CTOP[1]: un service de diffusion ordonnée causalement et totalement fonctionnant sur Internet

Christian Toinard

CNAM-CEDRIC 292 rue Saint-Martin 75141 Paris Cedex 03
toinard@cnam.fr; http://cedric.cnam.fr/~toinard

RÉSUMÉ — Le principe d'un arbre de diffusion vers des groupes, garantissant un ordre causal et total grâce à un résultat formel, provient de [Toinard 93]. CTOP (*Causally and Totally Ordered Protocol*) applique ce principe à l'Internet et permet d'éviter complètement la construction d'une couche causale de communication. Pour chaque groupe, le système positionne une racine de diffusion de façon optimale afin de couvrir tous les membres du groupe. Un composant de routage permet d'acheminer un message seulement vers les entités concernées. Chaque entité de l'arbre couvrant le groupe maintient une connaissance de l'appartenance au groupe. L'état à chaque nœud est minimal. Les états inutiles sont supprimés de façon cohérente. On peut montrer que la diffusion est identifiable. L'émetteur appartenant au groupe et la diffusion étant localement ordonnée (livraison en séquence), le résultat formel garantit un ordre causal et total. Les identifiants ne sont ni connus, ni transportés par le système. Ce système est implanté sur une pile TCP/IP standard. Son administration et son utilisation sont simples. Il offre une API qui permet de construire des applications réparties au moyen de groupes d'entités.
MOTS-CLÉS: diffusion, groupes, ordre, appartenance aux groupes, routage, simulation sur l'Internet.

ABSTRACT — A multicast tree, guarantying a causal and total order with a formal result, comes from [Toinard 93]. CTOP (*Causally and Totally Ordered Protocol*) applies the principle to the Internet and avoids building a causal layer of protocol. For each multicast group, the system maintains a multicast root covering all of the group members. Routing minimizes the crossed nodes. Each entity of the multicast tree maintains the group membership. Each entity has the knowledge of the group membership. State at each node is minimal. Out of date states are discarded consistently. We prove that the multicast is identifiable. With an emitter belonging to the group and a local order (FIFO), our formal result guaranties a causal and total order. Identifiers are neither known nor transmitted. System uses a classical TCP/IP stack. Administration and use are easy. The system provides an application-programming interface to develop distributed applications using groups.
KEY WORDS: multicast, groups, ordering, group membership, routing, simulation over the Internet.

[1] Ce travail est soutenu par la communauté européenne sous le contrat IST AIT VEPOP numéro IST-1999-13346 comprenant EADS Corporate Research Lab., EADS Airbus, CNAM-CEDRIC, Flow-Master, University of Oulu

1. INTRODUCTION

Un message diffusé vers un groupe (multicast) est destiné à un sous-ensemble d'entités (entité d'application ou processus système) qui forment le groupe destinataire (multicast group). Nous considérons que les groupes peuvent s'intersecter et que n'importe quelle entité peut émettre un message vers un groupe donné. Nous considérons qu'il n'y a pas de temps global dans le système. Ces hypothèses correspondent au cas le plus général de communication de groupe puisqu'il n'y a aucune contrainte sur la composition du groupe et la nature de l'émetteur.

Avec un ordre local aussi appelé ordre FIFO, les messages sont livrés en préservant l'ordre d'émission. Avec un ordre causal, les relations causales de communication entre les messages sont respectées. Un ordre total garantit que tous les destinataires reçoivent les messages dans le même ordre. De façon classique, pour obtenir un ordre causal et total, deux couches de communication sont nécessaires. L'ordre total est construit en utilisant les services de communication causale. La couche causale nécessite de transmettre une histoire causale qui peut être grande spécialement lorsque les groupes sont quelconques et peuvent s'intersecter. La construction de l'ordre total n'est pas simplifiée par la couche causale. [Moser 96] [Birman 91] [Renesse 95] [Amir 92] [Mishra 93] [Malloth 96] utilisent cette méthode. Ces systèmes permettent différentes qualités d'ordre. Certaines améliorations permettent de réduire la taille de l'histoire causale transmise dans les messages. Certains protocoles [Verissimo 89] [Moser 96] évitent la construction d'un ordre causal dans le cas particulier de la diffusion vers un seul groupe (broadcast) correspondant à toutes les entités (tout le monde). Cependant un résultat général obtenu dans [Florin 92] prouve le caractère causal de tous les protocoles de type broadcast qui construisent un ordre total et local. Dans le cas général de la diffusion vers des groupes différents (multicast), il est plus difficile d'éviter la construction d'un ordre causal. Actuellement, la plupart de ces protocoles utilisent une structuration en arbre mais dans une optique différente. Ils cherchent à limiter les surcoûts engendrés par l'ordre causal. En général, on trouve un ordre causal réalisé dans un domaine de diffusion [Huleilhel 96][Moser 96][Baldoni 96]. Le message doit ensuite être ordonné causalement vis à vis des autres domaines. Enfin, un ordre total doit être construit en demandant un numéro de séquence à la racine de l'arbre [Huleihel 96] ou en attendant des messages des autres domaines [Moser 96] pour ordonner les messages concurrents. Donc, on trouve deux voir trois [Moser 96] couches de protocoles. Chacune de ces couches introduit des attentes pour réordonner les messages et les temps de livraison se trouvent allongés d'autant. A l'inverse, dans [Toinard 93] [Toinard 99] on trouve un ensemble de résultats formels qui permettent d'éviter la construction de la couche causale dans le cas général de groupes différents pouvant s'intersecter (multicast). Ces résultats ont permis de définir les principes d'un protocole arborescent [Toinard 93] qui évite un ordre causal. Il n'y a alors plus qu'une seule couche de protocole.

Dans ce papier, les résultats formels qui ont permis d'établir le principe de la solution sont d'abord présentés. L'algorithme correspondant à la mise en oeuvre du protocole sur l'Internet est ensuite défini. Les entités construisent un arbre au moyen de leur extrémité réseau (adresse IP et numéro de port). Une entité qui veut s'installer dans l'arbre existant envoie une requête d'installation à l'extrémité réseau d'une autre entité qui devient son père. L'entité qui s'installe reçoit de son père un nom en notation pointée qui définit la position de cette entité dans l'arbre. Ces noms pointés servent essentiellement au composant de routage pour établir de façon optimale l'arbre couvrant. Ils sont aussi utilisés pour définir la composition d'un groupe. Ensuite, chaque entité peut demander à entrer dans un groupe en donnant le nom du groupe. Si le nom n'existe pas déjà, l'entité devient racine de diffusion pour ce groupe. La demande d'entrée d'un nouveau membre peut conduire à repositionner la racine du groupe pour couvrir tout le groupe. Le repositionnement de la racine du groupe sur un événement d'entrée repose sur le parcours de la requête d'entrée en remontant l'arbre jusqu'à atteindre la racine courante. La racine du groupe transmet alors une indication d'entrée qui parcourt l'arbre de façon minimale pour mettre à jour les listes d'acheminement et les informations d'appartenance au groupe. Une entité membre d'un groupe peut émettre un message vers ce groupe. Ce message est envoyé à la racine du groupe et il parcourt l'arbre de façon minimale. Une entité peut demander à la racine du groupe de sortir. L'indication de sortie parcourt l'arbre toujours de façon minimale pour mettre à jour les listes d'acheminement et l'appartenance au groupe. Le parcours des indications d'entrée, indications de sortie et messages

est rendu minimal par le composant de routage en utilisant l'appartenance au groupe et la notation pointée. Un groupe étant un ensemble de noms pointés, chaque entité se sert localement des noms pointés pour calculer la position de la nouvelle racine et déterminer quels fils acheminent le groupe.

D'un point de vue architecture réseau et système, CTOP comprend une couche de transport fiable qui est soit TCP soit UDP. Dans le cas d'UDP, une fiabilité des transmissions est effectuée par CTOP. CTOP comprend un composant de routage qui maintient la racine du groupe et les listes d'acheminement. Un composant d'appartenance est renseigné par les indications d'entrée et de sortie. L'ordonnancement est garanti par preuve. Il n'y a donc aucun mécanisme particulier pour réordonner les messages au moyen de file d'attente. En fait, un acheminement arborescent fournit naturellement un ordre total pour peu que la racine de diffusion soit positionnée de façon à couvrir les destinataires. L'ordre causal est garanti par preuve grâce à un de nos résultats formels.

Une première partie rappelle les définitions et les résultats formels que nous avons obtenus. Une seconde partie détaille le principe de CTOP. Nous présentons la construction de l'arborescence des entités, le positionnement sur cette arborescence des différentes racines de diffusion ainsi que l'algorithme permettant de gérer les informations de routage et d'appartenance aux groupes. Une troisième partie donne le schéma de preuve qui permet de montrer grâce aux résultats formels la propriété d'ordre causal et total. La preuve complète se trouve dans [Toinard 93].

2. DÉFINITIONS ET RÉSULTATS FORMELS

2.1. Notations de base

Nous présentons ici une typologie des diffusions ordonnées permettant de définir les notions de livraison ordonnée couramment rencontrées dans la littérature. Nous distinguons les entités d'application des entités de protocoles. Une entité d'application correspond typiquement à un processus système qui utilise les services de diffusion vers des groupes. Tandis qu'une entité de protocole correspond à un processus système qui réalise le service de diffusion pour le compte d'une ou plusieurs entités d'application.

Groupe: une entité d'application envoie un message vers un sous-ensemble d'entités d'application que l'on appelle le groupe et qui est noté *G(M)*. Un groupe correspond à la notion classique sur l'Internet de multicast. Pour nous un groupe est désigné par un nom textuel (ex: *groupeSpécificationAérodynamisme*).
Groupe d'adressage: pour atteindre le groupe *G(M)* le message *M* est envoyé sur le réseau à destination d'un groupe d'entités de protocole noté *A(G(M))*. *A(G(M))* correspond à l'ensemble des processus système qui vont traiter le message pour le livrer aux entités d'application correspondantes.
Evénements: nous considérons les événements $E^X(M)$et $D^Y(M)$ qui apparaissent sur les entités d'application *x* et *y*. $E^X(M)$ est l'émission du message *M* par l'entité d'application *x*. $D^Y(M)$ est la réception par l'entité $y \in G(M)$ du message *M*. Typiquement, une entité de protocole $j \in A(G(M))$ réordonne les messages avant de les livrer au destinataire $y \in G(M)$.

2.2. Propriétés d'ordre

Différentes propriétés de livraison ordonnée sont définies en utilisant cinq relations entre les messages: locale, causale directe, causal, identifiable ainsi qu'une relation de livraison ordonnée. La plupart de ces définitions utilisent la relation entre les événements "apparaît avant" (*happened before relation*) notée $\rightarrow$.

Relation de livraison ordonnée: lorsque deux messages *M* et *M'* sont reçus dans le même ordre par toutes les entités d'application destinataires de ces deux messages, on dit que *M* et *M'* présentent une relation de livraison ordonnée $\rightarrow_{do}$.

$$M \rightarrow_{do} M' \Leftrightarrow \forall z \in G(M) \cap G(M'), D^{Z}(M) \rightarrow D^{Z}(M')$$

Relation locale: nous définissons une relation locale $\rightarrow_{l}$ lorsqu'une entité d'application *x* émet un message *M* avant d'émettre un message *M'*.

$$M \rightarrow_{l} M' \Leftrightarrow \exists x, E^{X}(M) \rightarrow E^{X}(M')$$

Ordre local: lorsque le système garantit une livraison selon un ordre local, les messages sont livrés selon leurs relations locales.

$$M \rightarrow_{l} M' \Rightarrow M \rightarrow_{do} M'$$

Relation de causalité directe: une telle relation existe, lorsqu'une entité *x* reçoit un message *M* avant d'émettre un message *M'*.

$$M \rightarrow_{dc} M' \Leftrightarrow \exists x, D^{X}(M) \rightarrow E^{X}(M')$$

Relation causale: une relation causale équivalente à [Lamport 78] est la fermeture transitive des relations locales et causales directes.

$$M \rightarrow_{c} M' \Leftrightarrow [\exists M_0 = M, \ldots, M_{\beta+1}, \beta \geq 0]: M_{q-1} \rightarrow_{l} M_q \vee M_{q-1} \rightarrow_{dc} M_q$$

Ordre causal: lorsque le système garantit une livraison selon un ordre causal les messages sont livrés selon leurs relations causales.

$$M \rightarrow_{c} M' \Rightarrow M \rightarrow_{do} M'$$

Sur la figure 1, *M* précède localement *M'* car *x* émet *M* avant *M'*. *M'* précède *M"* selon la relation de causalité directe car *y* reçoit *M'* avant d'émettre *M"*. Donc par transitivité, *M* précède causalement *M"*. Comme la diffusion garantit un ordre causal, *z* reçoit *M* avant *M"*. Si *M"* arrive sur le réseau avant *M*, les deux messages vont être réordonner afin de respecter la relation causale.

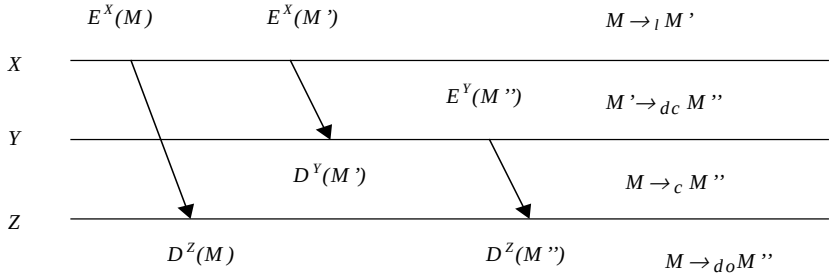

Figure 1. Ordre causal

Ordre total: un protocole qui livre tout couple de messages dans le même ordre à tous les destinataires, garantit un ordre total selon l'appellation classique.

$\forall M, M', M \neq M', M \rightarrow_{do} M' \vee M' \rightarrow_{do} M$

Il est communément admis qu'un ordre total ne satisfait pas les relations causales. La figure 2 illustre un protocole qui garantit un ordre total puisque toutes les entités d'application reçoivent les messages dans le même ordre. Cependant, la relation causale n'est pas respectée. Donc il est classiquement connu qu'un ordre total n'est pas causal.

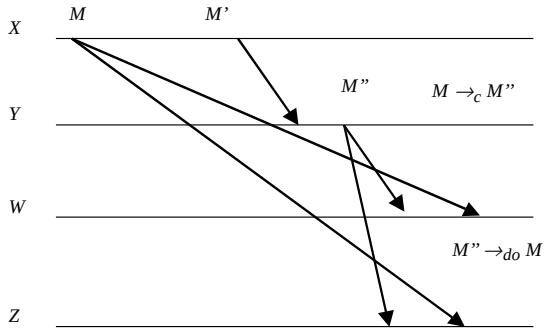

Figure 2. Ordre total

Relation d'identifiants: cette notion a été définie initialement dans [Toinard 93]. Selon le mode de fonctionnement du système, nous pouvons définir une relation *id* qui attribue à chaque message un identifiant *id(M)* pris dans un espace $\mathcal{I}$d. $\mathcal{I}$d est muni d'une relation $<_{id}$ telle que $\leq_{id}$ est une relation d'ordre sur l'espace $\mathcal{I}$d ($\leq_{id}$ est réflexive, antisymétrique et transitive). La relation d'identifiants $\rightarrow_{id}$ (non réflexive, non symétrique, transitive) est définie formellement de la façon suivante.

$M \rightarrow_{id} M' \Leftrightarrow [M \neq M' \wedge id(M) \leq_{id} id(M')]$

Ordre identifiable: afin de présenter nos résultats formels, nous introduisons une notion qui n'est pas classique dans les propriétés d'ordre définies dans la littérature. Les auteurs de système à diffusion ou plus largement les publications concernant les propriétés d'ordre n'utilisent généralement pas cette notion. Elle a été définie initialement dans [Toinard 93]. Avec un protocole qui garantit l'ordre identifiable, les messages sont livrés par les entités de protocoles selon leurs identifiants.

$M \rightarrow_{id} M' \Rightarrow M \rightarrow_{do} M'$

Plusieurs remarques ou résultats s'imposent:

- un ordre identifiable est un ordre total, $(M \rightarrow_{id} M' \Rightarrow M \rightarrow_{do} M') \Rightarrow (\forall M, M', M \neq M', M \rightarrow_{do} M' \vee M' \rightarrow_{do} M)$.
- à l'inverse, un ordre total n'entraîne pas un ordre identifiable.
- les identifiants n'ont pas besoins d'être calculés, connus ou maintenus par le protocole. Si l'on sait montrer que le système produit un ordre identifiable c'est suffisant. Cependant, le système peut maintenir directement des identifiants qui respectent notre définition, le système fournit alors l'ordre identifiable par définition.

-on peut remarquer que les identifiants n'ont pas besoin d'être uniques. C'est-à-dire que deux messages différents peuvent avoir le même identifiant (c'est le cas des messages pour lesquels $A(G(M)) \cap A(G(M')) = \varnothing$).

Relation de numérotation: c'est un cas particulier de la relation d'identifiant. Dans ce cas, l'espace $\mathcal{I}d$ est tout simplement l'ensemble des entiers. Chaque message *M* se voit alors attribuer un entier *n(M)*.

$M \rightarrow_n M' \Leftrightarrow [M \neq M' \wedge n(M) \leq n(M')]$.

Ordre numérotable: cet ordre est un cas particulier de l'ordre identifiable. Il est défini initialement dans [Toinard 93]. Il est défini de la façon suivante.

$M \rightarrow_n M' \Rightarrow M \rightarrow_{do} M'$

Ordre identifiable satisfaisant une autre relation: un ordre identifiable respecte une autre relation $\rightarrow_R$ si $M \rightarrow_R M' \Rightarrow M \rightarrow_{id} M'$. Cette définition est utilisée ensuite lorsque l'on considère un ordre identifiable qui satisfait $M \rightarrow_l M'$ ou $M \rightarrow_{dc} M'$. Par exemple, lorsque *M* précède localement *M'* ($M \rightarrow_l M'$) alors l'identifiant de *M* est inférieur ou égal à celui de *M'* ($M \rightarrow_{id} M'$).

2.3. Résultats formels

Les résultats suivants sont valables pour le cas général de diffusion vers des groupes (multicast). Les résultats reposent sur un théorème principal et des théorèmes pour des cas particuliers. Ici, ne donnons pas les preuves. Le lecteur peut les consulter dans [Toinard 93] ou [Toinard 99].

a) Résultat principal

Théorème 1: *un ordre identifiable qui satisfait à la fois les relations locales et causales directes fournit un ordre causal et total*.

$[(M \rightarrow_{id} M' \Rightarrow M \rightarrow_{do} M') \wedge (M \rightarrow_l M' \vee M \rightarrow_{dc} M' \Rightarrow M \rightarrow_{id} M')]$
$\Rightarrow (M \rightarrow_c M' \Rightarrow M \rightarrow_{do} M') \wedge (\forall M, M', M \neq M', M \rightarrow_{do} M' \vee M' \rightarrow_{do} M)$

En fait, le point important est la propriété de transitivité de $\rightarrow_{id}$ qui assure la fermeture transitive de $\rightarrow_l$ et $\rightarrow_{dc}$. En pratique, un protocole qui respecte ce résultat n'a pas à transporter la fermeture transitive pour préserver les relations causales $\rightarrow_c$. En pratique, dans [Florin 92] on trouve un protocole centralisé qui utilise ce résultat. Les identifiants sont tout simplement des numéros attribués selon l'ordre d'arrivé sur le serveur des messages à diffuser. C'est le cas d'un protocole qui maintient directement des identifiants. Dans ce cas, on montre que la complexité en quantité d'information transportée dans les messages est en $O(n)$ au lieu de $O(n^2)$ avec les solutions classiques.

b) Résultat pour l'envoi à soi-même:

Nous considérons le cas où une entité d'application qui émet un message vers un groupe *G(M)* appartient obligatoirement à *G(M)*, c'est-à-dire que l'entité s'envoie le message à elle-même.

$\forall x, E^x(M) \Rightarrow x \in G(M)$

Théorème 2: *un ordre identifiable et local avec un envoi à soi-même fournit un ordre causal et total.*

$[(M \rightarrow_{id} M' \Rightarrow M \rightarrow_{do} M') \wedge (M \rightarrow_{l} M' \Rightarrow M \rightarrow_{do} M') \wedge (E^x(M) \Rightarrow x \in G(M)]$
$\Rightarrow (M \rightarrow_{c} M' \Rightarrow M \rightarrow_{do} M') \wedge (\forall M, M', M \neq M', M \rightarrow_{do} M' \vee M' \rightarrow_{do} M)$

Nous avons montré dans [Toinard 93] qu'un système de diffusion arborescent fournit un ordre identifiable. De plus, s'il garantit un ordre local et fait de l'envoi à soi-même alors il garantit un ordre causal et total. S'il ne respecte pas une de ces conditions alors il n'est pas causal.

Nous allons donner un contre exemple qui montre qu'en l'absence d'envoi à soi-même, une diffusion qui parcourt un arbre ne peut être causal. Sur la figure 3, on voit un message *M* émis par un processus du site *S6* à destination de *G(M)={S7,S4}*. Le processus du site *S7* reçoit *M* (la transmission de *M* est représentée par une flèche pleine) avant d'émettre *M'* (la transmission de *M'* est représentée par une flèche en pointillée), donc $M \rightarrow_{dc} M'$. Or rien n'empêche le message *M'* d'arriver avant *M* sur le site *S2* qui diffuse alors *M'* suivi de *M*. *S2* attribue donc un numéro de séquence à *M'* plus petit que le numéro de séquence de *M*. Le site S4 réordonne les deux messages selon le numéro attribué par *S2*. Le processus du site *S4* reçoit alors *M'* avant *M*, $M' \rightarrow_{do} M$.

Si ce sont les entités de protocole qui font de l'adressage à soi-même (plutôt que les entités d'application qui font de l'envoi à soi-même), le problème est identique. Nous ne détaillerons pas cet aspect.

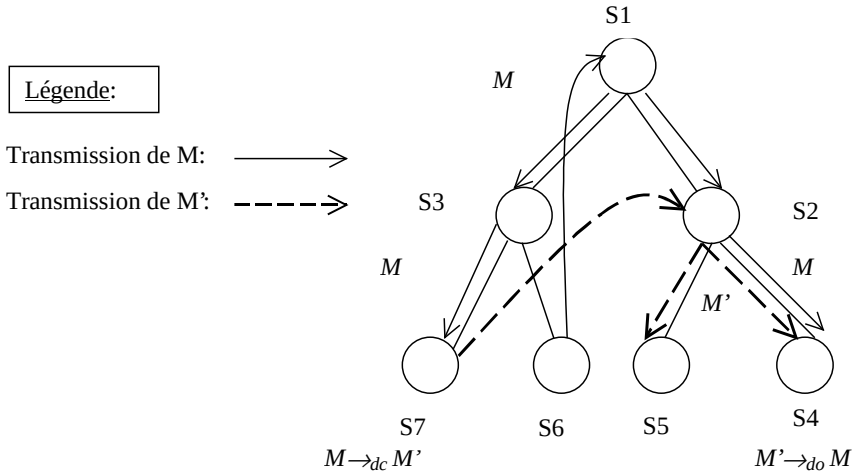

Figure 3. Ordre identifiable et local sans envoi à soi-même n'est pas causal

D'autres résultats ont été établis.

Notamment, nous prouvons [Florin 92] le caractère causal de la plupart des protocoles en diffusion à tout le monde (broadcast) qui fournissent un ordre total dans la mesure où ces protocoles garantissent aussi généralement l'ordre local c'est à dire la livraison en séquence. Ainsi, nous prouvons le caractère causal de [Chang 84] et [Kaashoek 89]. Dans [Toinard 96][Toinard 97] nous utilisons ce résultat pour montrer que le réseau de terrain FIP (Factory Instrumentation Protocol) garantit un ordre causal et total à des vitesses faibles de transmission. Pour les vitesses élevées nous utilisons le résultat pour proposer une extension construite au-dessus du coupleur FIP.

Notons que [Kindberg 95] utilise le principe définit dans [Florin 92] pour éviter la construction d'une couche causale.

Par ailleurs, la présentation des résultats est ici simplifiée et nous invitons le lecteur intéressé à consulter [Toinard 93] ou [Toinard 99]. On y trouve des résultats tenant compte du mode d'adressage. Notamment, l'envoi à soi-même est un cas particulier de l'adressage à soi-même que peut faire une entité de protocole.

3. CTOP

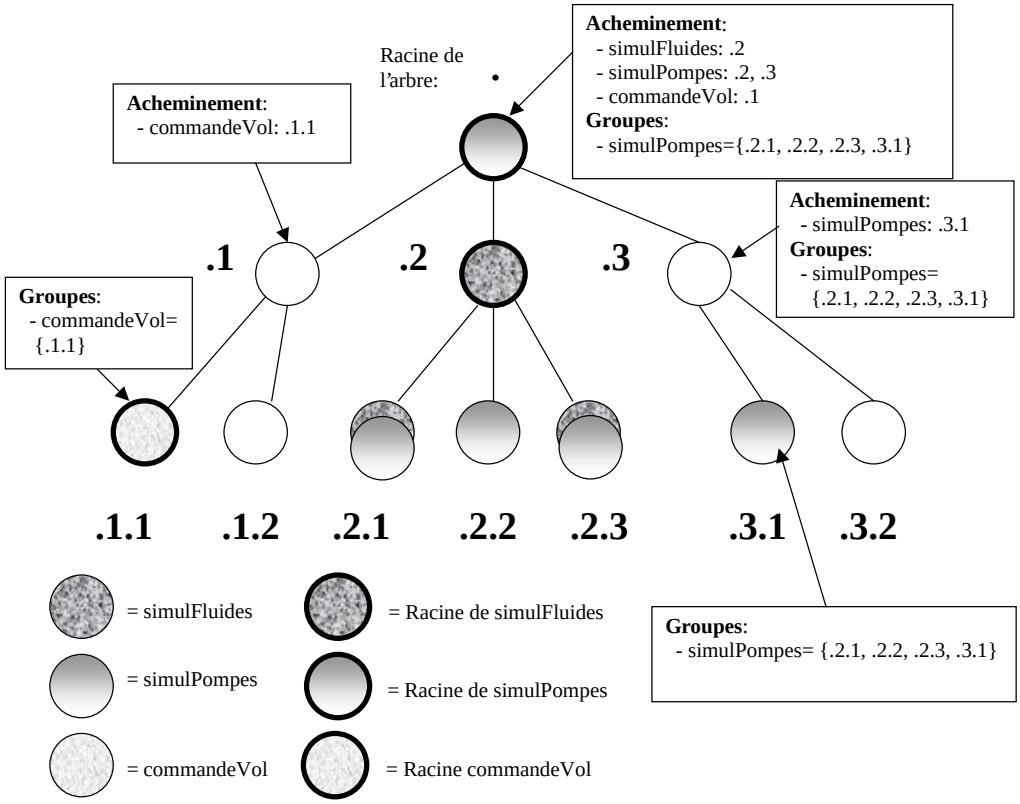

Figure 4. Exemple d'arborescence et de composition des groupes

CTOP repose sur l'utilisation d'une arborescence d'entités d'application. Sur la figure 4, chaque entité d'application de l'arbre porte un nom pointé qui définit sa position dans l'arbre. Dans cet exemple, il existe trois groupes *simulFluides = {.2.1, .2.3}*, *simulPompes = {.2.1, .2.2, .2.3, .3.1}* et *commandeVol={.1.1}*. Chaque groupe possède une racine qui couvre de façon minimale le groupe. Une entité peut appartenir à plusieurs groupes, c'est le cas de *.2.1* qui appartient aux groupes *simulFluides* et *simulPompes*. Une racine de diffusion n'appartient pas forcément au groupe, c'est le cas de la racine *.2* pour le groupe *simulFluides*. Seuls les membres de l'arbre couvrant connaissent la composition du groupe. Les entités, qui vont de la racine de l'arbre à la racine de l'adresse, ont seulement une table d'acheminement qui contient les fils vers lesquels il faut retransmettre les messages. C'est le cas de l'entité *.1* qui n'a qu'une table d'acheminement pour le groupe *commandeVol*. Les autres entités n'ont pas d'information pour le groupe. On voit que chaque entité maintient la quantité d'information minimale qui permet à la fois de maintenir l'appartenance au groupe et d'assurer le routage. Les nœuds qui ne sont pas concernés par un groupe n'ont aucune information et ne reçoivent aucun message concernant ce groupe.

Nous allons voir par la suite les protocoles et les algorithmes qui permettent de maintenir les états (groupe et acheminement) de façon cohérente. Ces protocoles sollicitent très peu la racine de l'arbre grâce à la connaissance répartie de la composition du groupe et à la notation pointée. Lors des entrées et des sorties de groupe, ces informations permettent à la racine courante de calculer localement la position d'une nouvelle racine et de la déplacer en conséquence. Hormis, lors de la création d'un groupe, la racine de l'arbre n'est pas sollicitée pour effectuer ces déplacements de racine de groupe.

L'architecture de CTOP est représentée sur la figure 5.

Au niveau le plus bas, un composant assure la transmission fiable notamment entre un père et ses fils mais aussi entre un demandeur et la racine du groupe ou de l'arbre. Ce composant peut être instancié sous deux formes. La première consiste à utiliser des connexions TCP entre une paire d'entités, ainsi chaque liaison père-fils est réalisée par une connexion TCP. Une autre instance de ce composant est proposée au moyen d'UDP. Dans ce cas, CTOP fiabilise les transmissions en utilisant un mécanisme d'acquittement négatif qui évite d'avoir systématiquement à retransmettre des acquittements. Le débit en sortie peut alors être supérieur à celui de TCP qui souffre des problèmes connus de *slow-start* et est mal adapté au cas où un même message est destiné à plusieurs entités (ce qui est le cas ici). L'application peut choisir d'instancier CTOP en mode TCP ou UDP. Nous ne détaillerons pas plus avant l'aspect transmission fiable.

Ensuite, les événements transmis de façon fiable sont traités par le composant de routage. Celui-ci est le cœur de CTOP. Il calcule la position de la racine, la déplace, et retransmet les événements reçus aux fils qui acheminent le groupe.

Le composant d'appartenance aux groupes est appelé par le routage pour mettre à jour l'appartenance au groupe sur un événement d'entrée ou de sortie. C'est un composant simple car le routage effectue la plus grande partie du travail et n'achemine que les événements nécessaires. Donc les événements reçus par le composant d'appartenance sont forcément des informations utiles et qui permettent de mettre à jour de façon cohérente la base de donnée répartie d'appartenance aux groupes.

On constate qu'il n'y a aucun composant d'ordonnancement dans CTOP. En effet, l'ordonnancement est réalisé par preuve. La livraison en séquence réalisée par la transmission fiable et le parcours de l'arbre garantissent les propriétés d'ordonnancement. L'ordre total est garanti par le fait qu'une entité retransmet les événements reçus sur l'arbre. C'est le parcours de l'arbre qui garantit une diffusion identifiable et donc un ordre total. Il suffit de faire un envoi à soi-même pour garantir l'ordre causal au moyen de notre résultat formel. Une remarque importante est que les identifiants ne sont pas connus ni maintenus par le système. Ils ne correspondent pas directement aux numéros de séquence pour la transmission fiable. L'identifiant d'un message est obtenu en assemblant d'un point de vue formel les numéros de séquences locaux des messages qui sont arrivés sur la racine de diffusion avant la diffusion de *M*. Ces identifiants ne sont pas maintenus par le système ni transmis dans les messages. Ils servent juste à montrer que la livraison se fait selon ces identifiants.

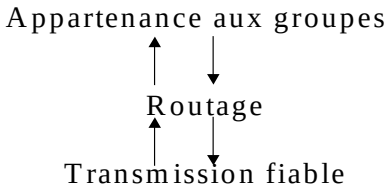

Figure 5. Architecture de CTOP

3.1. Service d'installation dans l'arbre

Ce service est simple, il consiste en une méthode `void installer(char *ip, short port)`qui envoie une requête d'installation à l'entité d'application (`ip,port`). Cette méthode récupère en réponse le nom pointé alloué par le père (`ip,port`) et installe le fils demandeur en affectant son nom pointé avec l'information retournée par le père. Ce service peut être utilisé même en cours de fonctionnement de l'arborescence. C'est à dire qu'il est possible à tout moment d'installer une nouvelle entité d'application dans l'arbre.

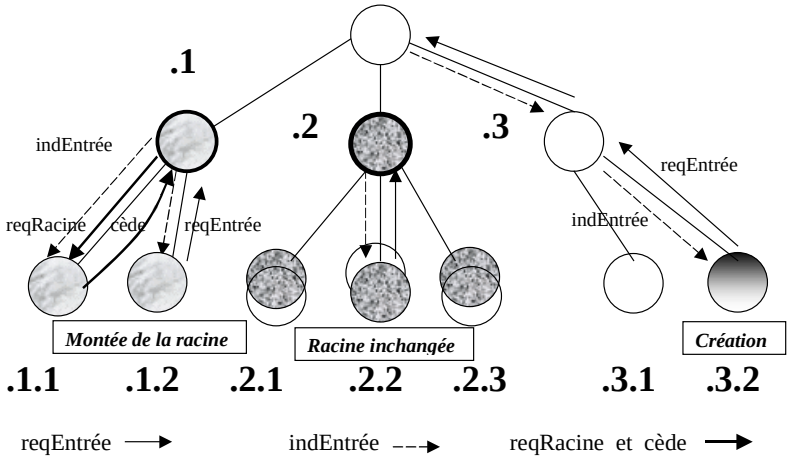

Figure 6. Protocole d'entrée

3.2. Service d'entrée dans un groupe

Ce service correspond à une méthode `void entrer(char *nomGroupe)`qui est utilisée par une entité d'application pour entrer dans un groupe. Le service envoie une requête d'entrée (*reqEntrée*) au père.

Le protocole distingue trois cas principaux illustrés par la figure 6.

Le premier cas est la création d'un groupe. Il correspond à l'entrée d'un premier membre (ex: *.3.2*) dans le groupe. La requête d'entrée (*reqEntrée*) remonte l'arbre de proche en proche jusqu'à atteindre la racine de l'arbre. Celle-ci s'aperçoit qu'elle ne connaît pas le groupe. Elle en déduit que c'est une création et que le demandeur devient la racine du nouveau groupe. Elle envoie une indication d'entrée (*indEntrée*) au demandeur lui indiquant qu'il est la racine. Cette indication parcourt l'arbre en direction du demandeur (ex: *.3.2*). Chaque processus intermédiaire (ex: *.3*) met à jour sa liste d'acheminement en ajoutant le fils concerné. Il retransmet alors le message vers ce fils. Lorsque le demandeur reçoit l'indication, il mémorise qu'il est la racine et le parcourt de l'arbre s'arrête.

Le second cas est celui de la montée de la racine. La requête d'entrée (*reqEntrée*) remonte jusqu'à une entité qui achemine déjà l'adresse (ex: *.1*) avec un seul fils dans sa liste d'acheminement. L'entité (ex: *.1*) vérifie, au moyen de la composition du groupe et de la liste d'appartenance, que la racine courante est en dessous d'elle (ex: *.1.1*). Elle envoie alors une requête (*reqRacine*) de demande de transfert de la racine du groupe. Cette requête descend l'arbre. Lorsque la racine courante reçoit la requête de transfert, elle envoie une réponse (*cède*) de façon directe au demandeur (sans passer par l'arbre) et cède la racine. La réponse contient la composition du groupe. Lorsque le demandeur (ex: *.1*) reçoit la réponse, il peut traiter la requête d'entrée qui est à l'origine de la montée et envoyer une indication d'entrée (*indEntrée*) qui parcourt l'arbre selon les différentes listes d'acheminement.

Dans le troisième cas, la racine est inchangée. La requête (*reqEntrée*) remonte jusqu'à la racine courante (ex: *.2*). Dans l'exemple, le père du demandeur est la racine du groupe. Lorsqu'il y a des noeuds intermédiaires, ils font remonter la requête d'entrée (qui arrive sur une branche acheminée) pour atteindre la racine du groupe. Lorsqu'elle reçoit la requête, la racine courante envoie en réponse une indication d'entrée (*indEntrée*) qui descend l'arbre.

L'algorithme permet que les messages ne parcourent que les chemins nécessaires. L'entrée dans un groupe est donc effectuée au plus vite. Un point important est que l'indication d'entrée permet de mettre à jour les listes d'acheminement. Ainsi, un nœud ne retransmettra les futurs messages à destination de ce groupe que vers des fils présents dans la liste d'acheminement du groupe. Enfin, le nouvel arrivant connaît la composition du groupe lorsqu'il reçoit l'indication d'entrée.

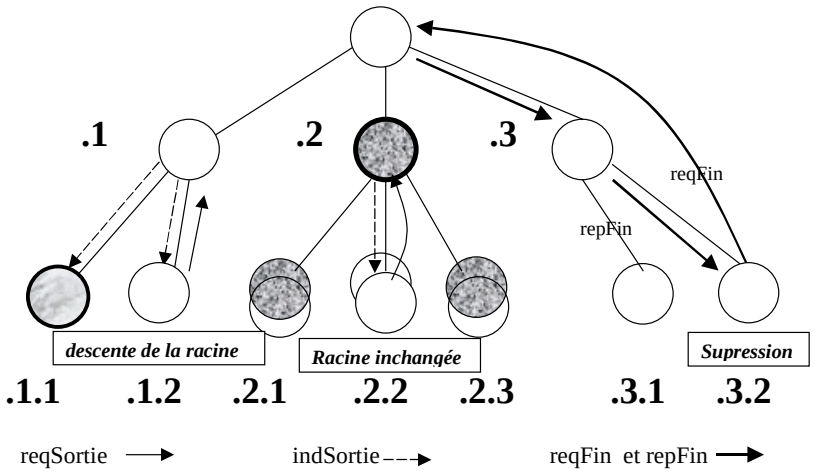

Figure 7. Protocole de sortie

3.3. Service de sortie d'un groupe

Ce service correspond à une méthode `void sortir(char *nomGroupe)` qui est utilisée par une entité d'application pour sortir d'un groupe auquel elle appartient. Le service envoie une requête de sortie (*reqSortie*) directement à la racine du groupe.

Le protocole distingue trois cas illustrés figure 7.

Dans le premier cas, la racine est inchangée. La racine du groupe (ex: *.2*) reçoit la requête (*reqSortie*). Elle recalcule la position de la racine grâce à la nouvelle composition du groupe (la composition courante moins le sortant). Elle s'aperçoit qu'elle reste racine. Elle envoie alors une indication de sortie (*indSortie*) en signalant via un champ du message (*indSortie.descendRacine=faux*) qu'il n'y a pas de changement. L'indication de sortie descend l'arbre selon les listes d'acheminement.

Dans le second cas la racine descend. La racine courante (ex: *.1*) s'aperçoit en recalculant la position de la racine qu'elle n'est plus racine. Elle envoie une indication de sortie (*indSortie*) en signalant que la racine descend vers une nouvelle racine dont elle fournit le nom pointé. Les nœuds entre l'ancienne et la nouvelle racine en profitent pour supprimer les structures de données d'appartenance pour ce groupe.

Le troisième cas correspond à la suppression du groupe. Lorsque le dernier membre (ex: .3.2) sort, il envoie une requête de fin de groupe à la racine de l'arbre. La racine de l'arbre transmet alors une réponse de fin qui descend la dernière branche possédant des listes d'acheminement. Au fur et à mesure de la descente de la réponse, chaque nœud supprime la liste d'acheminement de ce groupe.

3.4. Service de diffusion d’un message vers un groupe

Une entité d'application doit préalablement demander à entrer dans un groupe pour pouvoir diffuser un message vers ce groupe. Cette propriété est nécessaire sinon l'envoi à soi-même n'est pas assuré est la diffusion ne peut être causale.

L'application utilise la méthode `void emettre(char* nomGroupe, char* messAppli)`pour émettre un message applicatif vers un groupe. Elle utilise la méthode `void recevoir(char* nomGroupe, listeMessages *premier)`qui retourne le premier message disponible ainsi que le nom du groupe. Cette primitive livre les messages selon l'ordre causal et total. Pour émettre un message, le service envoie une requête à la racine de diffusion, celle-ci reçoit le message et le diffuse vers tous les fils de sa liste d'acheminement. Un nœud qui reçoit le message, le retransmets en utilisant sa liste d'acheminement. Ainsi, le message atteint tous les membres du groupe en parcourant de façon minimale l'arbre.

La primitive `recevoir` se contente de retirer un message de la file de livraison. A l'inverse d'autres protocoles, l'entité réceptrice n'effectue pas de réordonnancement selon des informations de causalité ou selon un séquencement global. Elle se contente de faire une livraison en séquence de tous les messages qui lui proviennent de son père. Avec des connexions TCP, les entités de protocoles ne gèrent pas de numéros de séquence.

3.5. Algorithmes d'entrée et de sortie

Voici de façon simplifiée les algorithmes pour le traitement des principaux messages:

```
réception_reqEntree(entrant,groupe,emetteur){
  si(nbFilsQuiacheminent(groupe)){
  // J’ai un seul fils qui achemine ce groupe
  // donc je peux éventuellement devenir la racine du groupe.
    si(filsQuiAchemineDifferent(emetteur) et racineCouranteEstUnDescendant(groupe)){
      // La requête est arrivée sur une nouvelle branche, je demande à devenir la racine
      envoiFilsQuiAcheminent(reqRacine);
      retour;}}
  si( suisLaRacineDuGroupe(groupe) ou (!achemine(groupe) et suisRacineArbre()) ){
  // Je suis la racine du groupe ou je n'achemine pas le groupe mais suis la racine
  // de l'arbre
    si (!achemine(groupe) et suisRacineArbre()){ // C'est une création de groupe
      // Affecter l'entrant comme racine du groupe dans l'indication d'entrée
       indEntree.racineGroupe=entrant;
    sinon { // C'est moi la racine du groupe
       indEntree.racineGroupe=moi;
      // Demander au composant d'appartenance d'ajouter l'entrant au groupe
      ajouteMembreAuGroupe(entrant,groupe);
      }
    // Ajouter l'émetteur dans ma liste d'acheminement du groupe
    ajouteFilsQuiAchemine(emetteur,groupe);
    //envoyer à tous  les fils qui acheminent le groupe l'indication d'entrée
    envoiFilsQuiAcheminent(indEntree); return; }
  retransmetAuPere(reqEntree); }

réception_reqRacine(demandeur,entrant,groupe){
  si (suisLaRacineDuGroupe(groupe)){
     si ( (niCessionNiFinEnCours(groupe)) ){
        envoiDirect(demandeur,cedeRacine);
        retour;}
  sinon { // Il faut que je retransmette la demande à mon fils
    envoiFilsQuiAcheminent(reqRacine); return;}
  repondreNegativement(demandeur);}
```

```
réception_cedeRacine(demandeur,entrant,groupe){
  // Demander au composant d'appartenance de mettre à jour le groupe
  // avec moi comme racine
  miseAJourGroupe(moi,groupe);

  // Je peux maintenant traiter la demande d'entrée qui m'a amené à être racine

  // J'ajoute le fils qui couvre l'entrant dans ma liste d'acheminement
  ajouteFilsQuiAchemine(filsAuDessus(entrant),groupe);
  // Demander au composant d'appartenance d'ajouter l'entrant au groupe
  ajouteMembreAuGroupe(entrant);
  envoiFilsQuiAcheminent(indEntree);}

réception_indEntree(entrant,racineGroupe,groupe){
  // Si je suis pas entre la racine de l'arbre et la racine du groupe,
  // demander au composant d'appartenance de mettre à jour le groupe
  miseAJourGroupe(racineGroupe,groupe);
  si (entrant==moi) {
    si (racineGroupe==moi){
      // Demander au composant d'appartenance de créer le groupe
      créerGroupe(groupe);}
    // Demander au composant d'appartenance de m'ajouter au groupe
    ajouteMembreAuGroupe(moi);}
  // Si besoin, ajoute un fils à la liste d'acheminement
  ajouteFilsQuiAchemine(filsAuDessus(entrant),groupe);
  // Je retransmets l'indication vers tous les fils qui acheminent
  envoiFilsQuiAcheminent(indEntree,groupe);}

réception_sortie(sortant,groupe) {
  si ( !suisLaRacineDuGroupe(groupe) ){
     repondreNegativement(sortant)};
  // Demander au composant d'appartenance de retirer le sortant du groupe
  retireMembreDuGroupe(sortant,groupe);
  nouvelleRacine=calculeNouvelleRacine(groupe);
  si (nouvelleRacine!=moi) indSortie.descendRacine=vrai; // C'est plus moi la racine
  sinon indSortie.descendRacine=faux; // Je reste la racine
  envoiFilsQuiAcheminent(indSortie,groupe);
  si ( sortantSeulSurSaBranche(sortant,groupe)){
      // Il faut retirer un fils de ma liste d'acheminement
      retireFilsQuiAchemine(filsAuDessus(sortant),groupe);}
  si (nouvelleRacine!=moi){
    // Ce n'est plus moi la racine
    // Je peux demander au composant d'appartenance de supprimer ce groupe
    // Remarque: le groupe continue d'être routé
    détruireGroupe(groupe);}
}

réception_indSortie(sortant,racineGroupe,groupe) {
  // Commencer par mettre à jour l'appartenance au groupe
  miseAJourGroupe(racineGroupe,groupe);
  si (indSortie.descendRacine==vrai et racineGroupe==moi)
     indSortie.descendRacine=faux;
  // Je retransmets l'indication à tous les fils qui acheminent ce groupe
  envoiFilsQuiAcheminent(indSortie);
  si (estLeDernierSurSaBranche(sortant,groupe)) { // C'est le dernier fils sur cette branche
      // Donc il faut retirer un fils de ma liste d'acheminement
      retireFilsQuiAchemine(filsAuDessus(sortant),groupe);
  si (indSortie.descendRacine=vrai)
     // Demander au composant d'appartenance de supprimer ce groupe
     détruireGroupe(groupe);
}
```

3.6. Traitement des requêtes concurrentes

Les requêtes concurrentes d'entrée ou de sortie sont traitées.

Si deux entrées modifient la racine, une atteint un nœud qui peut décider et obtient le transfert. La seconde sera traitée après le transfert. Afin d'éviter d'avoir à gérer des files de requêtes en attente, dès qu'un nœud en cours de transfert reçoit une requête de transfert il répond négativement afin que le second entrant rejoue sa demande ultérieurement. Les conflits sont donc résolus par rejeu en cas d'avis négatif. Si deux sorties modifient la racine, la seconde requête va atteindre une racine qui n'est plus la bonne. Celle-ci répond négativement et la seconde requête pourra être rejouée sur cet avis négatif.

Si une entrée et une sortie modifient toutes les deux la racine, l'entrée fait monter la racine et la sortie la fait descendre:

a) soit, la requête de sortie atteint la racine du groupe avant la demande de transfert et dans ce cas la requête de transfert va poursuivre sa route jusqu'à atteindre la nouvelle racine. Si cette nouvelle racine sort à son tour et tente de supprimer le groupe avant que la demande de transfert ne l'atteigne alors elle répondra négativement et l'entrant pourra rejouer sa demande. Si le message de transfert passe avant la réponse de fin alors la racine courante répond négativement (sinon il n'y a pas de requête de transfert qui descend puisque l'acheminement a été détruit donc la requête d'entrée monte jusqu'à la racine de l'arbre pour recréer le groupe).

b) soit, la requête de transfert atteint la racine de groupe avant la sortie et dans ce cas l'ancienne racine répond négativement à la sortie. Lorsqu'il reçoit la réponse négative le sortant attend simplement d'avoir l'indication d'entrée avec la nouvelle racine pour pouvoir rejouer sa requête.

Enfin, le système reste cohérent car il n'existe en permanence qu'une seule racine pour le groupe. En effet, la racine ne peut faire qu'un déplacement à la fois qui est une montée ou une descente.

4. SCHÉMA DE PREUVE DE L'ORDRE CAUSAL ET TOTAL

Nous donnons ici seulement le schéma de preuve qui utilise le théorème 2. La preuve est dans [Toinard 93].

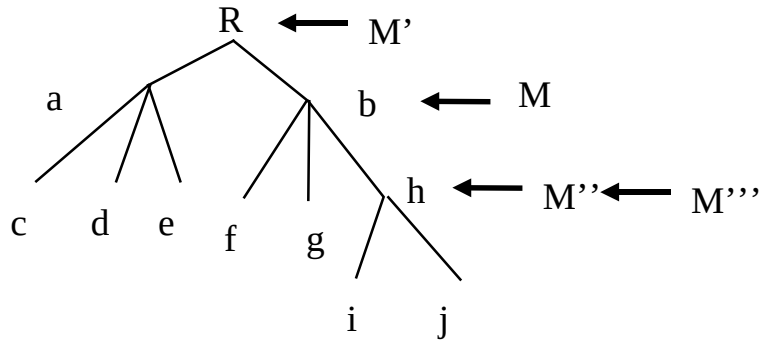

Figure 8. Scénario de transmission

L'idée de base repose sur la définition d'identifiants. Nous présentons la définition des identifiants sur une arborescence en fonction de numéros locaux. Ces numéros locaux n'ont pas à être gérés par le protocole. Nous notons *L(M)* le numéro local attribué sur la racine de diffusion *R(M)* au message *M* qu'elle diffuse. Le message *M* peut être soit une indication d'entrée, soit une indication de sortie soit un message applicatif. Chaque message *M* a donc un seul numéro local défini sur la racine de diffusion qui l'émet. Les nœuds de l'arbre sont nommés par une lettre *(R,a,b,c,d,...)* plutôt que par des noms pointés. La profondeur de l'arbre étant quatre, un identifiant *id(M)* correspond à un ensemble $C_M[D]$ de quatre éléments (*D=4*). Pour chaque élément *n* de la racine de l'arbre à la racine de l'adresse, $C_M[n]$ est le numéro *L(m)* du dernier message *m*, initié par la racine de niveau *n*, qui a été reçu

par *R(M)* avant d'émettre *M*. L'élément $C_M[R(M)]$ est *L(M)*. Pour chaque élément $C_M[n]$ sous la racine *R(M)*, $C_M[n]=0$. Sur la figure 8, le premier message *M* est diffusé par la racine *b* vers le groupe *{f,i,j}*. L'identifiant associé à *M* est *id(M)=[0,1,0,0]* (premier message diffusé par *b* de niveau *2*). Ensuite le message *M'* est diffusé par *R* vers le groupe de serveurs *{R,c,j}*. L'identifiant associé à *M'* est *id(M')=[1,0,0,0]* (premier message diffusé par *R*). Le message *M''* est diffusé par *h* vers le groupe *{i,j}*. *h* reçoit *M* avant d'émettre *M''*, donc l'identifiant de *M''* est *Id(M'')=[0,1,2,0]* (deuxième message diffusé par *h* après avoir reçu le premier message de *b*). Ensuite, le message *M'''* est diffusé par *h* vers *{i,j}* après traitement de *M, M''* et *M'*. Son identifiant est *id(M''')=[1,1,4,0]*.

On peut trouver d'autres façons de construire les identifiants. Mais pour cette méthode, la preuve qu'il y a un ordre identifiable, est dans [Toinard 93]. Pour garantir l'ordre local, la méthode la plus simple est que chaque émetteur attende d'avoir en retour son message diffusé (c'est-à-dire l'indication de diffusion) avant de pouvoir émettre le message suivant, donc les messages sont forcément livrés selon l'ordre local. En fait dans CTOP la transmission fiable entre l'émetteur et la racine du groupe garantit que la racine va transmettre les messages selon l'ordre local de l'émetteur. Donc, il n'est pas nécessaire d'attendre pour émettre un nouveau message. Enfin, comme une entité doit appartenir au groupe vers lequel elle émet, il y a bien envoi à soi-même. Ainsi, on a un ordre identifiable et local avec envoi à soi-même et le théorème 2 s'applique.

Les numéros locaux n'ont pas besoin d'être maintenus. En effet, la transmission fiable sur tout l'arbre garantit que deux messages *M* et *M'* sont bien reçus dans l'ordre de diffusion qui a lieu à partir de la racine qui est la plus basse pour les deux messages *M* et *M'*.

5. CONCLUSION

Nous avons présenté CTOP qui est une architecture de diffusion vers des groupes. CTOP maintient l'appartenance aux groupes et livre tous les événements (entrée, sortie, message applicatif) selon un ordre causal et total. La solution est originale car elle évite complètement la construction d'une couche causale et garantit la propriété d'ordre au moyen d'un résultat formel. Elle se fait en diffusant un seul message qui parcourt un arbre couvrant. L'application peut obtenir seulement un ordre total si elle ne demande pas à appartenir au groupe vers lequel elle émet. Le système ne transporte aucune information de séquencement dans les messages notamment lorsqu'elle fonctionne en utilisant TCP. Lorsqu'elle est instanciée en mode UDP, une simple livraison en séquence suffit. Le papier décrit les principes de la méthode formelle et montre son utilisation dans le cadre de CTOP. Il décrit l'architecture en trois couches de CTOP et présente le mécanisme de routage des messages sur l'arbre. Les services fonctionnent à l'heure actuelle sur des machines Unix. Un portage est en cours sur Windows. Cette architecture est intégrée à un environnement de réalité virtuelle distribuée [Costantini 2000] afin de permettre des simulations réparties sur l'Internet dans le cadre d'applications de conception assistée par ordinateur. Sur les bases de la méthode BB [Kaashoek 93], une optimisation permet de diffuser directement un message en multicast et une référence du message est diffusée sur l'arbre ce qui évite alors qu'un gros volume de données emprunte l'arbre.

6. BIBLIOGRAPHIE

[Amir 92] Y. Amir, D. Dolev, S. Kramer, D. Malki, "Transis: A communication subsystem for high availability", In Annual Symposium on Fault Tolerant Computing, number 22, pages 76-78, July 1992.

[Baldoni 96] Baldoni R., Friedman R., Van Renesse R., *The Hierarchical Daisy Architecture for Causal Delivery*, Cornell University Technical Report TR96-1610, September 1996.

[Birman 91] K.P. Birman, A. Schiper, P. Stephenson, "Lightweight causal and atomic group multicast", ACM TOCS, Vol 9 N°3, pp272-314, Aug. 1991.

[Costantini 2000] F. Costantini, A. Sgambato, C. Toinard, N. Chevassus, F. Gaillard, "An Internet Based Architecture Satisfying the Distributed Building Site Metaphor" IRMA2000 Multimedia Computing Track, Anchorage, Alaska, 21-24 May, conference proceedings, pp.151-155, published by IDEA Group Publishing ISBN 1-878289-84-5. 2000.

[Florin 92] G. Florin, C. Toinard, "A new way to design causally and totally ordered multicast protocols". ACM Operating Systems Review, Vol 26, Nunber4, October 1992.

[Garcia 91] H. Garcia-Molina, A. Spauster, "Ordered and Reliable Multicast Communication", ACM TOCS, Vol 9 N°3, pp272-314, Aug. 1991.

[Huleilhel 96] Huleilhel N., *Efficient Ordering of Messages in Wide Area Networks*, Master Thesis, http://www.cs.huji.ac.il/labs/transis/publications.html, March 1996.

[Kaashoek 93] Kaashoek, M. F., A. S. Tanenbaum, K. Verstoep. "Group Communication in Amoeba and its Applications", *Distributed Systems Engineering*, vol. 1, no. 1, September 1993.

[Kindberg 95] T. Kindberg, "A Sequencing Service for Group Communication", Proc. 14th annual ACM Symposium on Principles of Distributed Computing, p.260, Aug. 1995.

[Malloth 96] C.P. Malloth "Conception and implementation of a toolkit for building fault-tolerant distributed applications in large scale networks", Thèse de Doctorat. Ecole Polytechnique Fédérale de Lausanne. 1996.

[Mishra 93] S. Mishra, L.L. Peterson, R.D. Schlichting, "Consul: a communication substrat for fault-tolerant distributed programs". Distrib. Syst. Engng 87-103, 1993.

[Moser 96] L.E. Moser, P.M. Melliar-Smith, D.A. Agarwal, R. Budhia, C. Lingley-Papadopoulos, "Totem: A Fault Tolerant Multicast Group Communication System". Communications of the ACM, 39(4):54-63, April 1996.

[Peterson 89] L.L. Peterson, N.C. Buchholtz, R.D. Schlichting, "Preserving and Using Context Information in IPC". ACM TOCS, Vol 7 N° 3, pp217-246, AUg. 1989.

[Renesse 95] R. Van Renesse, K.P. Birman, R. Friedman, M. Hayden, and D. Karr, "A Framework for Protocol Composition in Horus", ACM Symposium on Principles of Distributed Computing, August 1995.

[Toinard 93] C. Toinard, "Etudes des diffusions ordonnées sur groupes". Thèse de doctorat de l'Université Paris 6, Février 1993. http://cedric.cnam.fr/~toinard/Recherche/these.ps

[Toinard 96] C. Toinard, N. Chevassus, "De nouvelles architectures de communication pour le contrôle de procédés", Real Time Systems (RTS'96), Paris, Proceedings in Teknea Editors France, 1996.

[Toinard 97] C. Toinard, N. Chevassus, "An object communication architecture for real time distributed process control", ECOOP97, Workshop on OO Technology and Real Time Systems, Conf. Proc., Springer Verlag, Lectures Notes in Computer Sciences, Object Oriented Technology ECOOP'97 Workshop Reader, 10 june 1997.

[Toinard 99] C. Toinard, G. Florin, C. Carrez,"A Formal Method to Prove Ordering Properties of Multicast Systems", ACM Operating Systems Review, Vol. 33, No.4, pp.75-89. 1999.

[Verissimo 89] P. Verissimo, L. Rodrigues, M. Baptista, "AMp: A highly parallel atomic multicast protocol", In Proceedings of SIGCOM'89 Symposium, pages 83-93. ACM, 1989.

Biographie

Christian Toinard est Maître de Conférences à l'ENSERB. Il effectue sa recherche au sein du Laboratoire CEDRIC du CNAM à Paris. Dans le projet européen IST AIT VEPOP, il assure actuellement la responsabilité scientifique pour le CNAM. Depuis 1993, il s'attache à développer une coopération industrielle à travers des contrats de recherche. Il travaille notamment, en coopération avec le Centre Commun de Recherches EADS, à la définition de services de répartition pour la conception coopérative en environnement de réalité virtuelle. Par ailleurs, depuis 1989 il étudie les mécanismes de diffusion, leur mise en œuvre et leur utilisation.