

NOTERE'2005

Techniques d'aspect pour
la gestion de la mémoire répartie
dans un environnement CORBA/C++

Toni SOUEID, Nesrine YAHIAOUI,
Lionel SEINTURIER, Bruno TRAVERSON

1^{er} Septembre 2005



Plan

- ☞ Contexte et objectifs
- ☞ Programmation orientée aspect
- ☞ Cycle de vie dans CORBA
- ☞ Implantation du cycle de vie
- ☞ Application des aspects
- ☞ Bilan et perspectives

Contexte et objectifs

Contexte

- Dans une application scientifique, des milliers d'objets CORBA peuvent être créés, utilisés et partagés, puis détruits au cours d'une simulation.
⇒ La gestion du cycle de vie (*Life Cycle*) de ces objets devient indispensable.
- La programmation orientée aspect est une approche récente permettant la séparation des préoccupations techniques et fonctionnelles.

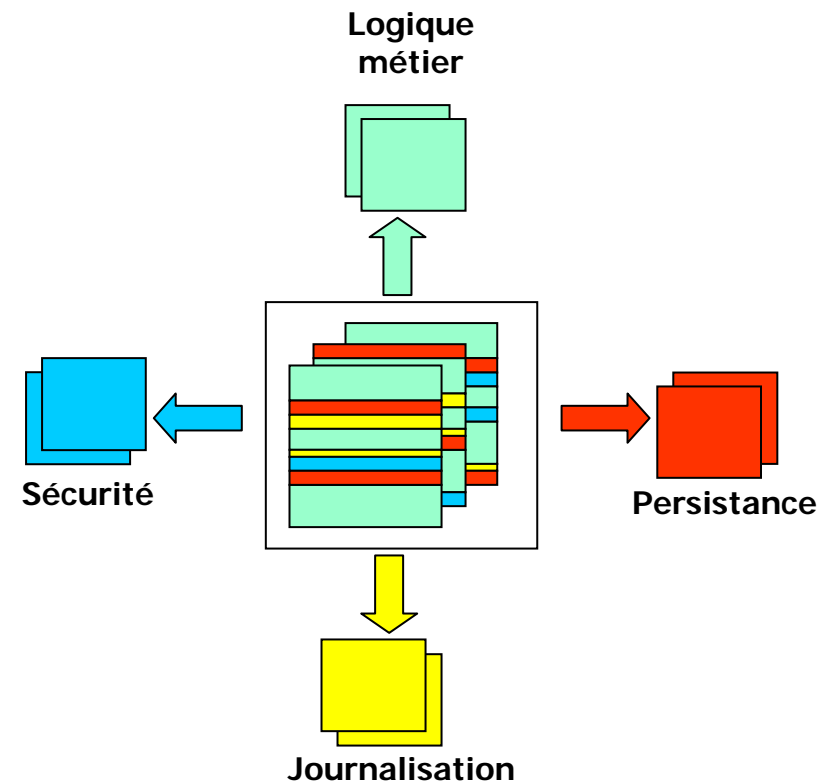
Objectifs

- Étudier les concepts de l'orienté aspect dans un cadre C++/CORBA.
- Proposer une solution aux problèmes posés par la gestion de cycle de vie des objets CORBA.
- Appliquer une approche orientée aspect à la gestion du cycle de vie dans une application scientifique.

Programmation orientée aspect (1/2)

Faciliter la séparation des préoccupations au niveau de l'activité de programmation

- Un système répond à un ensemble de préoccupations.
- Une préoccupation correspond à un « aspect » fonctionnel ou technique du système.



Programmation orientée aspect (2/2)

Terminologie

Concept	Commentaire
Point de Jonction (<i>joinpoint</i>)	Point précis dans l'exécution du programme.
Coupe (<i>pointcut</i>)	Spécifie un ensemble de points de Jonction.
Conseil (<i>advice</i>)	Fragment de code à insérer au niveau des Points de Jonction.
Aspect (aspect)	Unité de modularisation des préoccupations transverses.
Tisseur (weaver)	Applique le recoupement des aspects au code de base.

Deux approches

- Étendre la syntaxe d'un langage
- Fournir un cadre logiciel

Appliquer le tissage

- à la compilation
- au chargement
- à l'exécution

Cycle de vie dans CORBA (1/2)

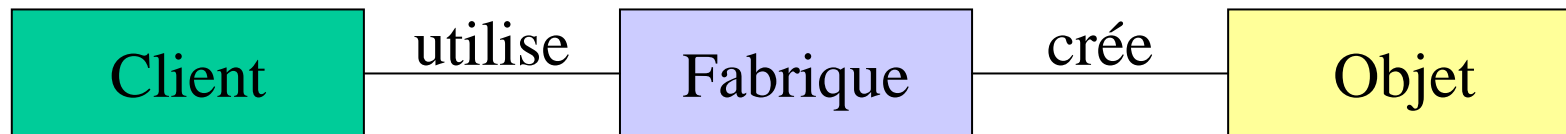
Problèmes posés

- Création d'un objet
 - quelle est l'entité que le client pourra contacter afin de créer un nouvel objet?
 - comment localiser cette entité?
- Destruction d'un objet
 - comment détruire les objets tout en garantissant de ne détruire que ceux réellement devenus inutiles?
 - quelle stratégie de destruction adopter? application ou système?

Cycle de vie dans CORBA (2/2)

Solution retenue

- Création/recherche d'un objet
 - Le client contacte une fabrique pour créer un nouvel objet ou récupérer une référence sur un objet existant.

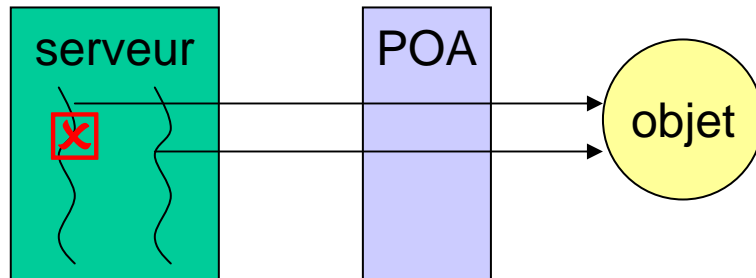


- Destruction d'un objet
 - Le client indique à l'objet qu'il n'utilise plus ses services en invoquant une méthode de libération exposée par l'objet.

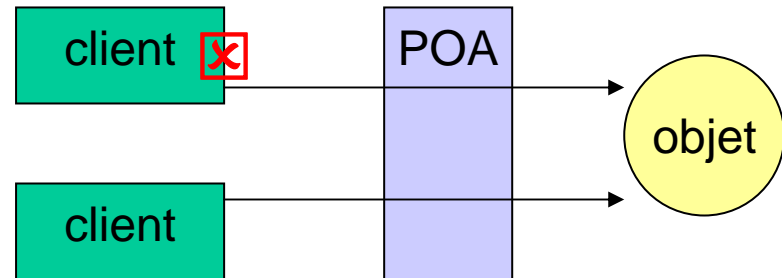
Implantation du cycle de vie (1/3)

Besoins

Gérer des activités simultanées



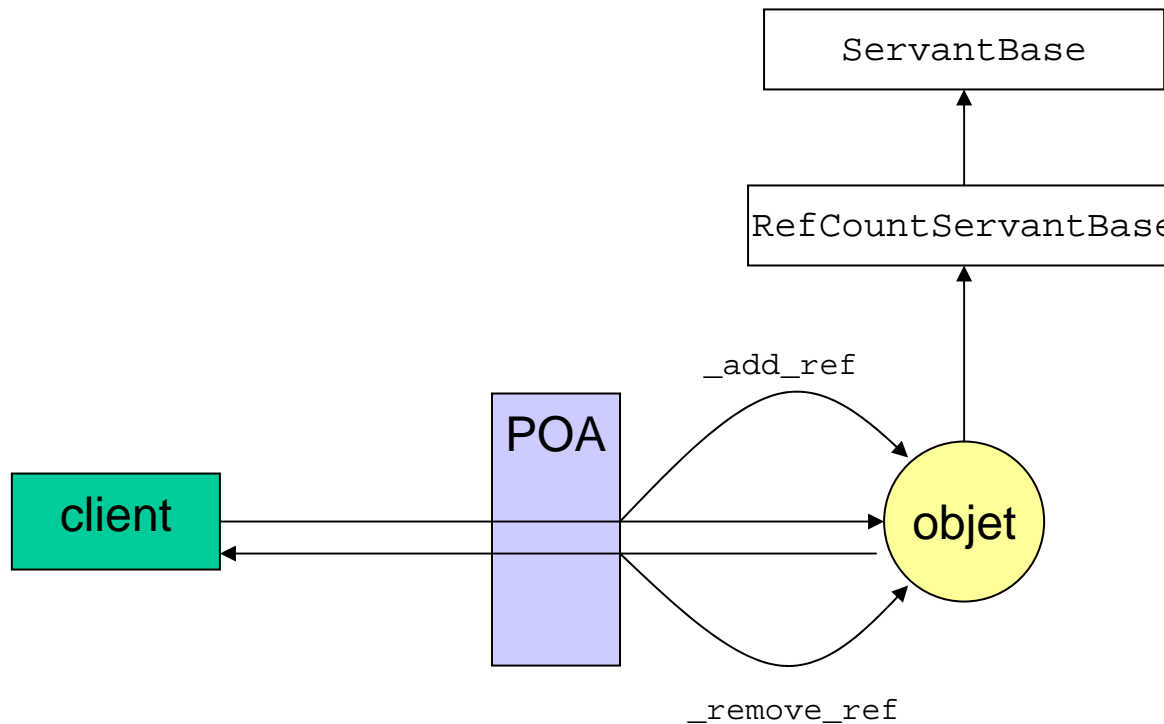
Gérer le partage d'objet entre plusieurs clients



Implantation du cycle de vie (2/3)

Solutions

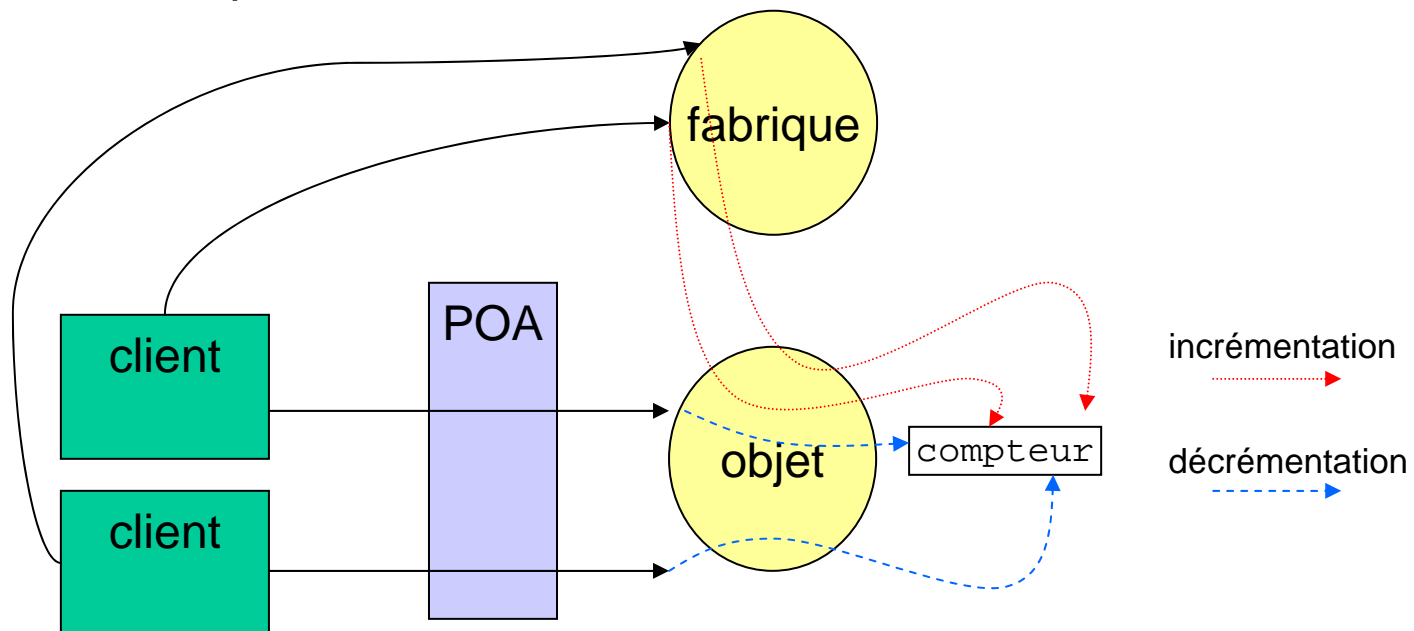
- utilisation du compteur de référence de `RefCountServantBase` pour répondre au premier besoin (gérer des activités simultanées).



Implantation du cycle de vie (3/3)

Solutions

- ajout d'un compteur de clients au niveau de l'objet pour répondre au second besoin (gérer le partage d'objet entre plusieurs clients).
- l'objet expose aussi deux nouvelles méthodes `register_client()` et `destroy()` qui ont pour effet, respectivement, d'incrémenter et de décrémenter le compteur.



Application des aspects (1/5)

Motivations

- La gestion du cycle de vie est un aspect technique d'une application CORBA
- Nécessité de modifier une multitude d'entités dans une large plate-forme
- Introduire cette modification implique du code éparpillé

→ Adopter une approche aspect pour la gestion de cycle de vie dans CORBA.

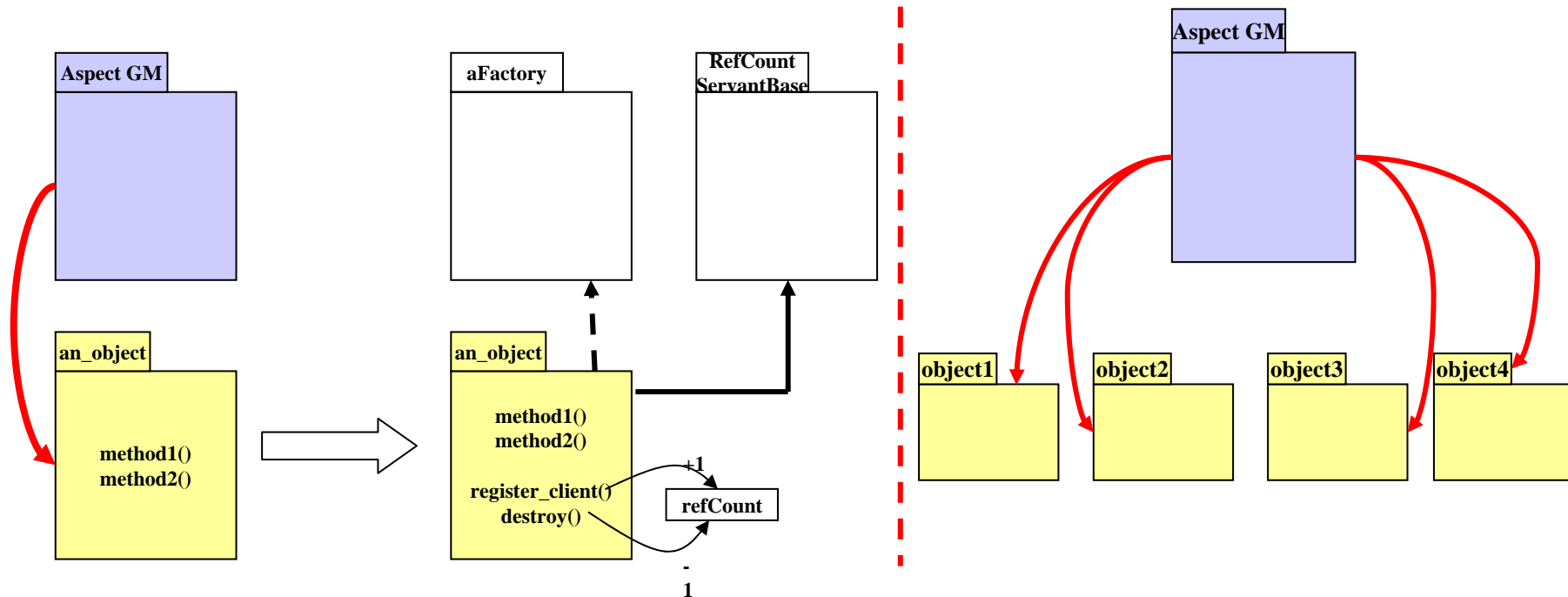
Application des aspects (2/5)

Syntaxe utilisée par le tisseur AspectC++

- Coupes de nom (*name pointcuts*): décrit un ensemble d'entités (connues à la compilation) du programme: types, attributs, fonctions, variables, espaces de désignation.
- Coupes de code (*code pointcuts*): référence un appel ou un point d'exécution d'une méthode.
- Exemples d'expressions de coupe:
 - `<< %List >> && !derived(<< Queue >>)`
 - `call(<< void draw(...) >>) && within(<< Shape >>)`
- Conseils (advice)
 - mots-clés : `before` / `after` / `around` / `baseclass`
 - par défaut : un conseil correspond à une introduction

Application des aspects (3/5)

Au niveau des serveurs



Au niveau des objets

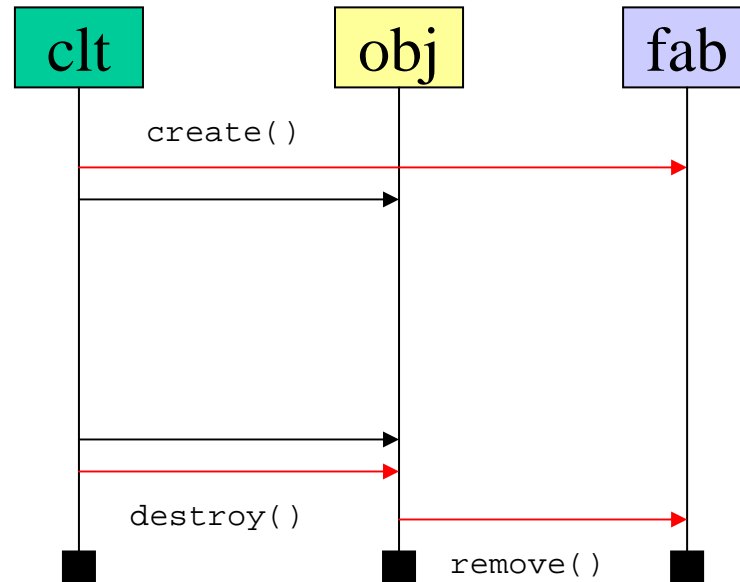
- Si inexistant, ajout de l'héritage de `RefCountServantBase`.
- Introduction d'un compteur de clients et d'une référence vers la fabrique.
- Introduction de deux méthodes `register_client()` et `destroy()`.

Au niveau des fabriques

- Si inexistante, ajout de la fabrique.
- Si existante, introduction de deux méthodes `create()` et `get()`.

Application des aspects (4/5)

Au niveau des clients



- Récupérer une référence sur une fabrique adéquate pour chaque type d'objet.
- Invoquer la méthode `create()` ou `get()` avant l'utilisation d'un objet.
- Invoquer `destroy()` lorsque le client n'utilise plus les services de l'objet.

Application des aspects (5/5)

Difficultés rencontrées

- IDL non pris en compte par le tisseur C++
- Interférences avec le code généré par le compilateur IDL
- Pas de possibilité de déclarer des coupes au niveau des constructeurs et des destructeurs
- Pas de possibilité d'introduire de nouveaux constructeurs
- Complexité du code C++ tissé

Bilan et perspectives

Bilan

- Application des aspects pour implanter, du côté des serveurs, le cycle de vie dans CORBA
- Utilisation du tisseur AspectC++

Perspectives

- Amélioration du mécanisme de cycle de vie
- Tissage dynamique en AspectC++
- Aspect au niveau IDL
- Modélisation orientée aspect
- Application des aspects à d'autres *design patterns*