

Club SEE *Systemes Informatiques de Confiance*
Journée «Technologie objet et systemes de confiance»

Description formelle d'un protocole à métaobjets

Eric Marsden
<emarsden@laas.fr>

Groupe *Tolérance aux fautes et Sûreté de Fonctionnement informatique*
LAAS-CNRS



Problématique

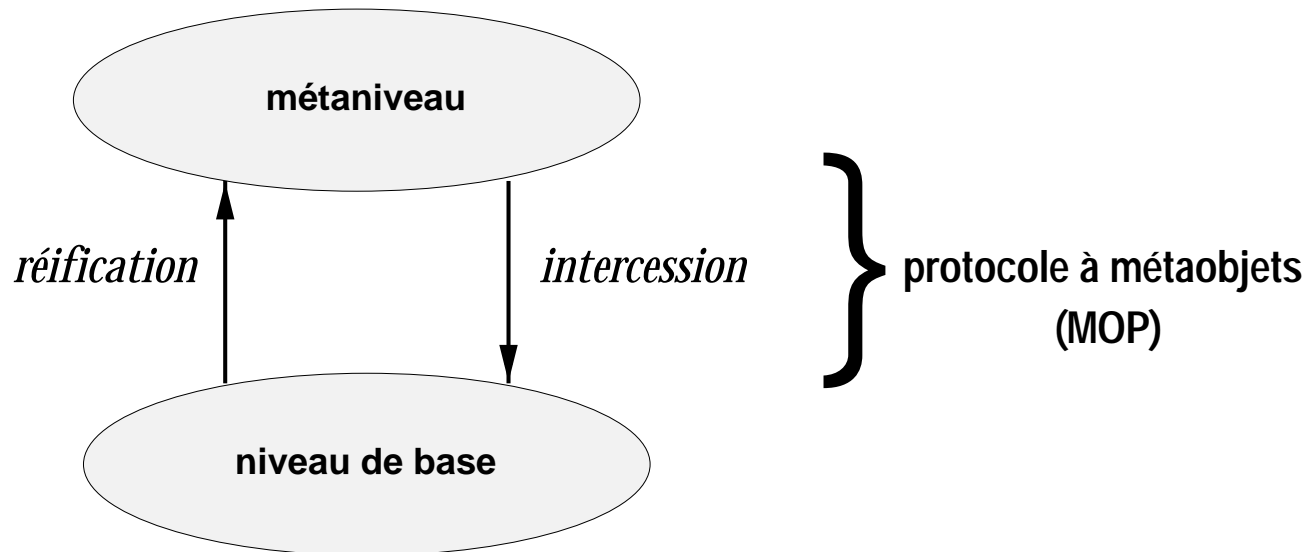
- ▶ **sûreté de fonctionnement** d'un système informatique : *capacité à délivrer un service dans lequel on peut placer une confiance justifiée*
- recherche de mécanismes adaptables, facilement utilisables, et réutilisables
- utilisation de techniques réflexives
- quid de la validation de ces mécanismes?
- commencer par le protocole à métaobjets, la pierre angulaire d'une architecture réflexive

Plan

- architectures réflexives et protocoles à métaobjets
- l'architecture FRIENDS : réflexive, basée sur CORBA
- choix d'un formalisme
- le π -calcul : un algèbre de processus pour la mobilité
- notre modèle du protocole à métaobjets
- résultats et analyse
- perspectives

Réflexivité

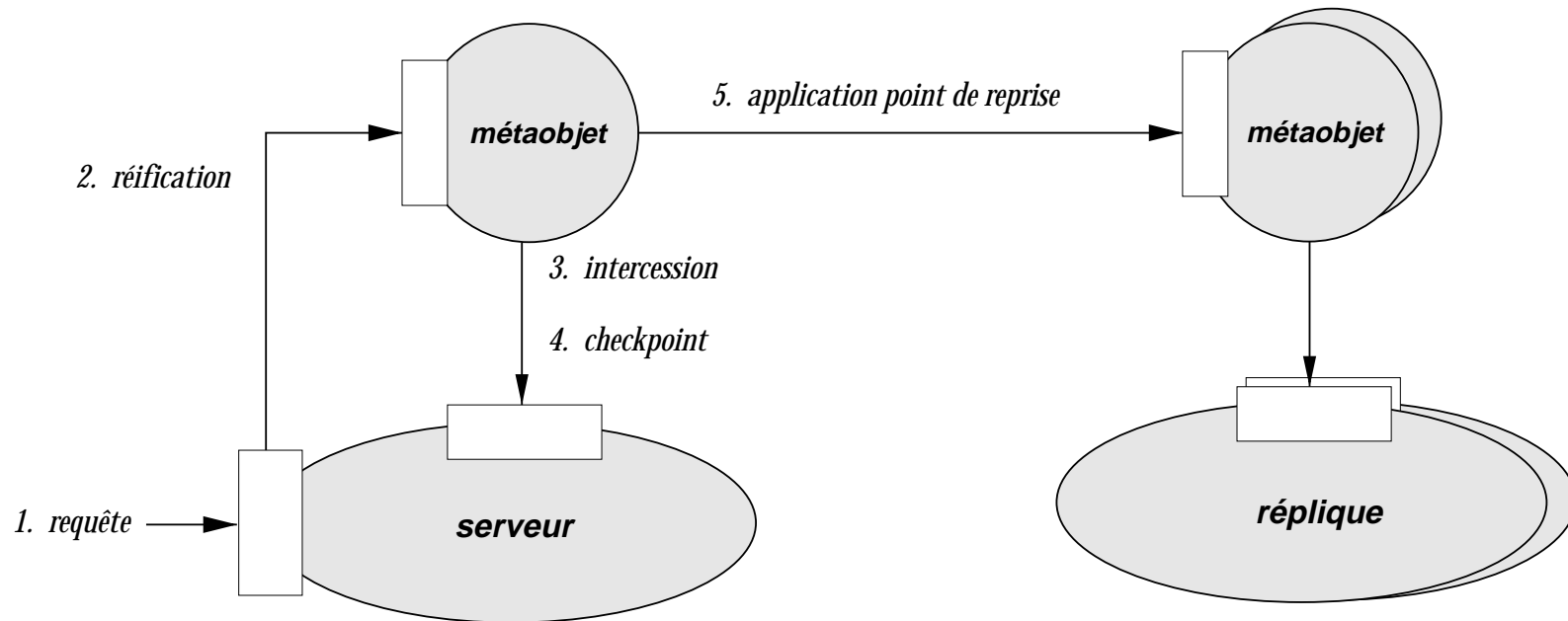
- ▶ la capacité d'un système à *raisonner et agir sur lui-même*
- ▶ un système réflexif peut *observer et contrôler son propre comportement*



- permet une séparation entre un niveau applicatif et un niveau non fonctionnel

Exemple : tolérance aux fautes

Objectif : assurer la disponibilité d'un service CORBA par réplication

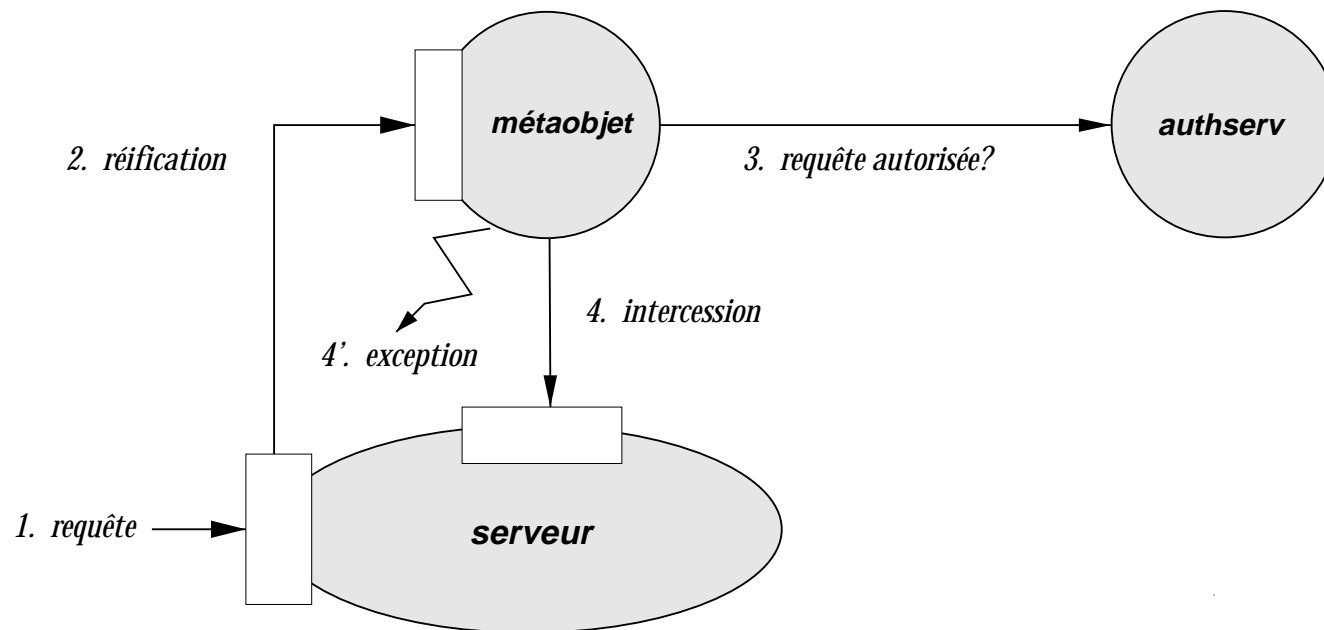


- transparent pour le programmeur d'application
- permet de changer dynamiquement de stratégie de tolérance aux fautes

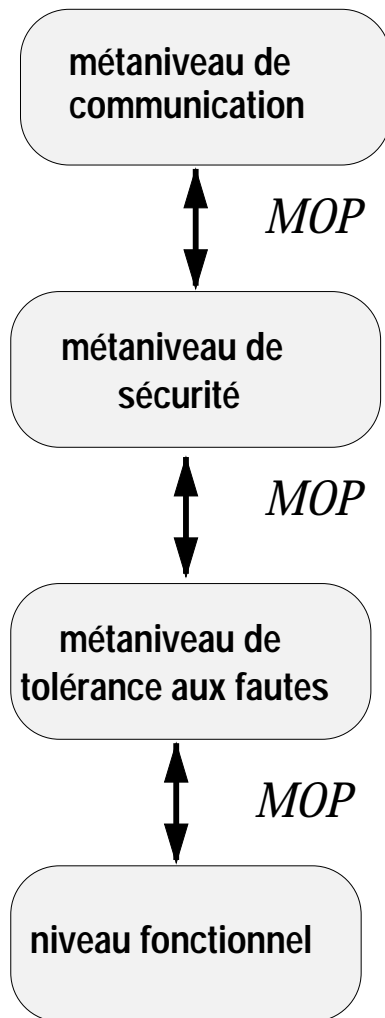
Exemple : sécurité

Sécuriser un système distribué :

- chiffrement des communications
- contrôle d'accès sur les invocations



L'architecture FRIENDS



- ▶ architecture réflexive orientée objet basée sur CORBA, pour la sûreté de fonctionnement

Bénéfices de l'approche :

- séparation des tâches de développement
- transparence des mécanismes
- composabilité et réutilisabilité des mécanismes non fonctionnels
- interopérabilité

Choix d'un formalisme

Objectifs de la modélisation :

- développer une description précise du protocole à métaobjets
- faciliter la communication avec les concepteurs et les spécialistes du test

Caractéristiques du problème :

communication : les mécanismes d'interception et de contrôle passent par échange de messages entre objet et métaobjet

concurrence : une application distribuée est la composition concurrente d'un ensemble d'objets

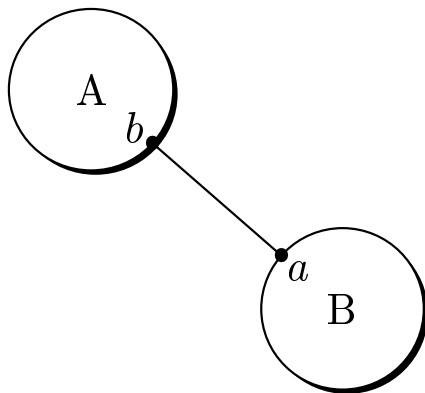
abstraction : un objet présente des services via un interface

dynamisme : les objets peuvent être instanciés dynamiquement; il faut créer de nouvelles entités et les inclure dans le modèle

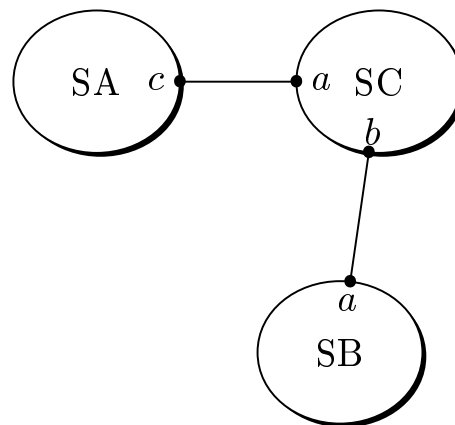
Le π -calcul

- modèle abstrait pour les systèmes concurrents [Milner 1989]
- algèbre de processus dans le style de CCS et LOTOS
- un système est représenté par un ensemble de processus communiquant et se synchronisant par échange de messages sur des ports
- langage formel simple comportant des opérateurs de choix, de séquence, de composition parallèle et de restriction
- bien adapté à la modélisation de systèmes où la topologie d'interconnexion peut évoluer dans le temps (mobilité)

Exemple π -calcul

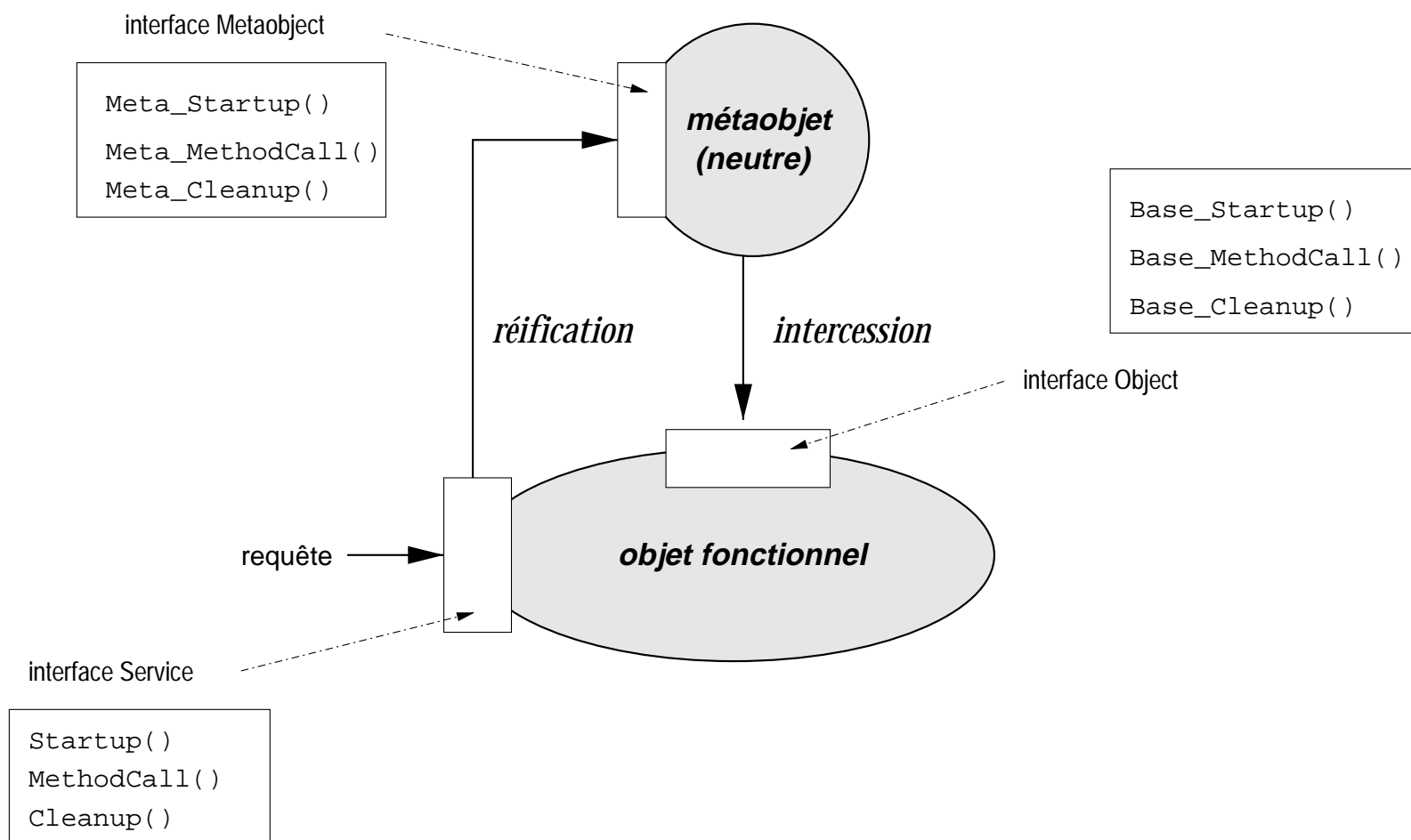


agent $A(b) = 'b<5>.A(b)$
 agent $B(a) = a(x).B(a)$
 agent $\text{System} = (\tilde{p})(A(p) \mid B(p))$

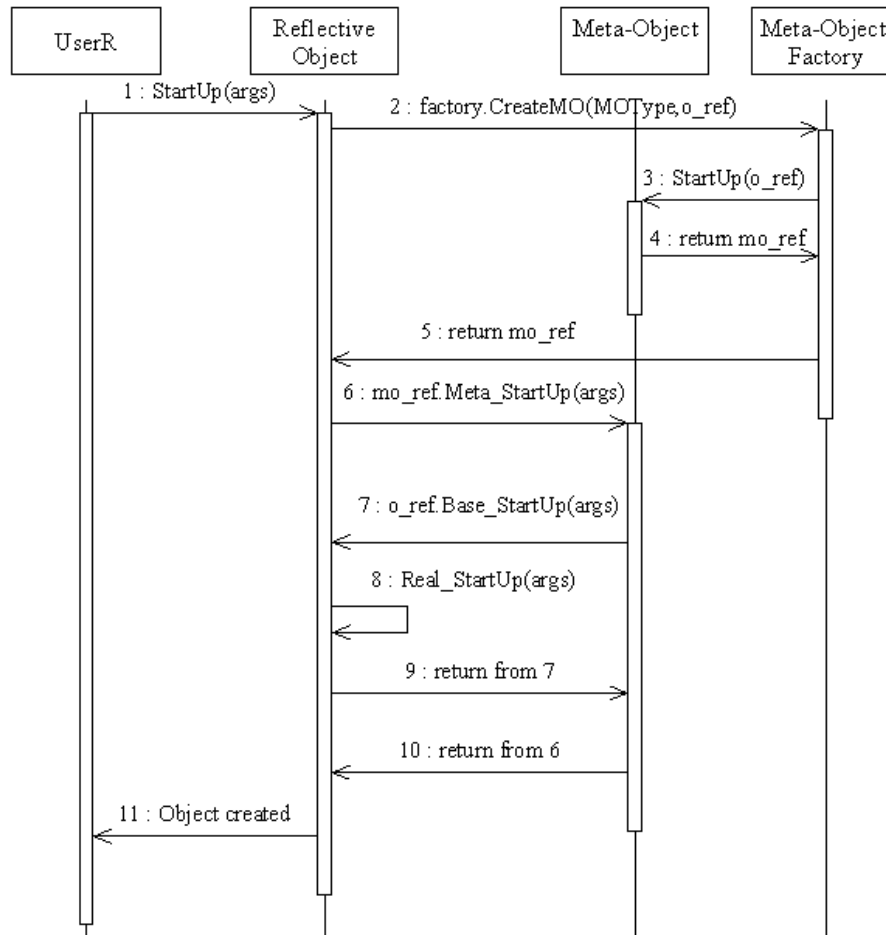


agent $SC(a, b) = a(x).'b<x>.SC(a, b)$
 agent $SA(b) = (\tilde{c})(SC(c, b) \mid A(c))$
 agent $SB(a) = B(a)$
 agent $S\text{System} = (\tilde{p})(SA(p) \mid SB(p))$

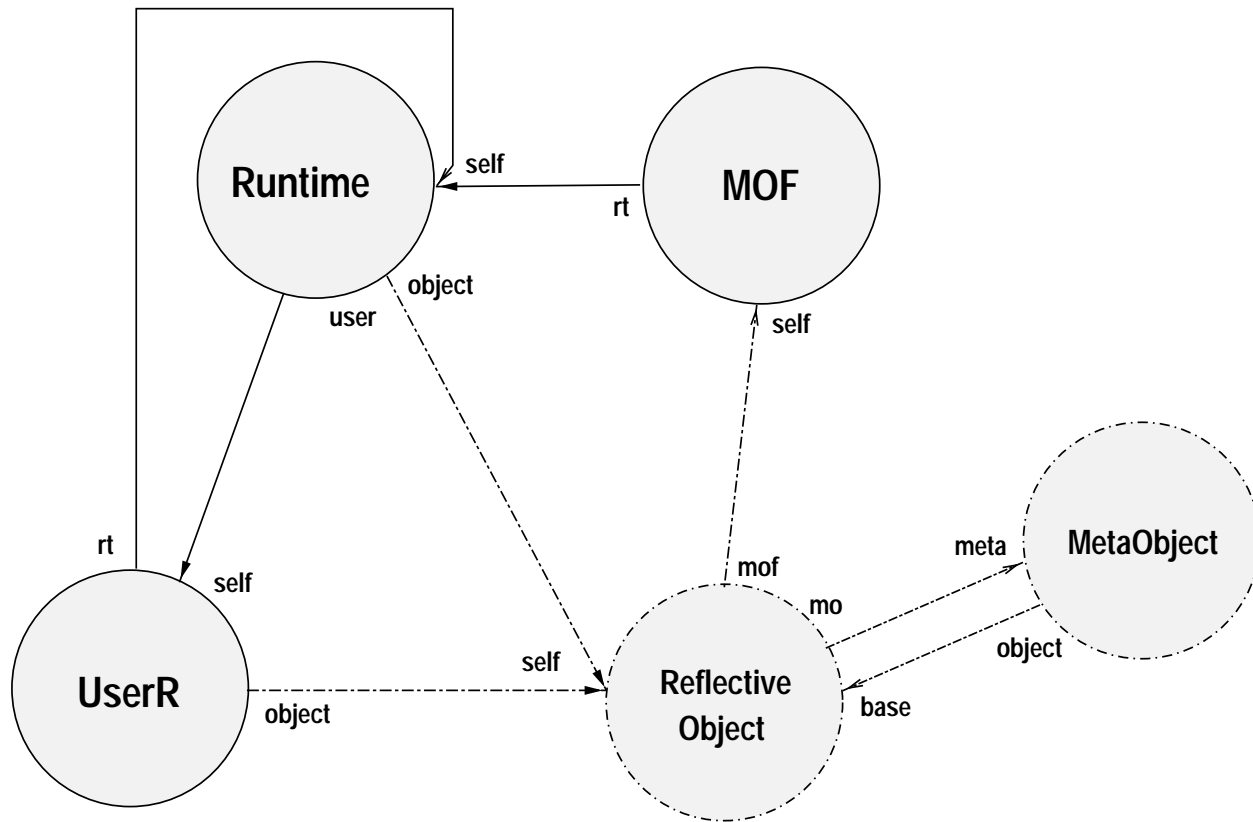
Le protocole à métaobjets



Phase d'initialisation



► diagramme de séquence UML, généré à l'aide de l'outil *Rational Rose*



```

agent ReflectiveSystem =
  (~mof, rt, user, MESSAGES)
  (UserR(user, rt, MESSAGES) |
   MOF(mof, rt, MESSAGES) |
   Runtime(rt, user, mof, MESSAGES))

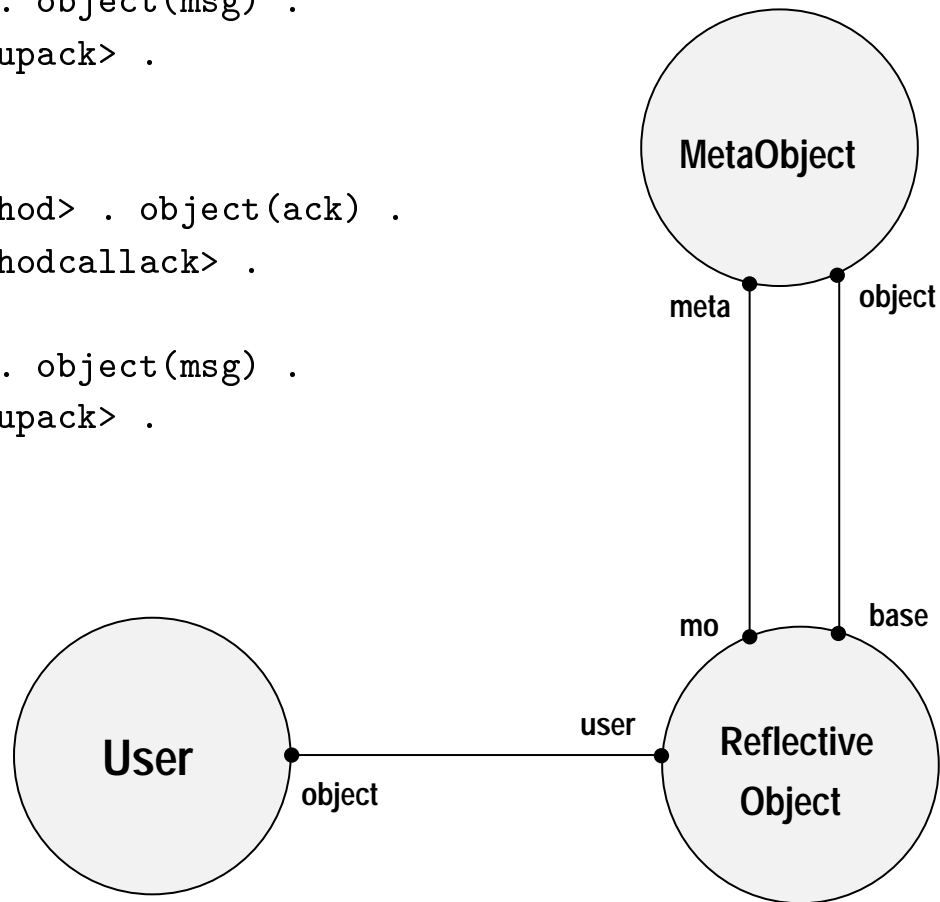
```

```

agent MetaObject(self, meta, object) =
  self(msg) . [msg=startup] 'self<startupack> .
  MetaObjectLoop(self, meta, object)

agent MetaObjectLoop(self, meta, object) =
  meta(msg) .
  ([msg=metastartup] 'object<basestartup> . object(msg) .
   [msg=basestartupack] 'meta<metastartupack> .
   MetaObjectLoop(self, meta, object)
  +[msg=metamethodcall] meta(method) .
   'object<basemethodcall> . 'object<method> . object(ack) .
   [ack=basemethodcallack] 'meta<metamethodcallack> .
   MetaObjectLoop(self, meta, object)
  +[msg=metacleanup] 'object<basecleanup> . object(msg) .
   [msg=basecleanupack] 'meta<metacleanupack> .
   MetaObjectLoop(self, meta, object))

```



Que faire du modèle?

- démontrer l'absence d'interblocage (propriété de sûreté)
- démontrer la **transparence fonctionnelle du protocole à métaobjets**, par bisimulation

$$O \cong (O, \wedge O_\phi)$$

(à l'aide d'un model checker)

- obtenir des **traces d'exécution** : montrer que les mécanismes d'interception fonctionnent correctement, que le contrôle est effectif (par animation ou logique modale)

Bisimulation

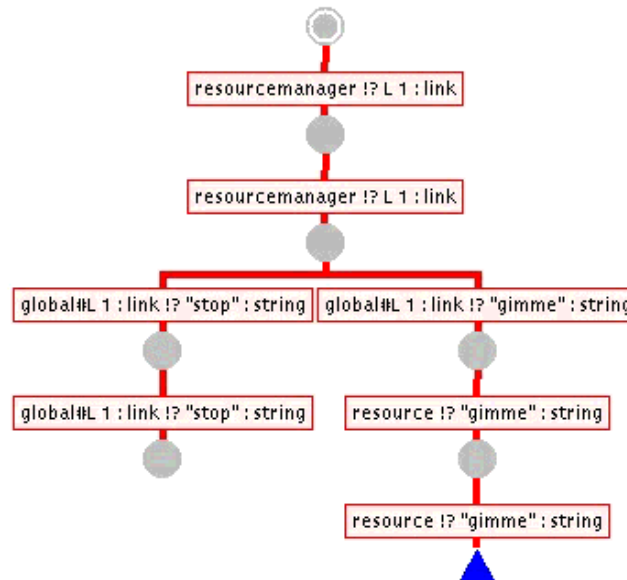
- ▶ deux processus sont dits *bisimilaires* lorsqu'ils ont le **même comportement observable** (chacun peut simuler le comportement de l'autre)
 - n'implique pas l'égalité, puisque les actions internes ne sont pas visibles
 - à chaque fois qu'un processus peut faire une action, l'autre doit pouvoir prendre la même action

Démontrer la bisimulation :

- model checkers tel que le *Mobility Workbench*
- prouveurs généraux (PVS, Isabelle)

Animation de modèle

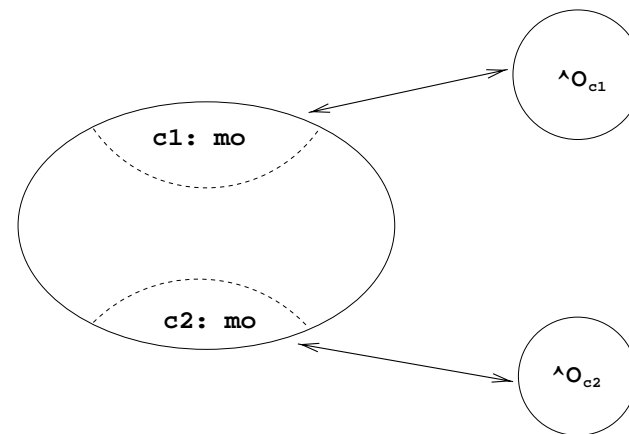
L'outil LCS (SML avec des extensions CCS) permet d'animer un modèle graphiquement :



Résultats

- validation des mécanismes réflexifs pour le comportement : le modèle sert à l'oracle
 - ▶ détection d'un problème d'unicité du lien object / métaobjet en comparant les traces de l'animation du modèle à celles de l'exécution de l'implémentation :

```
class c1 {  
    ...  
    MO_ref mo;  
}  
  
class c2: public c1 {  
    ...  
    MO_ref mo;  
}
```



- validation des mécanismes réflexifs pour la gestion de l'état : problème de granularité de la modélisation

Conclusions

- **originalité** : modélisation du protocole à métaobjets d'un système distribué orienté objet réflexif
- permet la **représentation du comportement dynamique** des différents agents et de leurs interactions
- on obtient une bonne visibilité sur les interactions concurrentes possibles
- représentation graphique intuitive du modèle
- l'écriture algébrique permet la **démonstration de propriétés** de sûreté et de vivacité
- complémentarité du travail de modélisation et de test

Perspectives

- extension de la modélisation aux mécanismes de tolérance aux fautes et de sécurité
- automatisation de la comparaison des traces d'exécution générées par le modèle et celles de l'implémentation
- nouveaux algèbres pour la mobilité : fusion calculus [Université Uppsala], join calculus [INRIA-Para], mobile ambients [Cardelli]
- autres méthodes pour démontrer la bisimulation (prouveurs généraux)