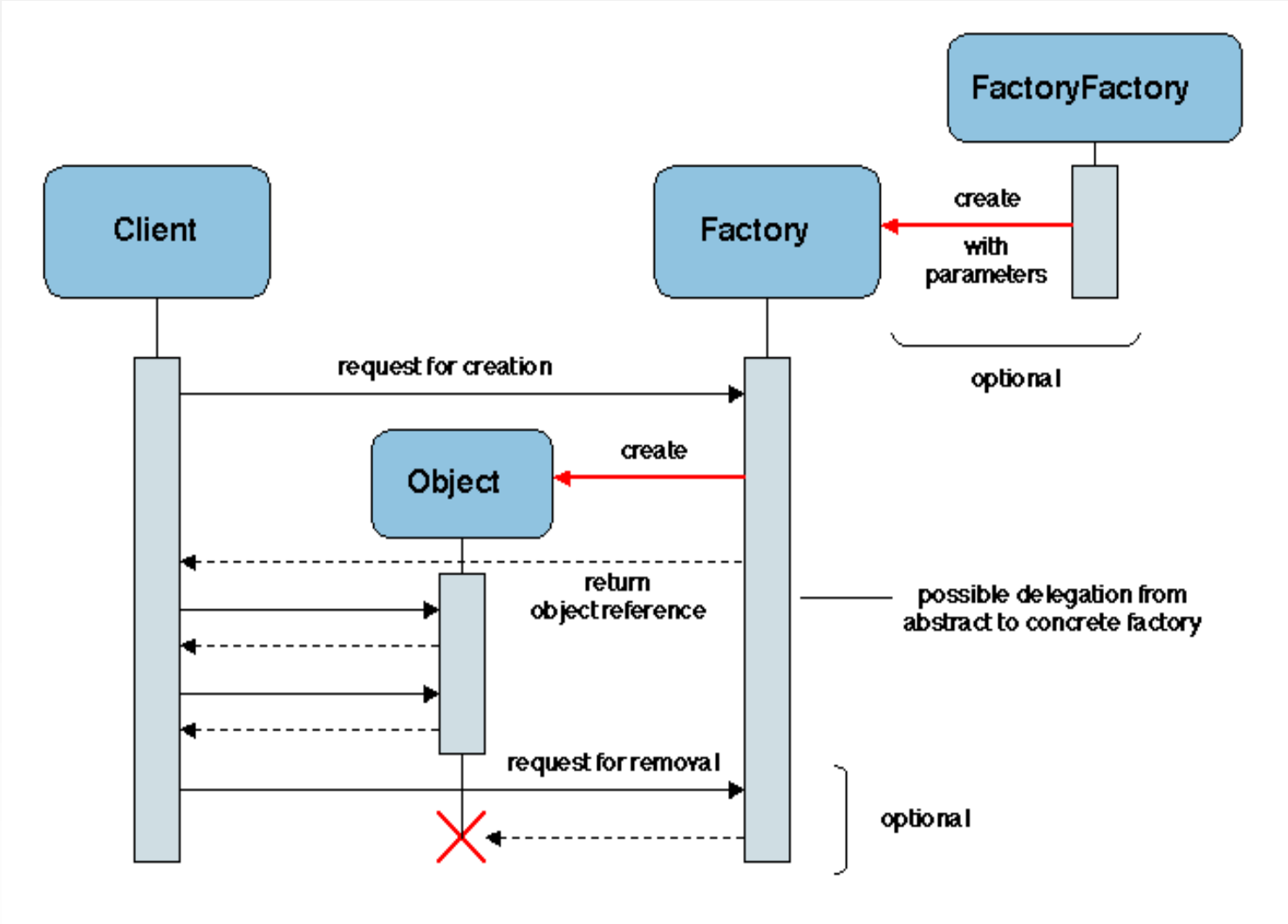
Factory

- Propriétés souhaitées.
  - détails de réalisation des objets créés invisibles.
  - processus de création paramétrable.
  - L'évolution du mécanisme doit être facilitée (pas de décision «en dur»).
- Contraintes.
  - le client (qui demande la création de l'objet) et le serveur (qui crée effectivement l'objet) sont dans des espaces d'adressage différents.

Factory

- Solution.
  - Abstract factory. Définit une interface et une organisation génériques pour la création d'objets ; la création est déléguée à des usines concrètes.
  - Abstract factory peut être réalisé en utilisant Factory methods (une méthode de création redéfinie dans une sous-classe).
  - Amélioration : usine de fabrication d'usines.
  - Une usine peut aussi être utilisée comme un gestionnaire des objets qu'elle a créés (localiser un objet (en renvoyant une référence pour cet objet), détruire un objet sur demande)

Factory


FactoryFactory
Client
Factory
create
with
parameters
optional
request for creation
Object
create
return
object reference
possible delegation from
abstract to concrete factory
request for removal
optional

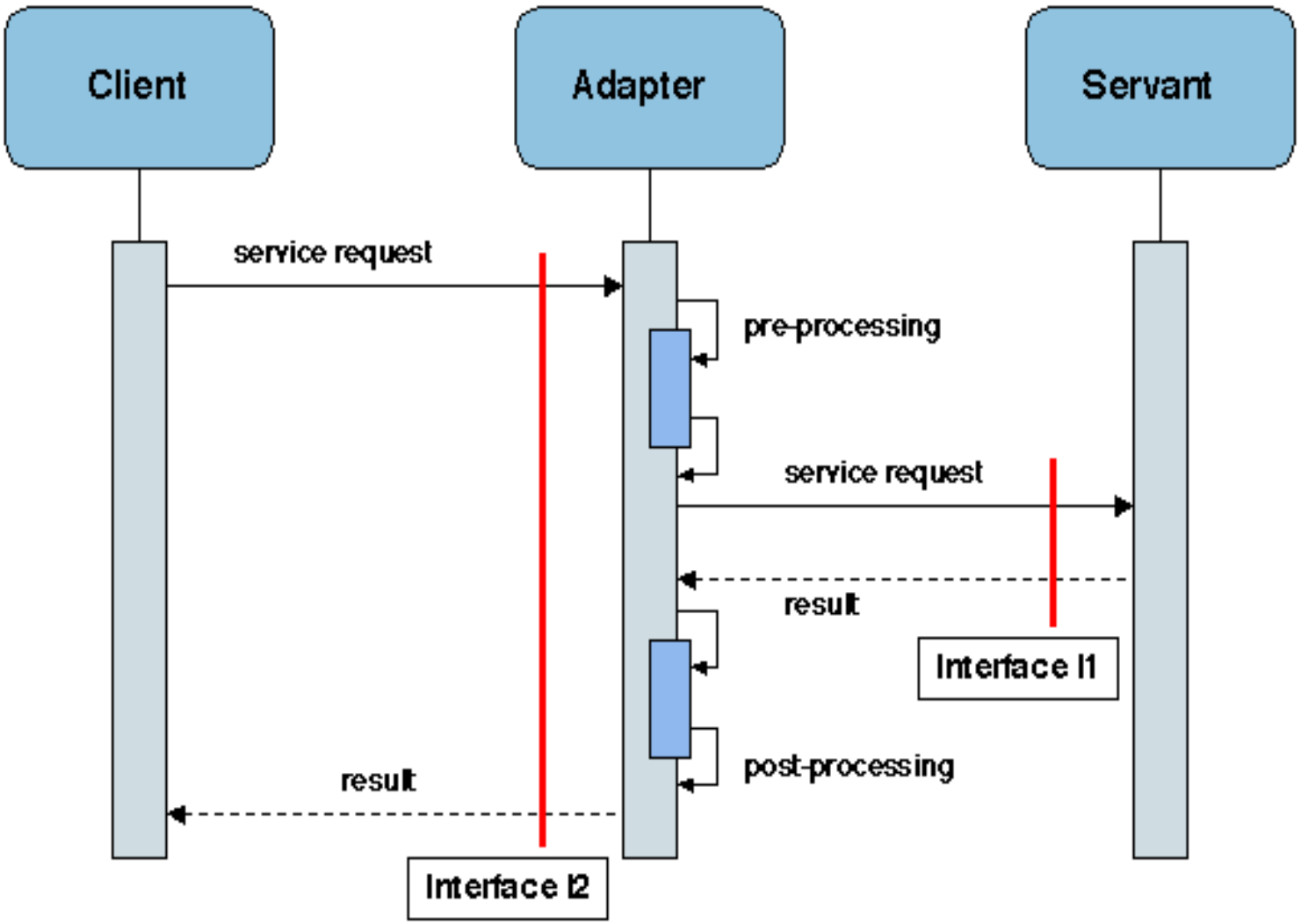
Adapter

- Contexte.
  - un service est défini par une interface ; les clients demandent des services ; des servants, situés sur des serveurs distants, fournissent des services.
- Problème.
  - On souhaite réutiliser un servant existant en le dotant d'une nouvelle interface conforme à celle attendue par un client (ou une classe de clients).

Adapter

- Propriétés souhaitées.
  - Le mécanisme de conversion d'interface doit être efficace à l'exécution. Il doit aussi être facilement adaptable, pour répondre à des changements imprévus des besoins. Il doit être réutilisable (c'est-à-dire générique).
- Solution.
  - Fournir un composant (l'adaptateur, ou wrapper) qui isole le servant en interceptant les appels de méthode à son interface. Les paramètres et résultats peuvent nécessiter une conversion.

Adapter


Client
Adapter
Servant
service request
pre-processing
service request
result
Interface I1
post-processing
result
Interface I2

Interceptor

- Contexte.
  - un service est défini par une interface ; les clients demandent des services ; les servants, situés sur des serveurs distants, fournissent des services. Il n'y a pas de restrictions sur la forme de la communication (uni- or bi-directionnelle, synchrone ou asynchrone, etc.).
- Problème.
  - On veut ajouter de nouvelles capacités à un service existant, ou fournir le service par un moyen différent.

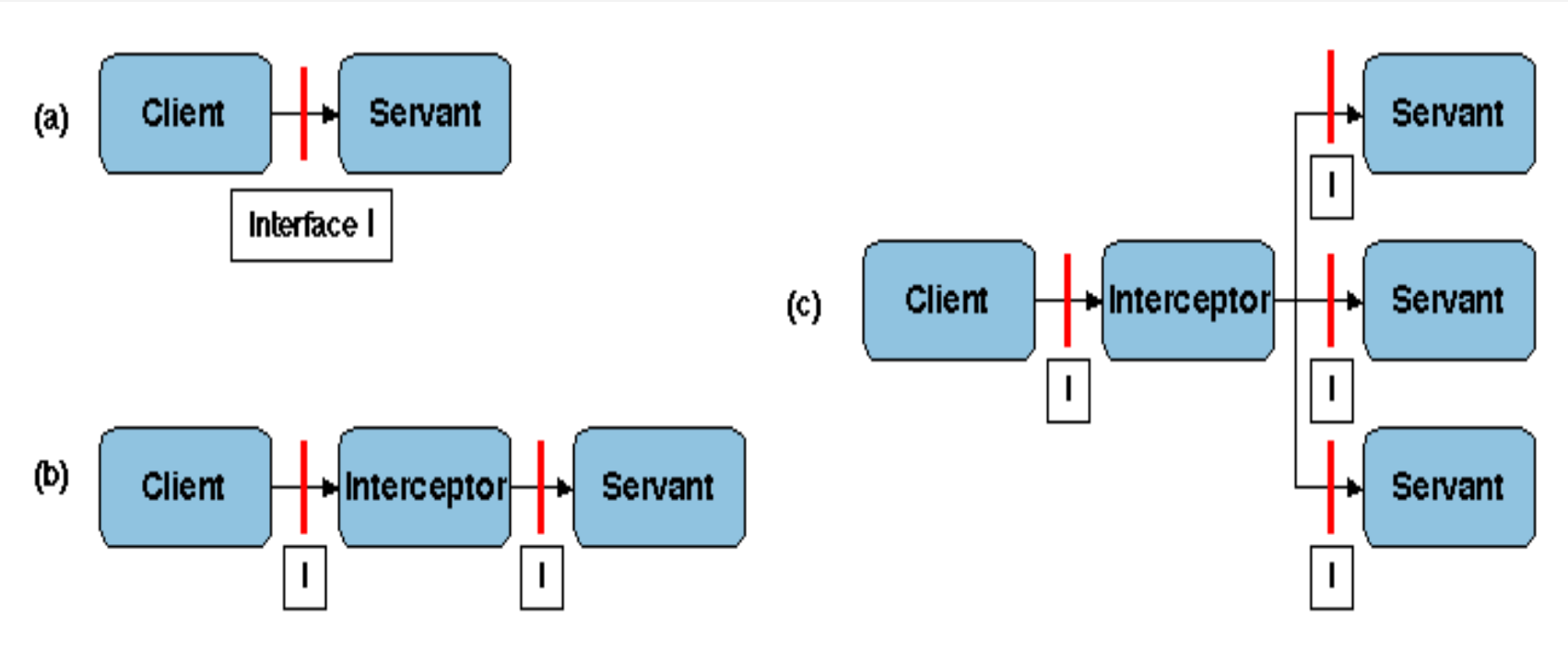
Interceptor

- Propriétés souhaitées.
  - Le mécanisme doit être générique (applicable à une large variété de situations). Il doit permettre de modifier un service aussi bien statiquement (à la compilation) que dynamiquement (à l'exécution).
- Contraintes.
  - Les services peuvent être ajoutés ou supprimés dynamiquement.

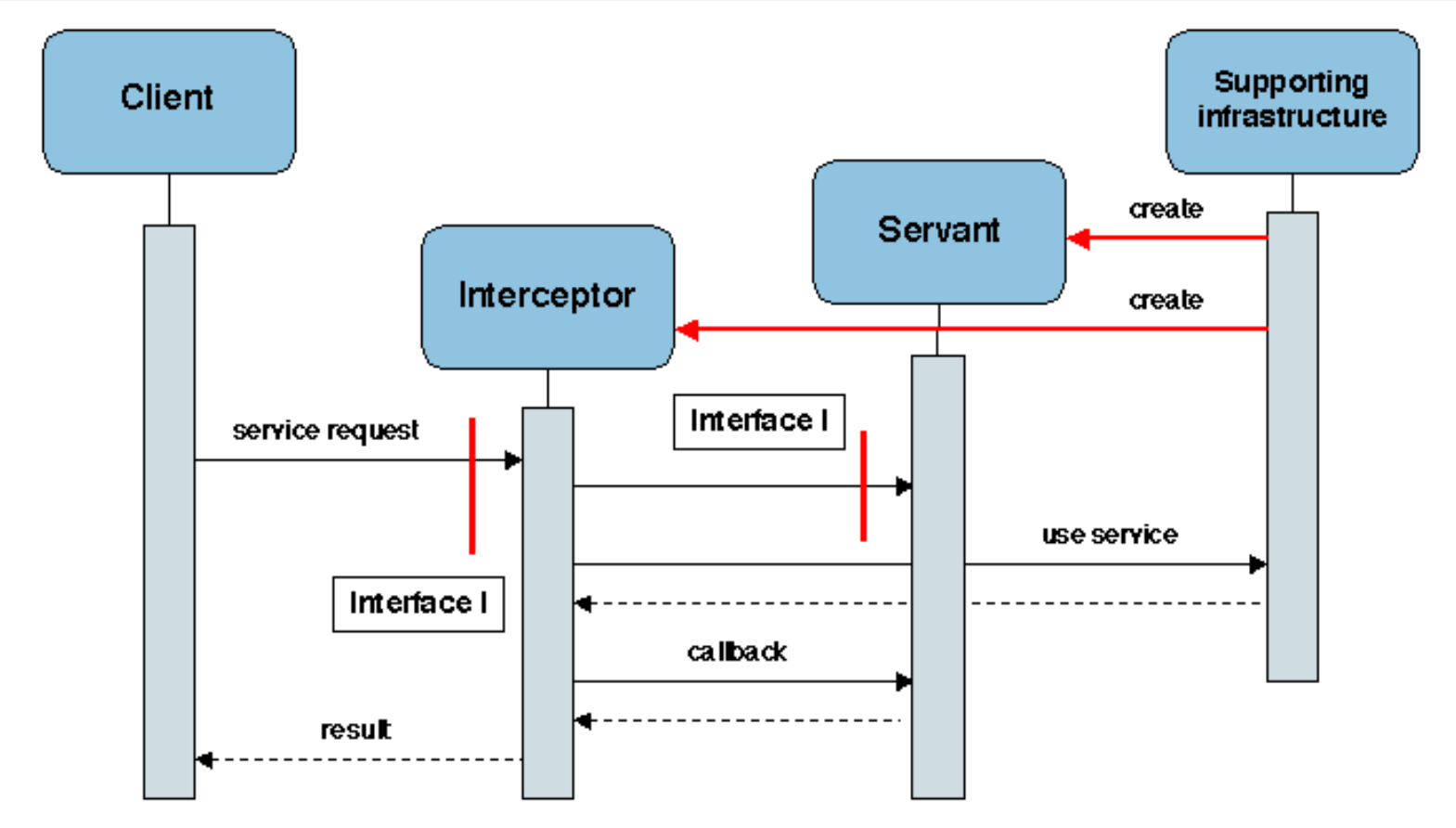
Interceptor

- Solution.
  - Créer (statiquement ou dynamiquement) des objets d'interposition, ou intercepteurs.
  - Ces objets interceptent les appels (et/ou les retours) et insérent un traitement spécifique, qui peut être fondé sur une analyse du contenu.
  - Un intercepteur peut aussi rediriger un appel vers une cible différente.

Interceptor



Interceptor



Exemple dans un ORB (Object Request Broker)