

Un Protocole à Métaobjets pour Applications CORBA Tolérantes aux Fautes

**Marc-Olivier Killijian, Jean-
Charles Fabre, Juan
Carlos Ruiz-Garcia**

Groupe TSF

LAAS-CNRS

Toulouse

Shigeru Chiba

Université de Tsukuba

Japon

Objectifs

● Conception d'une Architecture CORBA Tolérante aux Fautes

- **Sûreté de Fonctionnement**
- **Flexibilité**
- **Réutilisabilité**

● Approche Réflexive

- **Expérience Précédente**
 - ↳ Transparence/Visibilité
 - ↳ Séparation/Composition des mécanismes
 - ↳ Limitations dues aux mécanismes réflexifs
 - ↳ Non CORBA
- **Pas de solution réflexive pour CORBA**
 - ↳ Intéressant pour d'autres domaines

Approche Réflexive

● **Autres Approches**

- **Intégration**
- **Interception**
- **Notion de service**

● **Propriétés Intéressantes**

- **Indépendant ORB**
- **Mécanismes de tolérance aux fautes en tant que logiciel CORBA**
- **Transparent pour le programmeur d'application**

● **L'Approche Réflexive fournit ces propriétés**

- **Basée sur le modèle objet CORBA**
- **Mécanismes comme objets CORBA**
- **Transparence grâce a la réflexivité**

La Réflexivité

● Système Réflexif

- un système qui peut raisonner à propos et agir sur lui même
- un système réflexif peut observer et contrôler son propre comportement

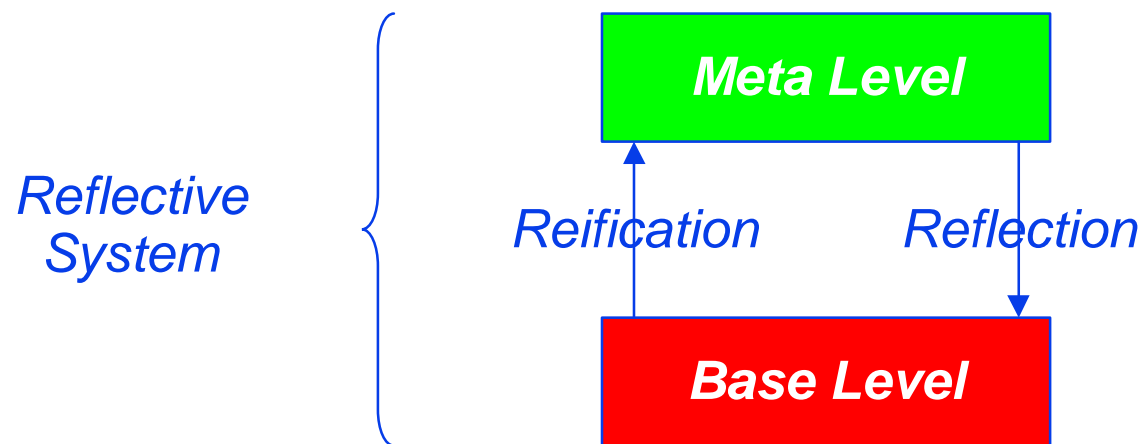
● Deux Notions Principales

○ Réification

↳ le processus qui expose un modèle de lui même

○ Réflexion

↳ le processus qui permet d'effectuer des modifications sur lui même



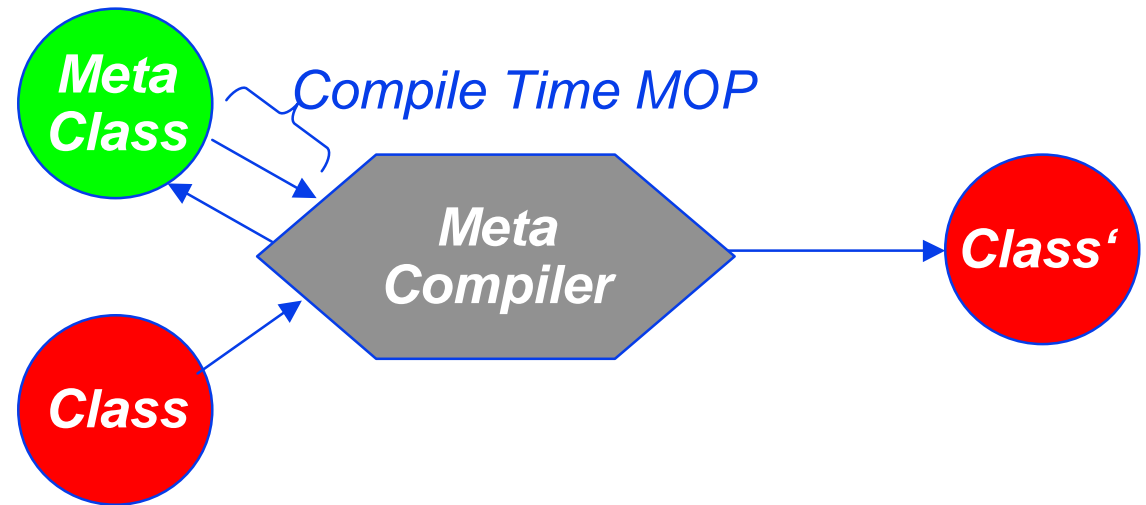
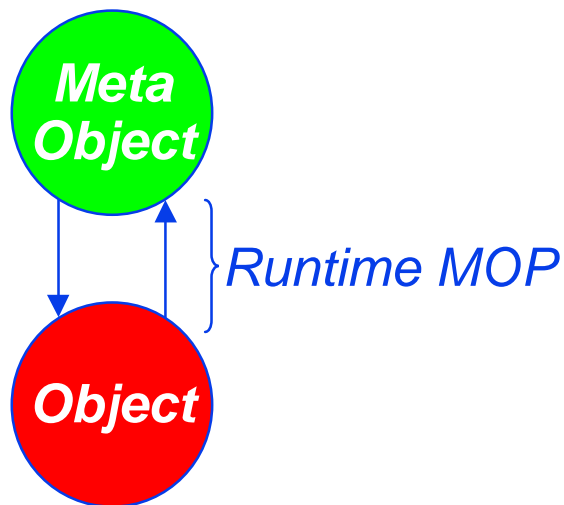
Protocoles à Métaobjets (MOPs)

● MOP Runtime

- métaobjet et objet existent en même temps
- pauvre modèle objet au métaniveau
- protocole fixe
- ex. Open C++ v1

● MOP Compile-Time

- des métaclasses contrôlent le processus de compilation
 - ↳ MOP runtime personnalisé
- modèle objet complet
- ex. Open C++ v2: méta-compilateur



Buts et Moyens

● Définition du MOP

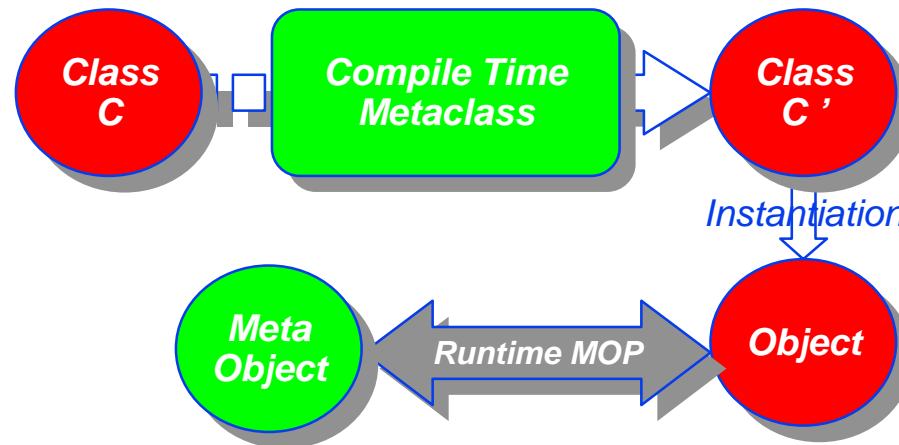
- **compatible avec le modèle objet CORBA**
- **conçu pour la tolérance aux fautes**
 - ↳ comportement objet
 - ↳ état objet
- **implémenté avec réflexivité compile-time**

● Conception du Système

- **tolérance aux fautes**
 - ↳ logiciel CORBA
 - ↳ utilisant ce MOP runtime
- **définition des services de base**
 - ↳ protocoles de communication de groupe
 - ↳ fabriques d'objets

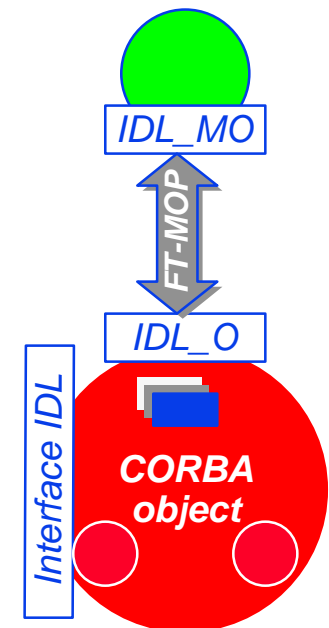
Notre Solution

● Réflexivité Compile-Time pour implémenter un MOP runtime



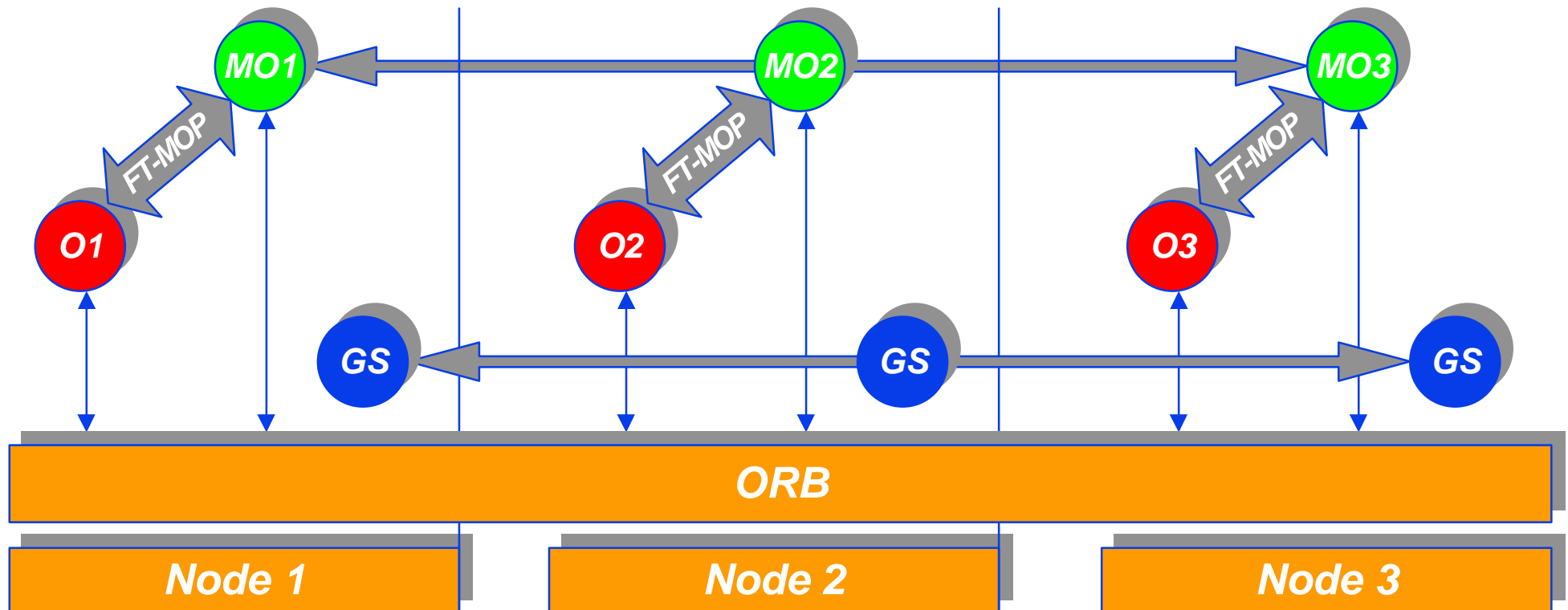
● MOP Runtime

- spécialisé pour la tolérance aux fautes
- Métaobjets runtime avec lien dynamique
 - ↳ adaptabilité aux hypothèses de fautes et conditions opérationnelles
 - ↳ configuration dynamique des applications en fonction des stratégies de tolérance aux fautes



Architecture Réflexive Tolérante aux Fautes

- **Metalevel** : métaobjets implémentant les besoins non fonctionnels
- **Service-level** : communication de groupe, fabriques d'objet etc.
- **Base-level** : objets applicatifs



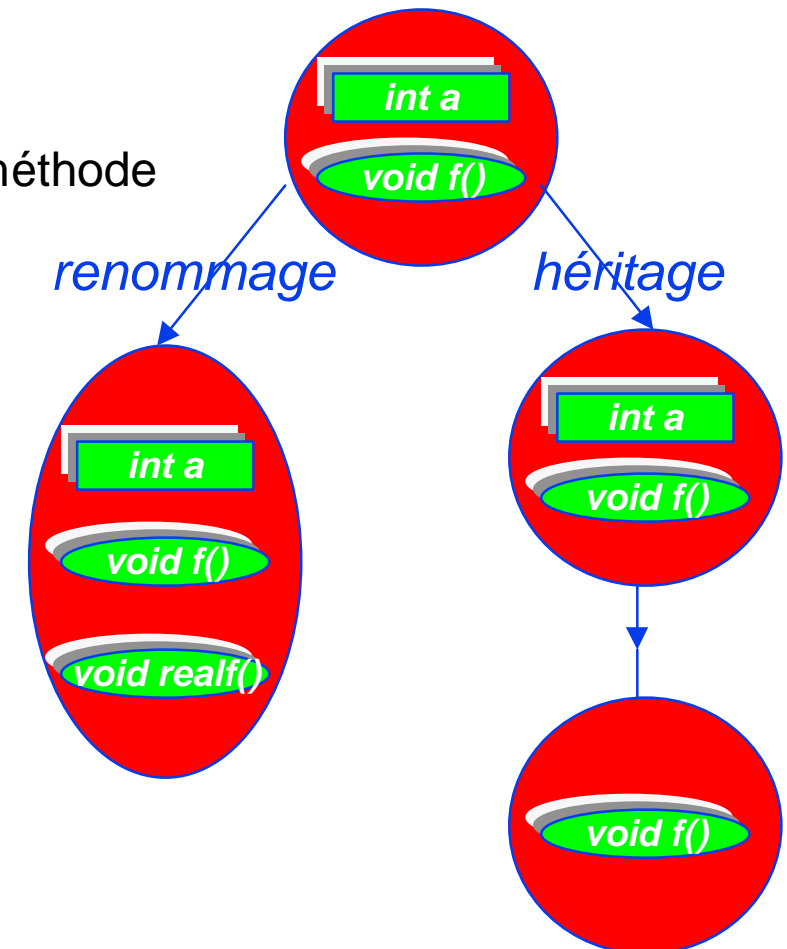
FT-MOP : Observation et Contrôle Comportemental

● A la compilation, une classe C est transformée pour réifier

- l'instanciation de classe (création d'objet)
- destruction d'objet
- invocation de méthode
 - ↳ paquetage et dépaquetage des paramètres de méthode

● Implémentation de l'Interception

- renommage des méthodes
- héritage



FT-MOP : Etat de l'objet

● Dans les Systèmes TaF : état nécessaire

- envoyer des checkpoints aux répliques passives
- cloner un objet au cours d'une reconfiguration

● Autres Solutions

- pages mémoires d'un objet
- fonctions virtuelles définies par l'utilisateur

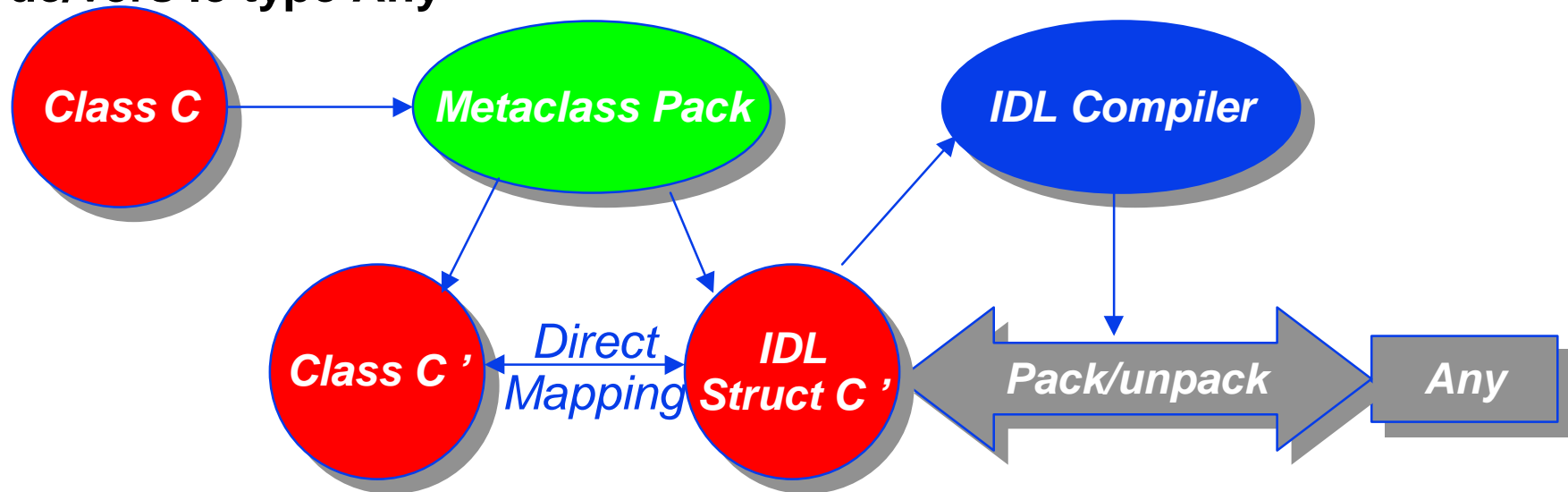
● Nos Solutions

- **Full-State : ensemble complet des attributs**
 - ↳ à la compilation, un type IDL est créé pour chaque classe
- **Delta-State**
 - ↳ à la compilation, un ensemble de règles de transformation permet à un objet de déterminer son propre delta-state au runtime.

Full State

- **Une structure IDL est créée à la compilation pour chaque classe**

- ces structures sont gardées dans l' "Interface Repository"
- le compilateur IDL génère les méthodes de paquetage/dépaquetage de/vers le type Any



- **Implémentation**

- types gérés : types basiques , tableaux, classes
- pointeurs (compatible avec les références Java)

Delta-Checkpointing

● Algorithme

- **Analyse à la compilation**
 - ↳ insertion de code pour “flagger” les attributs modifiés
 - ↳ création des méthodes pour fournir ou appliquer un delta-state
- **Runtime**
 - ↳ le métaobjet peut prendre le delta-state de son objet

● Implémentation

- **L'arithmétique de pointeurs est interdites**
 - ↳ quel attribut est accédé ?
 - ↳ solution coûteuse : copier avant/comparer après
 - ↳ full state
- **pas de pointeurs locaux**
- **tableaux : métaobjet-tableau**

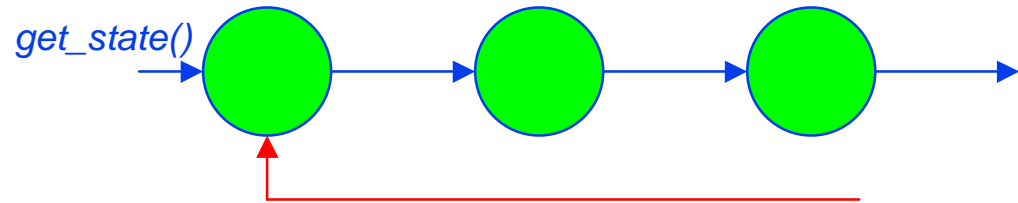
Restrictions de Programmation

	C++	Java
Fonctions et Classes « Friend » ↳ brisent l'encapsulation, non OO	☆	
Fonctions et Variables Globales ↳ non réifiées, non OO, données partagées	☆	
Pointeurs en Argument de Méthode ↳ peuvent briser l'encapsulation	☆	
Doubles, Triples Pointeurs, Arithmétique ↳ effets de bords, incontrôlables	☆	
Attributs Privés uniquement ↳ encapsulation, restriction contournable	☆	☆
Pointeurs en Variable Locale ↳ effets de bord, non contrôlable	☆	☆

Relations inter Objets

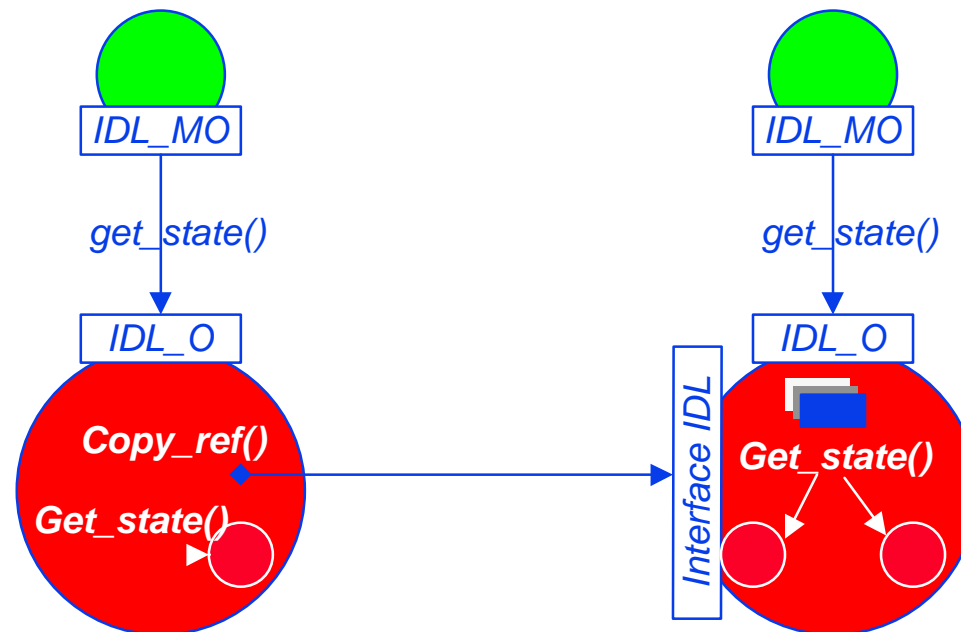
- **Composition** : un objet en contient un autre

- “deep copy”
- appels récursifs à `get_state()`

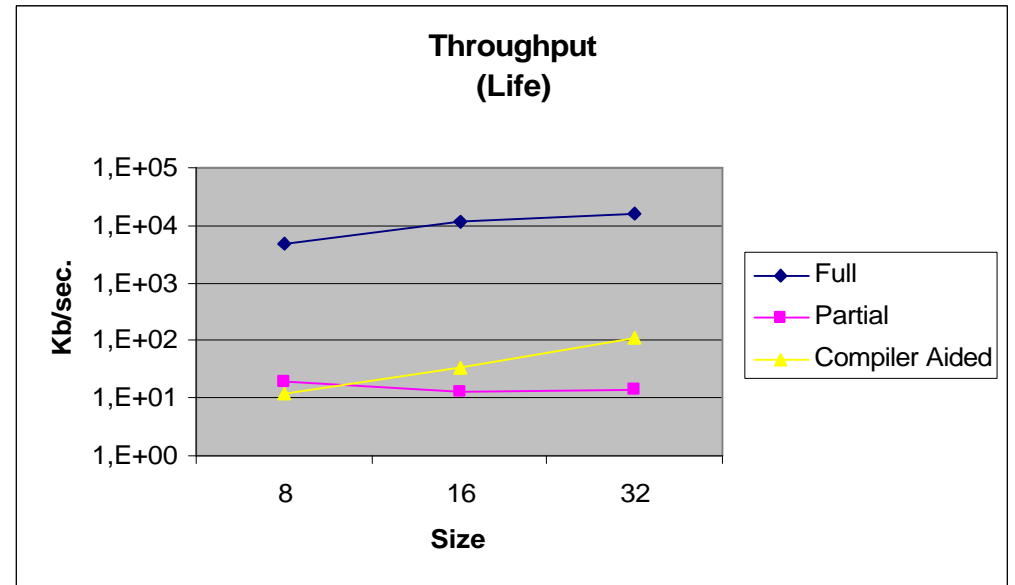
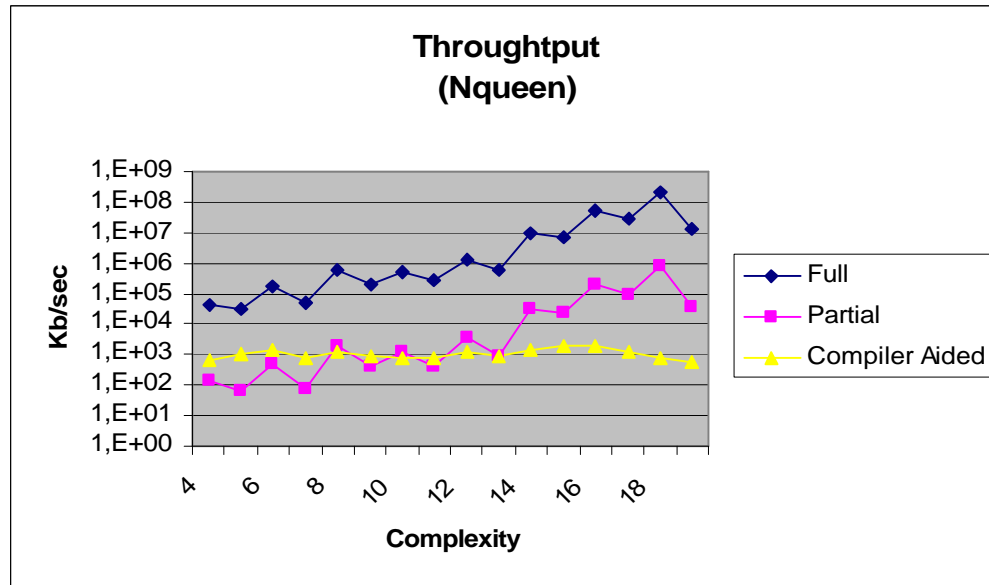
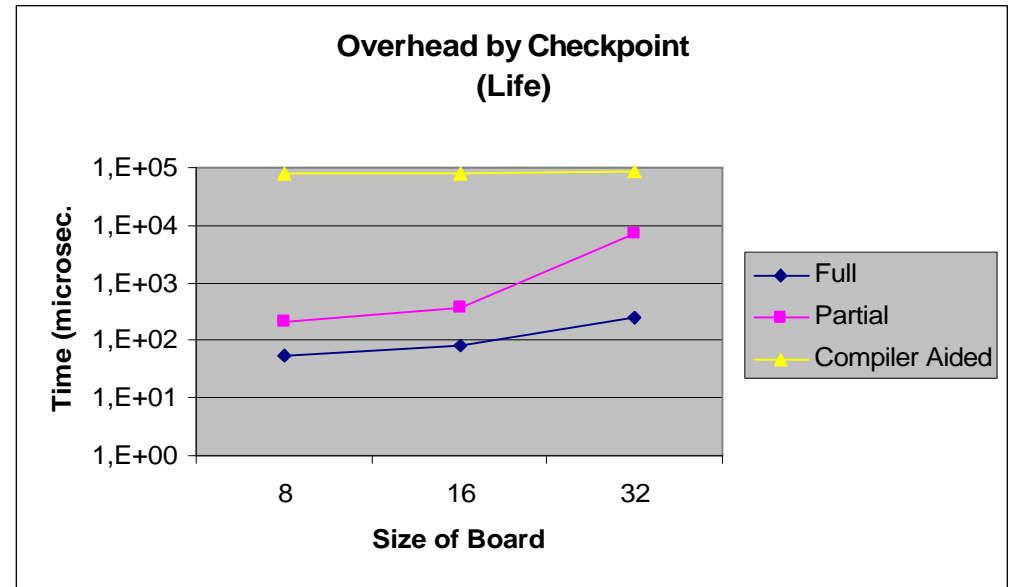
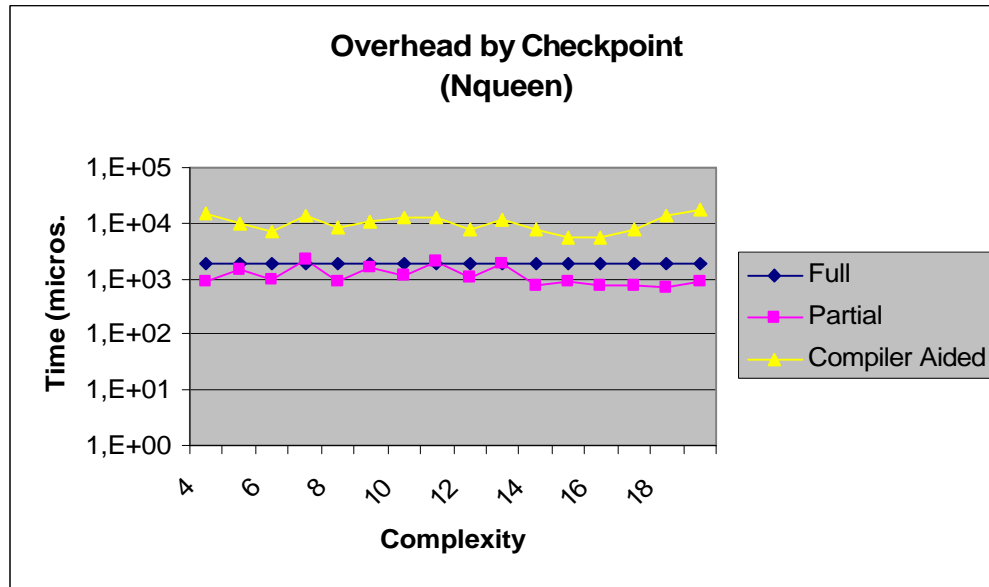


- **Délégation** : un objet en référence un autre

- la référence fait partie de l'état
- “shallow copy”



Performances



Work in Progress

● Prototype

- **Solaris ou Linux, avec 2 différents ORBs**
 - ↳ observation et contrôle comportemental (composition, héritage)
 - ↳ gestion du full-state avec restrictions du C++
 - ↳ delta-checkpointing
- **services de base:**
 - ↳ communication de groupe (avec xAMp)
 - ↳ fabrique d'objets

● En cours

- **extension du prototype : moins de restrictions, gestion des listes non-bornées etc.**
- **métaobjets de tolérance aux fautes**
- **objets/métaobjets Java**

 *Friends 2*

Conclusion

- **Réflexivité Compile-Time pour des MOPs personnalisés**
 - fournit un modèle objet complet
 - faire un MOP adapté aux besoins du système
- **MOP Runtime pour la Tolérance aux Fautes**
 - comportement des objets
 - capture automatique de l'état des objets
 - ↳ dépendant du langage
- **MOP pour CORBA**
 - Visibilité / Transparence / Séparation : MOP
 - Interoperabilité / Reutilisabilité / Extensibilité : CORBA software
 - Configuration Dynamique / Adaptation
 - Indépendant ORB