CHAP. 5 : PROTOCOLE "TWO-PHASE-COMMIT"

"exécution en deux phases"

exemple de description en LOTOS d'un protocole complexe

2PC = protocole pour la "communication de groupe"

- *utilisé pour la mise à jour de BD réparties*
- *tolère des défaillances temporaires des processeurs*

Principes :

- les utilisateurs diffusent des messages (mises à jour, par exemple)

• si une diffusion réussit, tous les utilisateurs du groupe ont reçu au moins une fois le message

• si une diffusion échoue, aucun utilisateur du groupe n'a pris en compte le message

HYPOTHÈSES SUR L'ENVIRONNEMENT

• On dispose d'un moyen de communication (réseau) fiable

• *Le groupe est statique*

• L'ordre de livraison des messages n'a pas à être garanti

• *Le fait de recevoir des messages dupliqués n'est pas un problème*

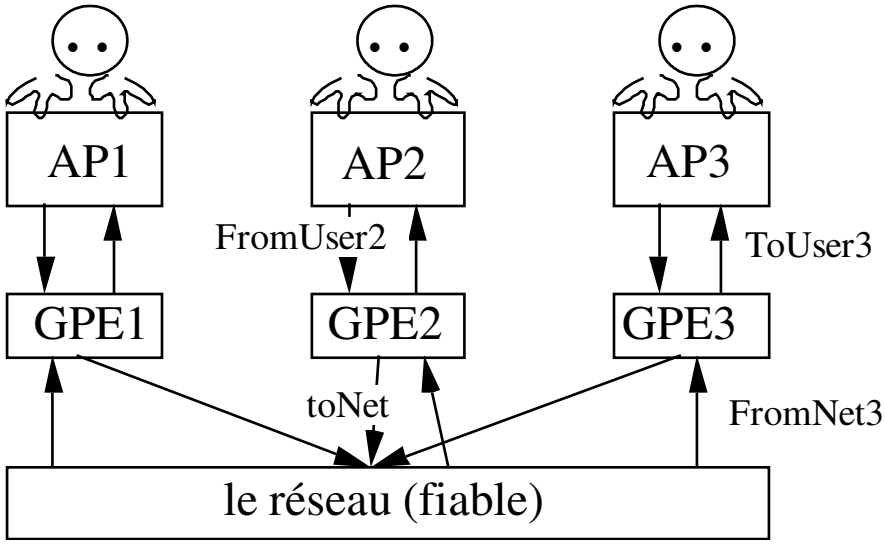
CHOIX D'ARCHITECTURE : GPE

GPE = "Group Protocol Entity"

• composants logiciels qui font l'interface entre les utilisateurs et le réseau

- pour le déclenchement d'une diffusion
- pour la gestion des diffusions en provenance des autres utilisateurs

Exemple avec un groupe de 3 utilisateurs



• quand l'utilisateur i veut faire une diffusion, il le demande à son GPE (GPE_i) qui devient le "coordinateur" de cette diffusion

• *Phase 1* : le GPE_i envoie le message à tous les autres GPE en leur demandant si ils peuvent le recevoir

• les autres répondent.

• *Phase 2* :

- si ils ont tous répondu, le GPE_i envoie le "**commit**" à tous ; ils exécutent le message

- si ils n'ont pas tous répondu (time-out), le GPE_i envoie "**abort**" à tous les autres ; le message est abandonné

SPÉCIFICATION DES PROPRIÉTÉS DEMANDÉES

• ***Consistance uniforme*** : si un message est livré à un utilisateur, il sera un jour livré à tous les autres

• *Terminaison* : toute diffusion atteint un jour l'état "commit" ou l'état "abort"

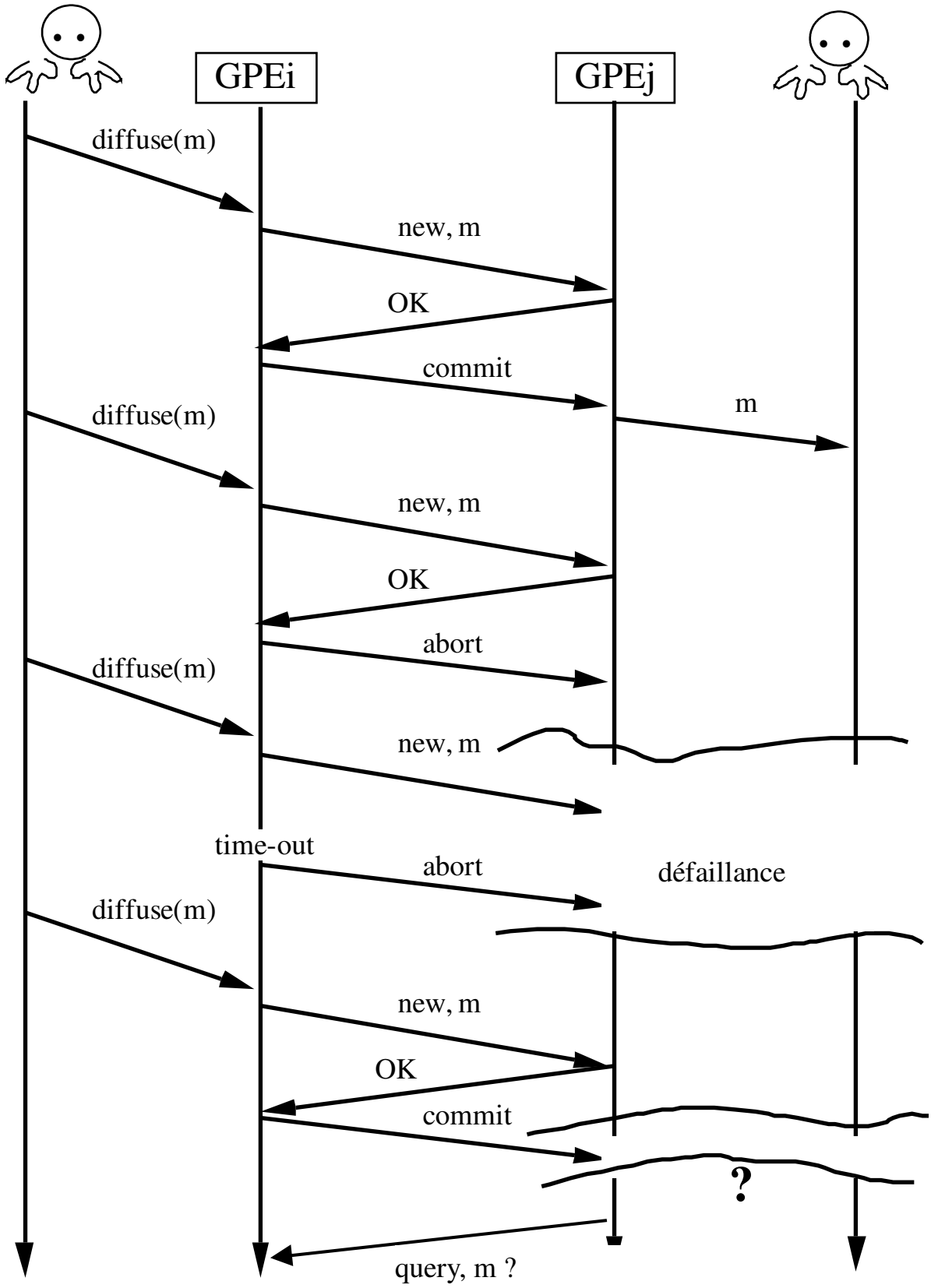
• *Correction* (1) : si une diffusion atteint l'état "abort", le message n'est livré à aucun utilisateur

• *Correction* (2) : si une diffusion atteint l'état "commit", le message est livré à tous les utilisateurs

• ***Garantie de service*** : si tous les utilisateurs sont en fonctionnement entre une demande de diffusion et sa terminaison, la diffusion atteint l'état "commit"

• *Pas de diffusions spontanées* : si un message est livré à un utilisateur, c'est que sa diffusion a été demandée

SCÉNARIOS

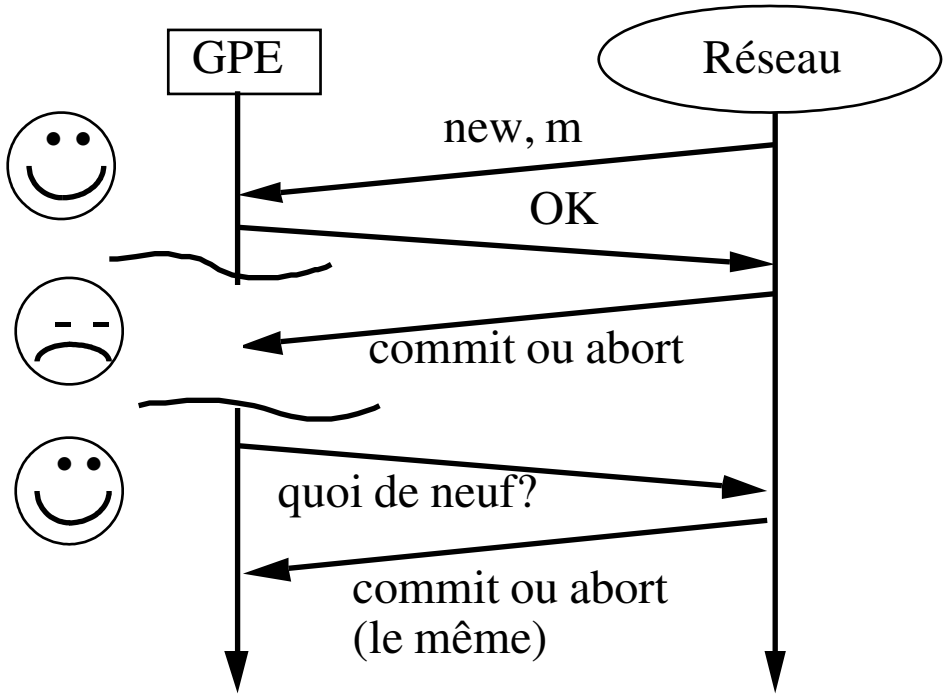

GPEi
GPEj
diffuse(m)
new, m
OK
commit
m
diffuse(m)
new, m
OK
abort
diffuse(m)
new, m
time-out
abort
défaillance
diffuse(m)
new, m
OK
commit
?
query, m ?

TOLÉRANCE AUX DÉFAILLANCES TEMPORAIRES

de type *"fail/silent"*

PB : Après une défaillance, un couple <application, GPE> qui repart doit récupérer tous les "commit" ou "abort" qu'il attendait et qui ont eu lieu pendant sa défaillance

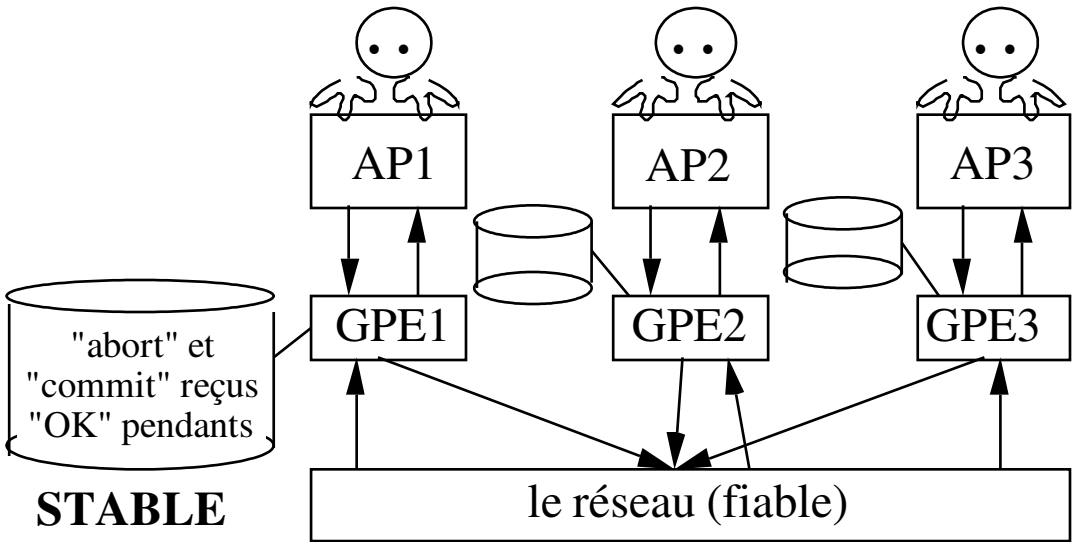
Condition : au moins un autre GPE a fonctionné!



• On doit donc *garder trace des "OK"* (messages "pendants") pour pouvoir se renseigner…

• On doit aussi *garder trace des "commit" et "abort"* pour pouvoir renseigner les autres!

ARCHITECTURE FINALE



Question posée :

Donner une spécification des GPE qui satisfait les propriétés demandées sous les hypothèses faites

Démarche :

par “brouillons” successifs

PREMIER BROUILLON :

le GPE dans un contexte à trois utilisateurs (5 serait mieux) et le réseau

BUT : *répertorier les ports et les synchronisations*

---

behaviour

(User**[ToUser1, FromUser1]** *?*

GPE**[ToUser1, FromUser1, ToNet, FromNet1]**)

?

(User**[ToUser2, FromUser2]** *?*

GPE**[ToUser2, FromUser2, ToNet, FromNet2]**)

?

(User**[ToUser3, FromUser3]** *?*

GPE**[ToUser3, FromUser3, ToNet, FromNet3]**)

?

Network**[ToNet, FromNet1, FromNet2, FromNet3]**

---

where …

- Les couples (User, GPE) ne sont pas synchronisés entre eux
- Ils se synchronisent avec le réseau
- Le $User_i$ communique uniquement avec le GPE_i

ORGANISATION GÉNÉRALE

---

behaviour

((**hide ToUser1, FromUser1 in**

User**[ToUser1, FromUser1]** |**[ToUser1, FromUser1]** |

GPE**[ToUser1, FromUser1, ToNet, FromNet1]**)

|||

(**hide ToUser2, FromUser2 in**

User**[ToUser2, FromUser2]** |**[ToUser2, FromUser2]** |

GPE**[ToUser2, FromUser2, ToNet, FromNet2]**)

|||

(**hide ToUser3, FromUser3 in**

User**[ToUser3, FromUser3]** |**[ToUser3, FromUser3]** |

GPE**[ToUser3, FromUser3, ToNet, FromNet3]**))

| [ToNet, FromNet1, FromNet2, FromNet3] |

Network**[ToNet, FromNet1, FromNet2, FromNet3]**

where

…

Il manque sans aucun doute des paramètres.

Reste à décrire sommairement User et Network, *du point de vue du GPE*

DEUXIÈME BROUILLON :

Modèle du réseau (vu d'un GPE)

- Le réseau transmet plusieurs messages en parallèle

---

process Network **[ToNet, FromNet1, FromNet2, FromNet3]**

: **noexit** :=

(ToNet?msg: … ;

([ ] […] -> FromNet1!msg

[ ] […] -> FromNet2!msg

[ ] […] -> FromNet3!msg)

)

||| Network **[ToNet, FromNet1, FromNet2, FromNet3]**

endproc

- Question : comment trouver le destinataire d'un message?
- Réponse : définir le type du message avec une opération qui le donne ;

donc : définition d'un type GPEaddr pour ce résultat.

QUELQUES TYPES DE DONNÉES ET ÉBAUCHES DE TYPES

---

type GPEaddr **is** Boolean

sorts GPEaddr

opns

GPEaddr1, GPEaddr2, GPEaddr3 : —> GPEaddr

_eq_, _ne_ : GPEaddr, GPEaddr —> Bool

eqns ofsort Bool

X eq X = true ;

GPEaddr1 eq GPEaddr2 = false

...

endtype

---

type MsgStat **is** Boolean, GPEaddr

sorts Msg

opns

setDest : Msg, GPEaddr —> Msg

getDest : Msg —> GPEaddr

...

---

eqns ofsort GPEaddr

getDest(setDest(M, GAD)) = GAD

…

endtype

NB : types très incomplets

LE RÉSEAU, VERSION COMPLÉTE

```
process Network [ToNet, FromNet1, FromNet2,
FromNet3] : noexit :=
(ToNet?msg: Msg ;
    (    [getDest(msg) eq GPEaddr1] ->
FromNet1!msg
    [ ]
        [getDest(msg) eq GPEaddr2]->
FromNet2!msg
    [ ]
        [getDest(msg) eq GPEaddr3] ->
FromNet3!msg          )
)
|||
Network [ToNet, FromNet1, FromNet2, FromNet3]
endproc
```

L'utilisation de ||| est un choix, qui permet d'indiquer qu'à tout moment le réseau accepte un message

TROISIÈME BROUILLON : USER

(Modèle d'un utilisateur vu d'un GPE)

---

process User **[ToUser, FromUser] (nbMsg : Nat) : noexit**

:=

(ToUser?m:Msg ;

User **[ToUser, FromUser] (nbMsg)**

[ ]

i ; (User **[ToUser, FromUser] (nbMsg)**

[ ]

FromUser?m:Msg [getSeq(m) eq nbMsg];

User **[ToUser, FromUser] (succ(nbMsg))**)

)

endproc

Pour un GPE, un "utilisateur" est une entité qui peut recevoir des messages, ou vaquer à des activités internes. De temps en temps, il échange un message numéroté avec le GPE.

NB : ajout à faire au type MsgStat (getSeq)

=> NOUVELLE ORGANISATION GÉNÉRALE :

- Les utilisateurs sont paramétrés par leur numéro de message
- Chaque GPE doit connaître sa propre adresse et celles des autres GPE

INTÉGRATION À L'ORGANISATION GÉNÉRALE

---

behaviour

let ***group: List = add(add(add(nil, GPEaddr1), GPEaddr2), GPEaddr3)*** **in**

((**hide ToUser1, FromUser1 in**

User**[ToUser1, FromUser1] (0 of Nat)**

| [ToUser1, FromUser1] |

GPE**[ToUser1, FromUser1, ToNet, FromNet1]**

(GPEaddr1, remove(GPEaddr1, group))

|||

(**hide ToUser2, FromUser2 in**

User**[ToUser2, FromUser2] (0 of Nat)**

| [ToUser2, FromUser2] |

GPE**[ToUser2, FromUser2, ToNet, FromNet2]**

(GPEaddr2, remove(GPEaddr2, group))

|||

(**hide ToUser3, FromUser3 in**

User**[ToUser3, FromUser3] (0 of Nat)**

| [ToUser3, FromUser3] |

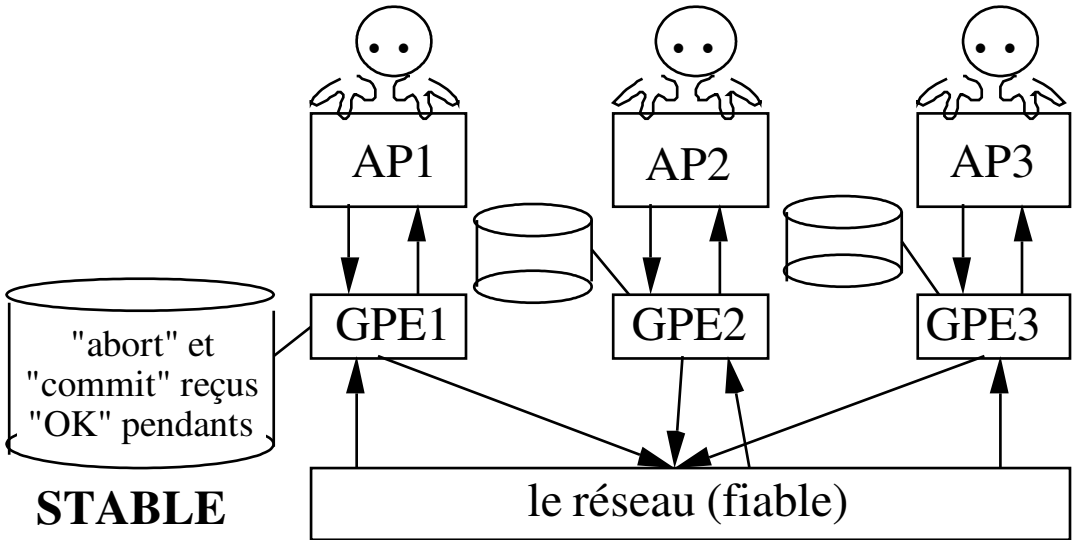
GPE**[ToUser3, FromUser3, ToNet, FromNet3]**

(GPEaddr3, remove(GPEaddr3, group) **))**

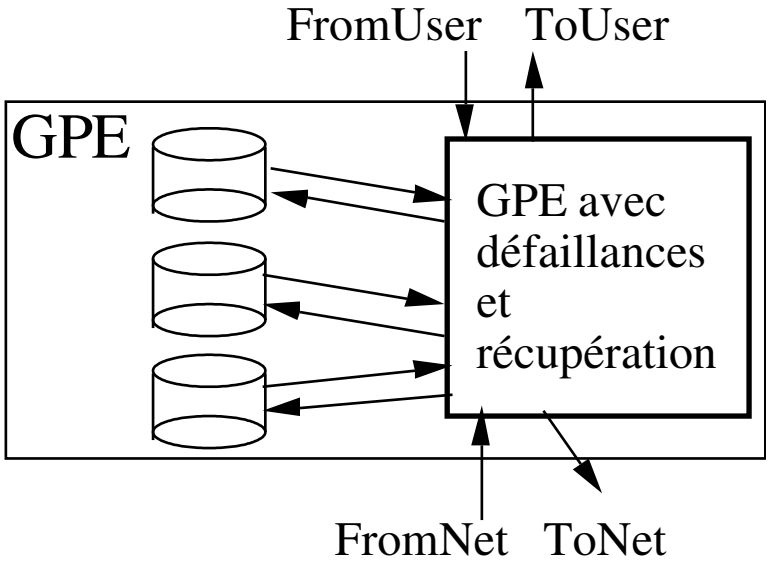
| [ToNet, FromNet1, FromNet2, FromNet3] |

Network**[ToNet, FromNet1, FromNet2, FromNet3]**

where …


AP1
AP2
AP3
GPE1
GPE2
GPE3
"abort" et
"commit" reçus
"OK" pendants
STABLE
le réseau (fiable)

SPÉCIFICATION D'UN GPE



Ici, on va voir un fichier comme un processus avec un contenu (paramètre) qu'on peut changer, ou qu'on peut consulter

(pas réaliste du tout, mais simple pour le cours…)

---

process MsgFile**[in, out]***(old: MsgDict)*:**noexit** :=
(in?new: MsgDict ; MsgFile**[in, out]***(new)*
[ ]
out!old ; MsgFile**[in, out]***(old)*)
endproc

UN GPE "ARMÉ CONTRE LES DÉFAILLANCES" AVEC SES FICHIERS LOCAUX

---

process GPE**[ToUser, FromUser, FromNet, ToNet]**

(moi:GPEaddr, lesAutres:List) : **noexit** :=

hide **ReadPending, WritePending, ReadOutcome,**

WriteOutcome, … in

(MsgFile**[ReadPending, WritePending]** *(emptyD)*

|||

MsgFile**[ReadOutcome, WriteOutcome]** *(emptyD)*

||| …)

|**[ReadPending, WritePending, ReadOutcome, WriteOutcome, …]**|

GPErecup**[ToUser, FromUser, FromNet, ToNet, ReadPending, WritePending, ReadOutcome, WriteOutcome, …]** *(moi, lesAutres)*

where

process MsgFile**[in, out]***(old: MsgDict)*:**noexit** :=

(in?new: MsgDict ; MsgFile**[in, out]***(new)*

[ ]

out!old ; MsgFile**[in, out]***(old)*)

---

endproc

GPErecup[…] est à spécifier

LE GPE "DÉSARMÉ" QUI DÉFAILLE ET RÉCUPÉRE

---

process GPErecup[**ToUser, FromUser, FromNet, ToNet, ReadPending, WritePending, ReadOutcome, WriteOutcome, …**] *(moi : GPEaddr, lesAutres: List)* : **noexit** :=

GroupProtocol [**ToUser, FromUser, FromNet, ToNet, ReadPending, WritePending, ReadOutcome, WriteOutcome, …**] *(moi, lesAutres)*

[>

Failure [**ToUser, FromUser, FromNet, ToNet, ReadPending, WritePending, ReadOutcome, WriteOutcome, …**] *(moi, lesAutres)*

where

process GroupProtocol …

…

endproc

---

process Failure …

…

endproc

MODÈLE DES DÉFAILLANCES

process Failure **[ToUser, FromUser, FromNet, ToNet, ReadPending, WritePending, ReadOutcome, WriteOutcome, …]** ***(moi : GPEaddr, lesAutres: List)*** : **noexit** :=

*(*çà peut repartir : *)*

(**i** ; Recup **[ToUser, FromUser, FromNet, ToNet, ReadPending, WritePending, ReadOutcome, WriteOutcome, …]** ***(moi, lesAutres)***

[ ]

(*(*çà peut persévérer dans la défaillance : *)*

i ; Failure **[ToUser, FromUser, FromNet, ToNet, ReadPending, WritePending, ReadOutcome, WriteOutcome, …]** ***(moi, lesAutres)***

[ ] *(*mais **sans bloquer le réseau** dont on accepte les messages*)*

FromNet?msg:Msg [getKind(msg) neq NEW] ;

*(*mais sans rien en faire!*)*

Failure **[ToUser, FromUser, FromNet, ToNet, ReadPending, WritePending, ReadOutcome, WriteOutcome, …]** ***(moi,lesAutres)***

[ ] *(*sauf pour "NEW": on envoie un time_out*)*

FromNet?msg:Msg [getKind(msg) eq NEW] ;

ToNet!setKind(msg, NOK) ; *(*time out: ?m:Msg [getKind(m) eq NOK]*)*

Failure **[ToUser, FromUser, FromNet, ToNet, ReadPending, WritePending, ReadOutcome, WriteOutcome, …]** ***(moi,lesAutres)***

))

endproc

ET MAINTENANT LE PROTOCOLE

process GroupProtocol **[ToUser, FromUser, FromNet, ToNet, ReadPending, WritePending, ReadOutcome, WriteOutcome, …]** ***(moi : GPEaddr, lesAutres: List)*** : **noexit**

:=

ReadPending?pending:MsgDict ;

ReadOutcome?outcome:MsgDict ;

…

(FromUser?msg:Msg ; *(*nouvelle diffusion à faire*)*

…

[ ]

FromNet?msg:Msg ; *(*le réseau diffuse*)*

([getKind(msg) eq OK or getKind(msg) eq NOK]

—> …

[ ]

[getKind(msg) eq NEW] —> …

[ ]

[getKind(msg) eq COMMIT] —> …

[ ]

```
        [getKind(msg) eq ABORT] —> …
      [ ]
        [getKind(msg) eq QUERY] —> … ) )
where
      …
endproc
```

À COMPLÉTER (± EN TD)

- définition d'un type Kind avec OK, NOK, NEW, COMMIT, ABORT, QUERY
- ajout au type MsgStat des opérations setKind et getKind, et d'autres …
- définition du type MsgDict avec des opérations de recherche, d'ajout, de mise à jour, de suppression

- spécification des différents choix dans GroupProtocol

- gestion des votes
- spécification de la diffusion

• spécification de Récup