Exposé de thèse

Conception d'Algorithmes Répartis

et de Protocoles Réseaux

en Approche Objet

Lionel Seinturier
seinturier@cnam.fr

CNAM - CEDRIC
292 rue St-Martin
75141 PARIS cedex 03
FRANCE

Plan

1 Introduction

2 Présentation de l'approche

3 Coordination inter-objets

4 Coordination intra-objet

5 Exemple d'applications

6 Conclusion

1 Introduction

But

Définir une démarche de conception pour des applications distribuées utilisant des objets concurrents

Types d'applications visées

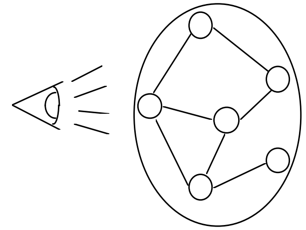
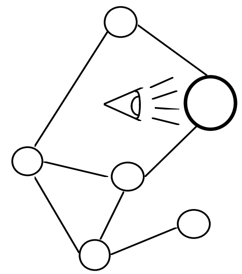
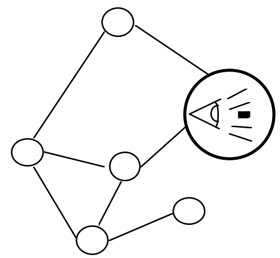
Algorithmes répartis
- **pour le domaine des réseaux**
- **pour la tolérance aux pannes**
- **pour le calcul parallèle réparti**

Algorithmes coopératifs pour les systèmes multi-agents

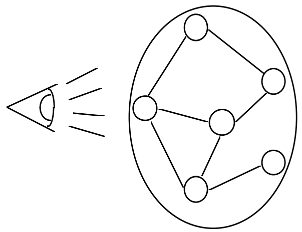
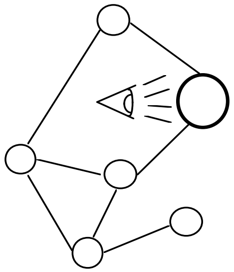
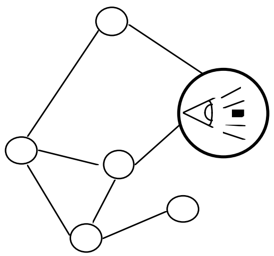
2 Présentation de l'approche

Démarche de conception

- descendante
- avec 3 niveaux méthodologiques
- principe de raffinement

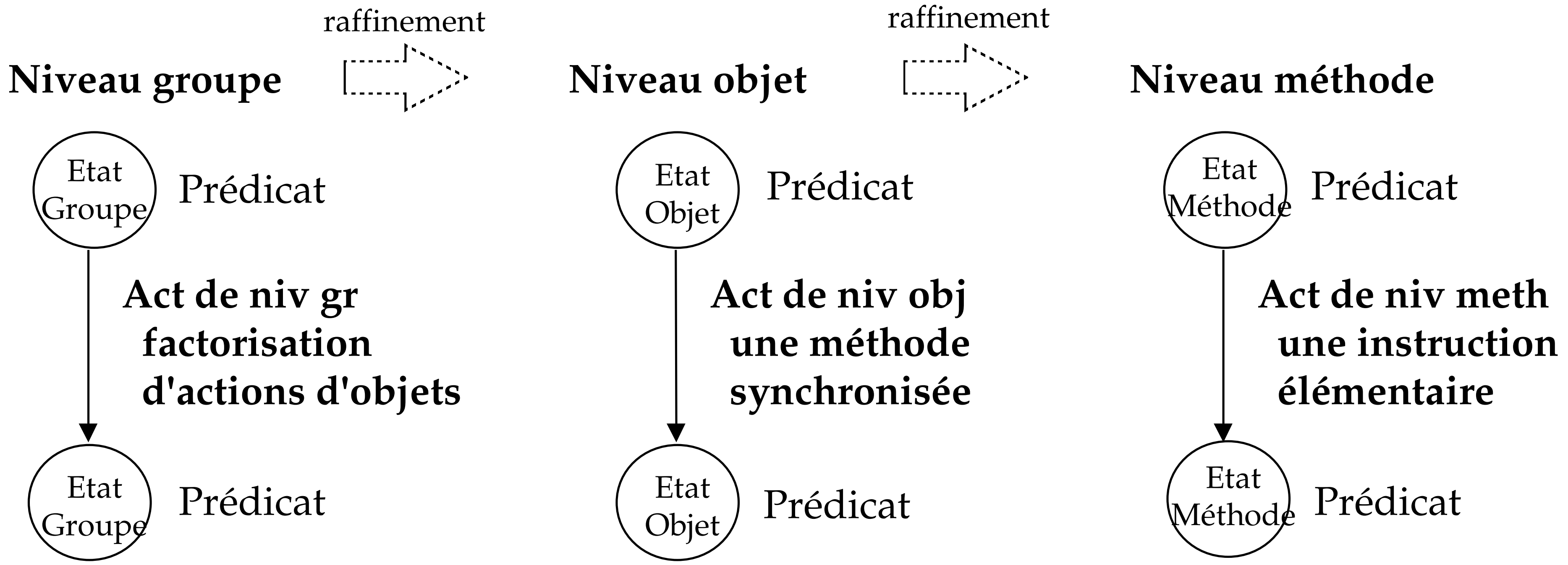
Niveau groupe	
Niveau objet	
Niveau méthode	

2 Présentation de l'approche

	- concurrence au sein d’un groupe - coordination des activités du groupe - définition des interactions => notion de programme de niveau groupe
	- concurrence au sein d'un objet - définition de politiques de synchronisation - maintien de la cohérence des données de l'objet => un modèle de synchronisation pour des objets concurrents
	- instructions - structures algorithmiques - données manipulées

2 Présentation de l'approche

Essayer d'avoir une démarche intégrée pour ces trois niveaux :
des modèles états/transitions
avec une sémantique adaptée au parallélisme inter et intra-objet



3 Coordination inter-objets

Aspects traités à ce niveau

- concurrence au sein du groupe
- coordination du comportement de groupe
- définition des interactions
- définition des niveaux de connaissance de groupe

Solution proposée

Des programmes de groupe utilisant

- une logique de la connaissance [Halpern 95]
- 4 structures de contrôle de groupe

3 Coordination inter-objets

Syntaxe du langage de groupe

group vars
    g1, g2 : ... /* variables de groupe typées */
environment
    h1 $\wedge$ h2 : ... /* hypothèses de bon fonctionnement */
begin
    a1 ; /* action de groupe */
    { Prédicat épistémique avant } a2; { Prédicat épistémique après }
    case variable **of** ... **endcase**
    if condition
    then ... **else** ... **endif**
    while condition **do** ... **enddo**
end

3 Coordination inter-objets

Etat de groupe

- un n-uplet (état de l'environnement , états des objets)

  ⇨ peu pratique dans le cadre d'un processus de conception

- définir la connaissance manipulée par chaque objet du groupe

  Halpern et al. : une logique épistémique et des modalités de connaissance

  $K_i\varphi$ ≡ le site $i \in G$ connaît le fait φ

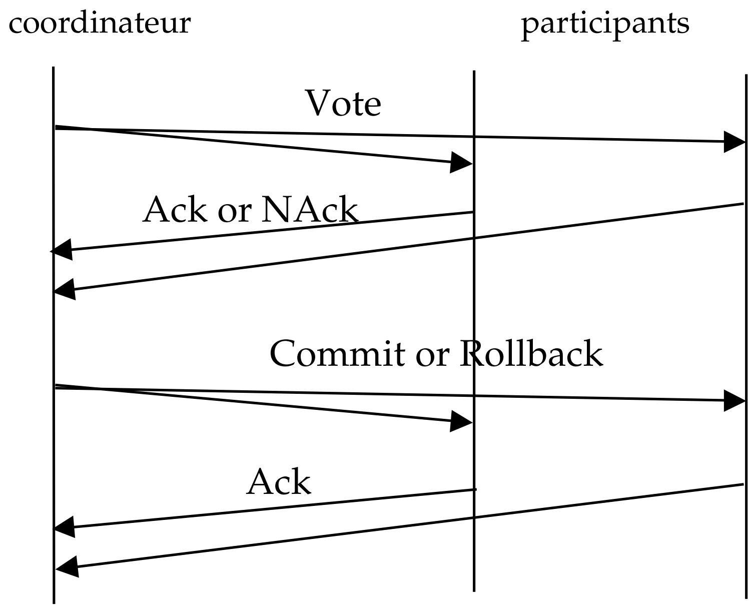
  $D_G\varphi$ ≡ personne dans G ne connaît φ mais G peut inférer φ

  $E_G\varphi$ ≡ tout le monde dans G connaît φ

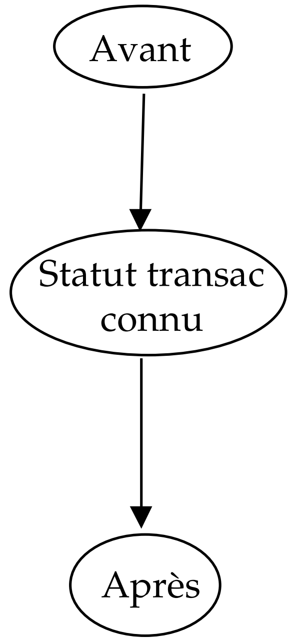
3 Coordination inter-objets

Protocole de validation à 2 phases

[Bernstein 87, Concurrency Control & Recovery in Database Systems]



Statut de la transaction =
(v:validable, a:annulable)

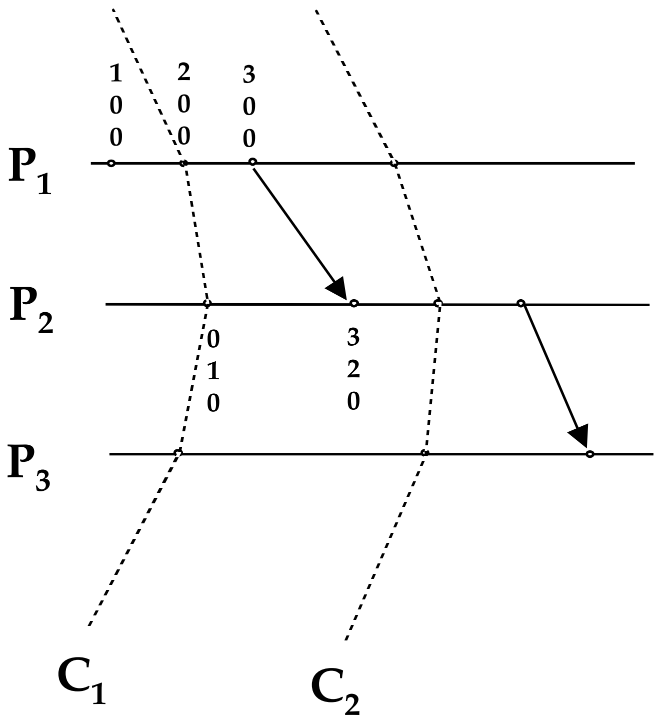


D_G(statut=v) $\vee$
D_G(statut=a)

Kcoord(statut=v) $\vee$
Kcoord(statut=a)

E_G(statut=v) $\vee$
E_G(statut=a)

3 Coordination inter-objets



C_1

P_1 : K locale
P_2 : K locale
P_3 : K locale

C_2

P_1 : K locale
P_2 : K locale + K sur P_1
P_3 : K locale

3 Coordination inter-objets

Actions de niveau groupe

Définition de structures de contrôle

- la phase
- la conditionnelle distribuée
- l'itération distribuée
- la récursion distribuée

phase	conditionnelle distribuée	itération distribuée	récursion distribuée
action	case	while	invocation récursive

3.1 La Phase

Comparable au : concept d'action à un niveau groupe

Déf : exécuter un comportement global sur l'ensemble des membres d'un groupe

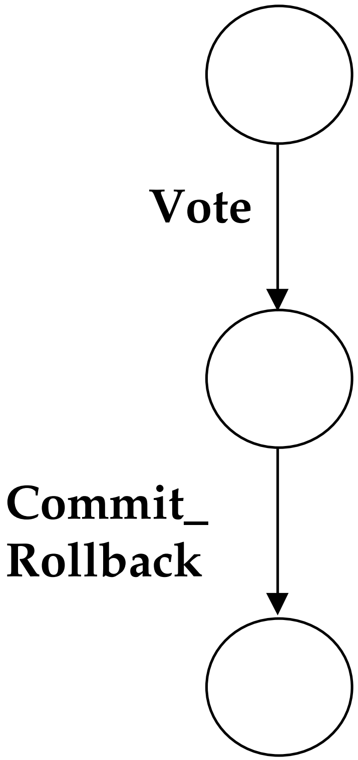
Exemple : la phase d'engagement dans un protocole transactionnel

Motivations :

- **rassembler sous une même dénomination des actions locales qui font partie d'un même schéma global**

- **désigner les actions exécutées entre deux états**

3.1 La Phase



Programme de niveau groupe

traduit l'évolution de l'ensemble
des objets qui font partie de l'algorithme

begin
    self. Vote **;**
    self. Commit_Rollback
end

self : le groupe courant

3.2 La Condition Distribuée

Comparable : les instructions *case* ou *if* d'un langage de programmation

Déf : modifier le comportement local de tous les membres d'un groupe
en fonction d'une condition sur l'état global du groupe

Exemple : un protocole de validation à deux phases

Motivations :

- **changer dynamiquement la stratégie de résolution du problème distribué**

- **choisir des comportements alternants en cas de panne**

- **détecter une condition globale**

3.2 La Condition Distribuée

Programme de niveau groupe

```
group vars
   Var
begin

   case Var of
      val1 : self.Action1
      ...
      valn : self.Actionn
      else : self.Actione
   endcase
end
```

2 cas :

- variable Var **dupliquée**
=> évaluation immédiate

- structure distribuée
=> insertion d'une phase de collecte
self. Collecter**;**

3.3 L'Itération Distribuée

Comparable à : l'instruction *while* d'un langage de programmation

Déf : itérer un comportement global tant qu'une condition est vérifiée

Exemple : réexécution périodique d'un algorithme de routage

```
group vars
    Cond : boolean
begin

    while Cond do
        self.ActionAItérer ;

    enddo
end
```

2 cas :

- variable dupliquée
=> évaluation immédiate

- structure distribuée
=> insertion d'une phase de collecte
self. Collecter**;**

3.4 La Récursion Distribuée

Comparable à : l'appel récursif

Def : invocation récurvise d'une même méthode sur un ensemble d'objets

Exemple : algorithme de calcul d'arbres couvrants
utilisant des vagues de parcours

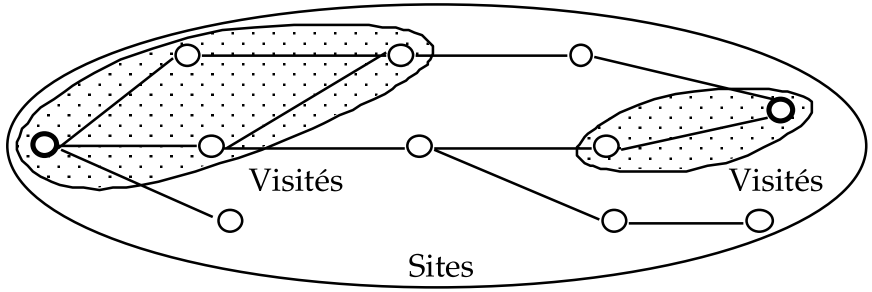
Motivations :

- **diffuser une information aux membres du groupe**

- **collecter un état global**

3.4 La Récursion Distribuée

```
method Rec( in Visités : set of object )
    Ajoutés : set of object;
begin
    if Visités G Sites then
        let Ajoutés in Sites-Visités;
        self.Pre;
        self.Rec( Visités U Ajoutés );
        self.Post;
    endif;
end;
```



Une étape de la récursion

3.5 Coordination inter-objets

Conclusion sur la coordination inter-objets

Notation

- documente les comportements de niveau groupe
- les décrit de façon concise

Passage à des comportements d'objets

=> guider le développeur par
un ensemble d'heuristiques

4 Coordination intra-objet

Aspects traités à ce niveau

- concurrence au sein d’un objet
- définition de politiques de synchronisation
- maintien de la cohérence des données partagées
- implantation des structures de contrôle de niveau groupe

Solution proposée

Un protocole méta-objet (CAOLAC)

au-dessus du langage à objets distribués GUIDE

4 Coordination intra-objet

Modèle d'objet de GUIDE :

- concurrent,
- passif,
- persistant,
- à parallélisme explicite avec le constructeur co_begin/co_end

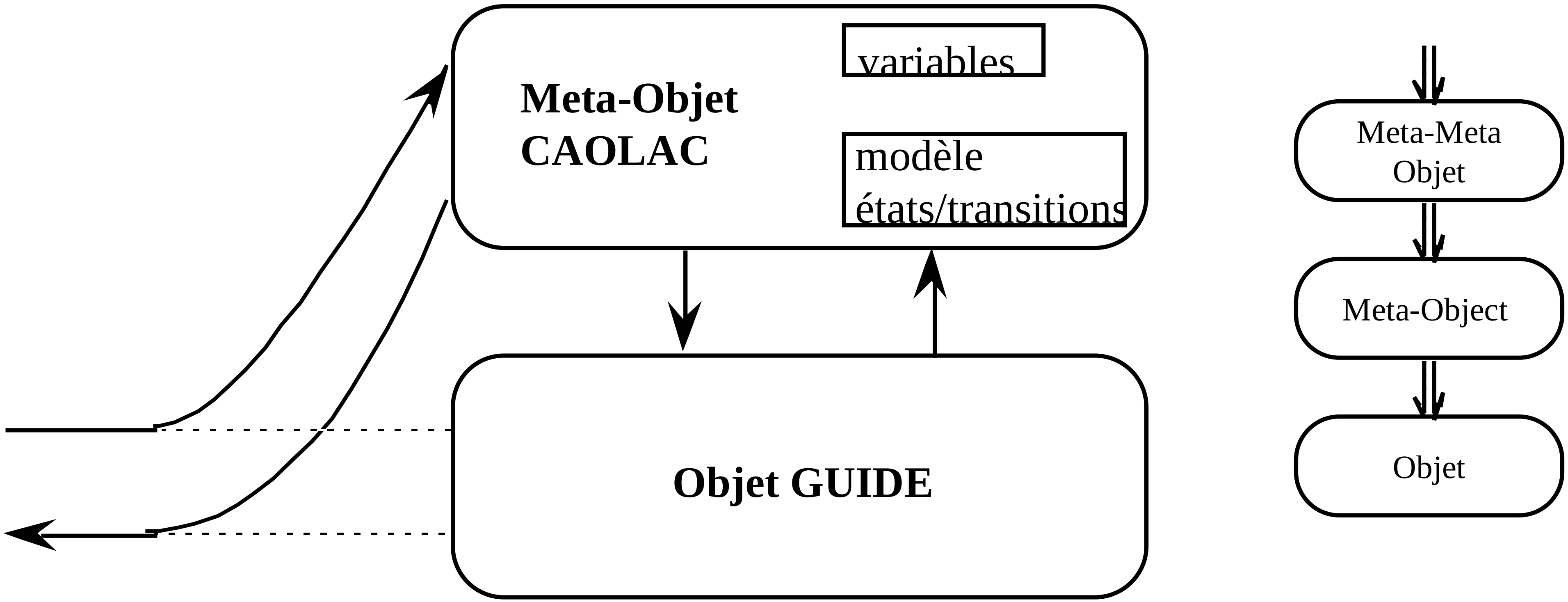
Pour synchroniser des objets concurrents :

- sémaphores	(C++)
- moniteurs (éventuellement réentrants)	(Java)
- gardes + compteur d'événements	(GUIDE)

CAOLAC

- ensemble d'états acceptables	(ACT++)
- protocoles méta-objet	(Open C++)

4 Coordination intra-objet

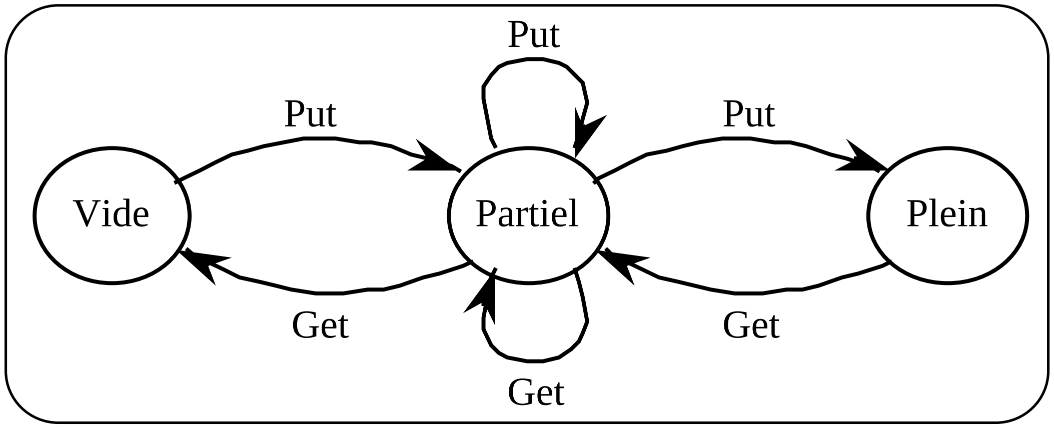


4 Coordination intra-objet

Tampon synchronisé

Un code de synchronisation

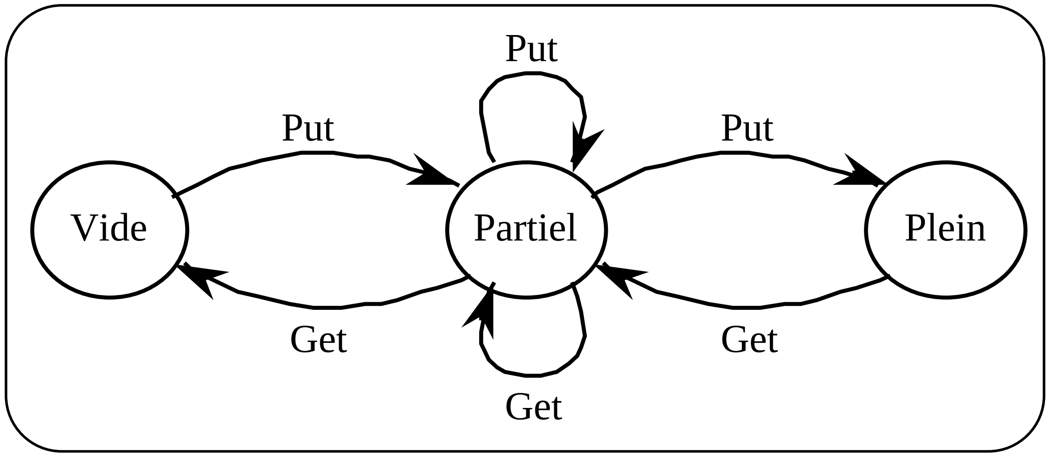
Un code des traitements



behaviour Buffer **{**
variables :
invocations :
Put**(IN** e **: REF** TElement **);**
Get **: REF** TElement**;**
methods :
IsFull **: Boolean;**
IsEmpty **: Boolean;**
initial state : Vide**;**
state Vide **{ ... }**
state Partiel **{ ... }**
state Plein **{ ... }**
}

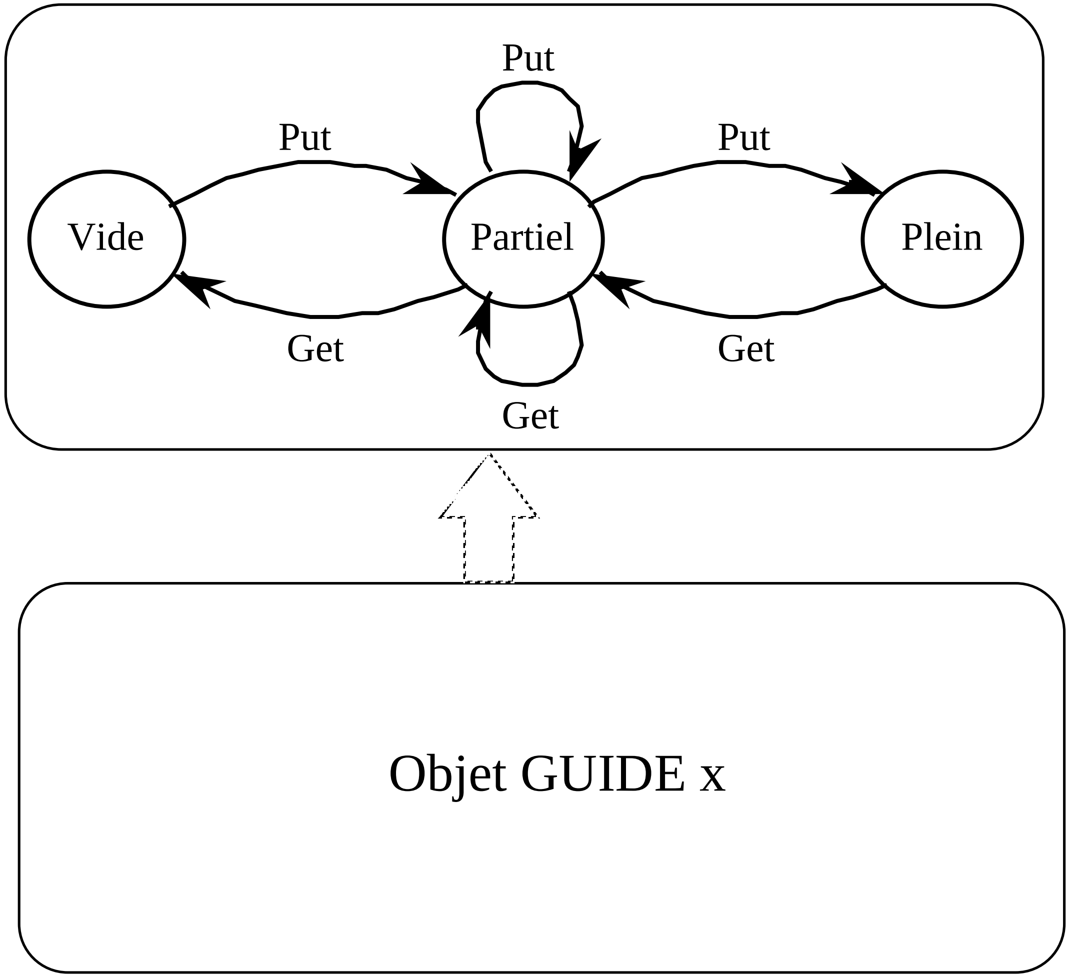
4 Coordination intra-objet

transition ≡ commande gardée
garde : invocation acceptée
+ condition évaluée
commande : instructions GUIDE




```
state Partiel {
    t1 {
        invocation( Put(e) );
        require( true );
        BASE.Put(e); return;
        if BASE.IsFull = true
        then become( Plein );
        else become( Partiel );
        end;
    }
    t2 {
        invocation( Get );
        ...
}   }
```

4 Coordination intra-objet



behaviour Buffer **{**
...
}

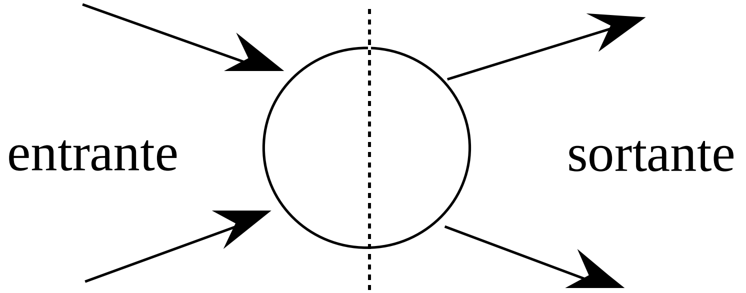
class File
  with behaviour Buffer **is**

   method Put**; ....**
   method Get**; ...**
   method IsFull**; ...**
   method IsEmpty**; ...**
end File**.**

class Pile
  with behaviour Buffer **is**
...
end Pile**.**

4 Coordination intra-objet

Sémantique des états



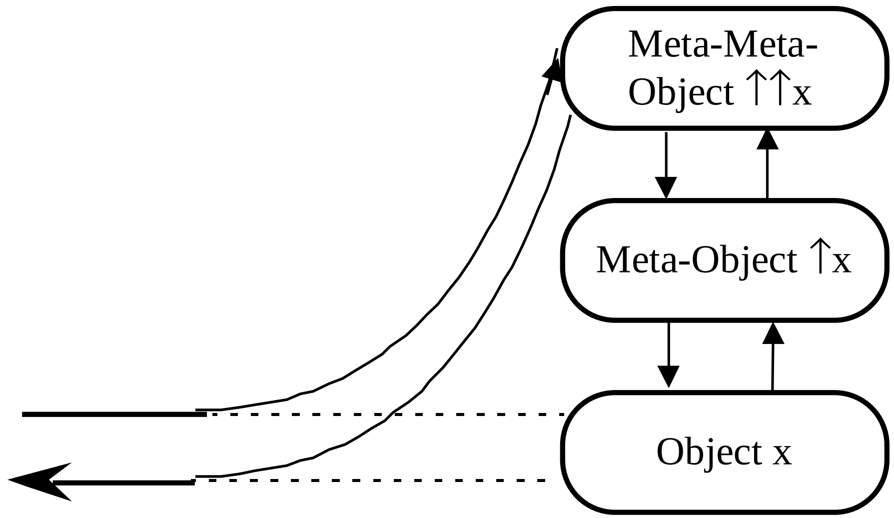
	transitions entrantes
nulle	une
rendez-vous	toutes

	transitions sortantes
séquentielle	une
parallèle	toutes
tant que + *condition*	toutes éventuellement plusieurs fois

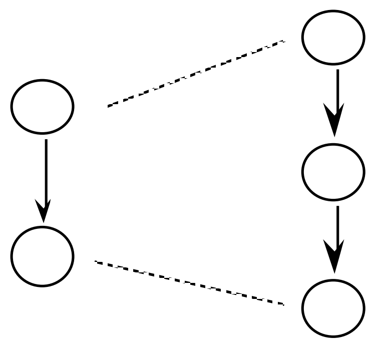
4 Coordination intra-objet

Deux relations entre comportements

Meta-comportements



Sous-comportements



4 Coordination intra-objet

Buts de ces deux relations

- augmenter le taux de réutilisation du code

- traiter les problèmes liés à l'association de l'héritage et de la synchronisation (*Inheritance Anomaly* [Yonezawa 93])

2 parties dans le code d'une méthode concurrente :
- le code de synchronisation (controle l'accès),
- le code "utile" (effectue le traitement).

3 causes limitent le niveau de réutilisation du code dans l'héritage
- la prise en compte de l'historique des exécutions,
- le partitionnement des états acceptables pour les méthodes,
- la modification des états acceptables.

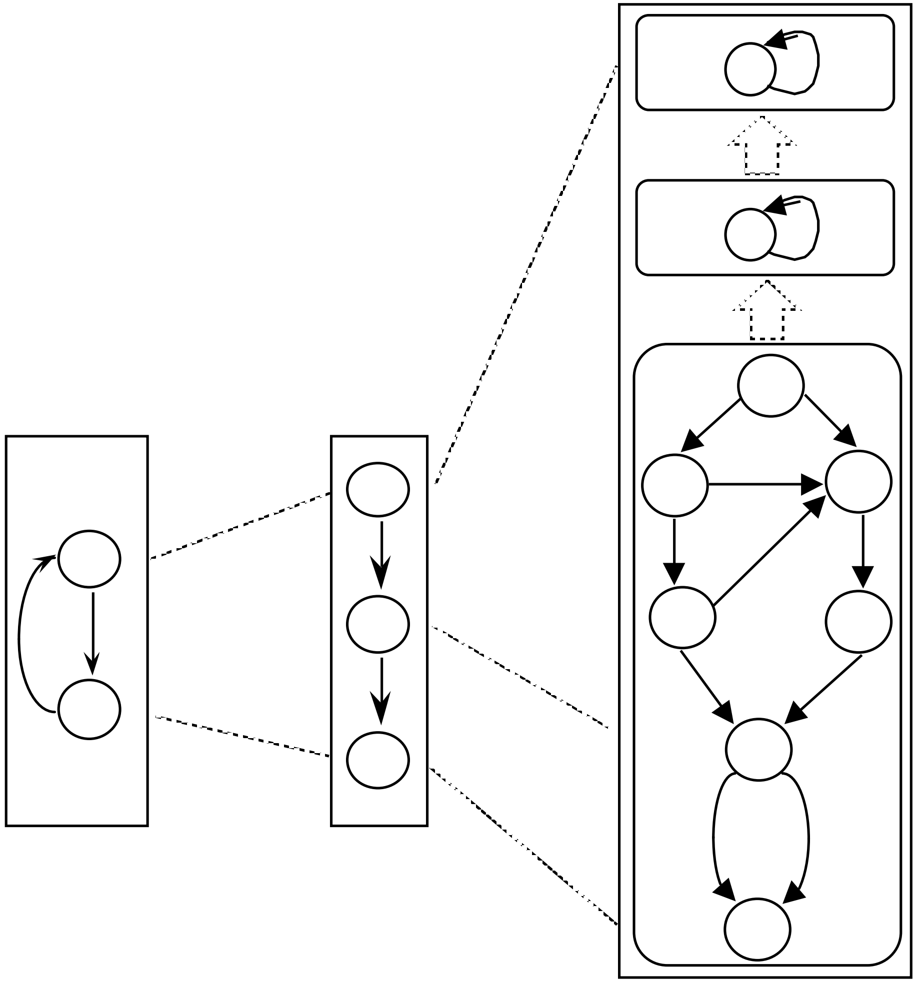
5 Exemple d'applications

Algorithme de calcul d'arbre couvrant

Coordination inter-objets
1 itération distribuée
2 phases raffinées en
2 phases récursions distribuées

Coordination intra-objet
une tour de 3 métas

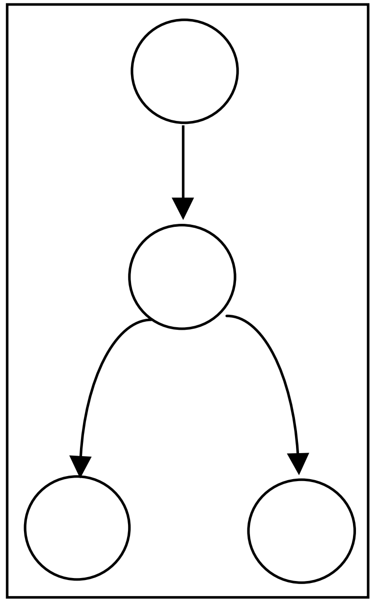
380 lignes de code CAOLAC
=> 800 lignes de code GUIDE
=> ~7000 lignes de code C



5 Exemple d'applications

Protocole de validation transactionnel à 2 phases

Coordination inter-objets
1 conditionnelle distribuée



Coordination intra-objet
2 sous-comportements
(coordinateur + participant)

100 lignes de code CAOLAC
=> 300 lignes de code GUIDE
=> ~2000 lignes de code C

6 Conclusion

Démarche de conception pour les applications distribuées

- descendante
- 3 niveaux (groupe, objet, méthode)

Un langage pour les programmes de groupe

- une approche à base de connaissance
- 4 structures de contrôle (phase, conditionnelle, itération, récursion)

Un protocole méta-objet au-dessus de GUIDE

6 Conclusion

Perspectives

Formaliser la définition du langage de groupe

Formaliser le raffinement - des connaissances
- des actions

Unifier les 3 niveaux

Définir d'autres structures de contrôle

Faire une implantation au-dessus d'un ORB CORBA

6 Conclusion

Références

[Bernstein 87] Bernstein, Hadzilacos, Goodman. Concurrency control and recovery in database systems. Addison-Wesley, 1987.

[Bonnet 94] Bonnet. Spécification et preuve d'algorithmes distribués en approche orientée objet. Mémoire d'ingénieur CNAM 1994.

[Halpern 95] Halpern, Fagin, Moses, Vardi. Reasoning about knowledge. MIT Press 1995.

[Yonezawa 93] Yonezawa, Matsuoka. Analysis of inheritance anomaly in object-oriented concurrent programming. In Research Directions in Concurrent Object-Oriented Programming. MIT Press 1993.

6 Conclusion

Publications

Seinturier, Duchien, Florin
*A Meta-Object Protocol for Distributed OO Applications. TOOLS USA'97

Seinturier, Duchien
* Group Behavior Patterns in an Object-Oriented Methodology for Distributed Applications. OOPSLA'96/Workshop MDO

Seinturier
* A Language for Distributed Behaviour Design of Cooperative Applications. ECOOP'96/Workshop DS
* Un Langage de Coordination pour les Applications Coopératives. RenPar'9

Bonnet, Duchien, Florin, Seinturier
* Some Specification and Proof Steps of a Spanning Tree Algorithm with an Object-Oriented Approach. FMOODS'96
* A Spanning Tree Object-Oriented Distributed Algorithm:Specification and Proof. MIW'96
* A Method for Specifying and Proving Distributed Algorithms. DIMAS'95