L'interprétation d'un nom est un calcul explicitement réparti. Tout message concernant une opération sur une entité désignée symboliquement a une entête standard. Cette entête comporte un code d'opération, le numéro du catalogue où l'interprétation doit commencer et le chemin d'accès. L'UID du serveur est implicitement spécifié en lui envoyant le message.

Algorithme de traduction

Le serveur qui reçoit le message commence la traduction du chemin d'accès à partir du catalogue spécifié dans l'entête. A ce niveau le code d'opération est ignoré. Si la traduction aboutit, le code d'opération est consideré et l'opération est appliquée à l'entité. Si la traduction n'aboutit pas, un message de réponse d'erreur est envoyé tout de suite au client.

Cependant, si pendant le processus de traduction le serveur rencontre un noeud qui désigne un catalogue d'un autre serveur, i. e., un noeud associé à un nom interne de catalogue géré par un autre serveur, il remplace le champ *Catalogue* de l'entête du message par celui qu'il a rencontré, le champ *Chemin-d'accès* par la partie du chemin d'accès non encore analysée et redirige la requête vers le serveur associé au nom interne du catalogue.

Un serveur peut ainsi participer à la traduction de noms désignant des descripteurs qu'il ne connaît pas, concernant des opérations qu'il ne peut pas exécuter et ayant des syntaxes dans la partie finale du chemin d'accès qu'il ne saurait pas interpréter.

Cette interprétation répartie des noms est réalisée par un ensemble de standards et par un algorithme que tous les serveurs doivent être capables d'exécuter. Parmi ces standards on trouve un ensemble d'opérations pour manipuler des catalogues et une opération qui permet de traduire un nom symbolique dans un nom interne.

Catalogues de préfixes

Chaque processus hérite du père le nom interne d'un catalogue courant. L'usager n'a donc pas à spécifier le catalogue où commence l'interprétation des noms. Par défaut, tous les noms sont interprétés par rapport au catalogue courant. L'usager peut changer ce catalogue par une opération similaire à l'opération "cd" d'Unix (*change dir*).

Pour augmenter la flexibilité, chaque station de travail exécute pour le compte de son usager un "serveur de préfixes". Ce serveur implante un catalogue qui ne sert qu'à associer des identificateurs à des noms internes de catalogues des autres serveurs. C'est un pur serveur de désignation.

Si un chemin d'accès commence par la syntaxe (d'exception):

"[" identificateur "]" chemin d'accès

la requête concernant l'entité ainsi désignée n'est pas adressée au serveur associé au catalogue courant mais au serveur de préfixes. Ce serveur analyse le préfixe "[identificateur]" et redirige la requête vers le serveur associé au préfixe.

Les serveurs de préfixes associent aussi automatiquement des identificateurs prédéfinis aux catalogues racines des serveurs du réseau. C'est cette association automatique (et non à la demande) qui permet aux usagers du système de "voir le monde de la même façon". Ces préfixes prédéfinis constituent donc une sorte de noms globaux. Dans la figure 1.14, les noms "d/c", "[a]c/d/c" et "[c]c", utilisés par le client A, et le nom "[a]c/d/c", utilisé par le client B, désignent tous la même entité.

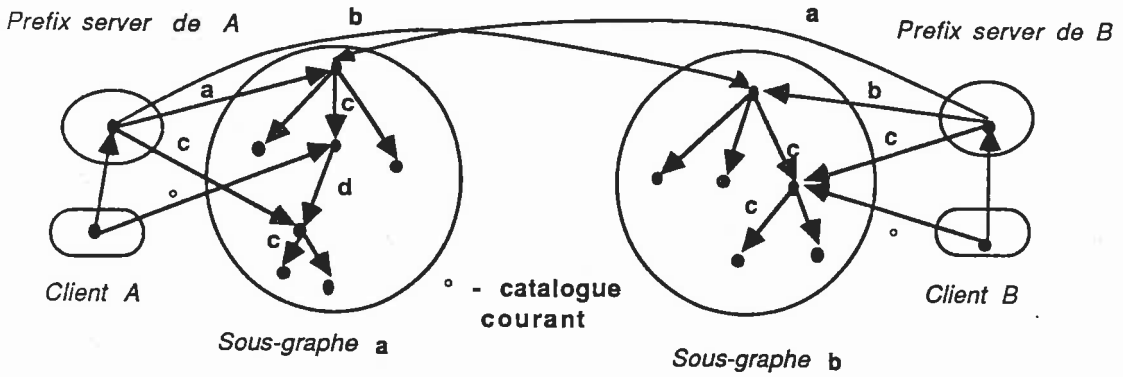


fig. 1.14 - Serveurs de préfixes

7.3. Comparaison

Nous venons de présenter deux démarches d'interprétation des noms dans un SDS d'un système réparti. Elles sont souvent présentées selon la dichotomie: "serveurs de noms vs serveurs de fichiers". Comparons maintenant leurs avantages et leurs défauts selon plusieurs critères.

Transparence de la localisation et du réseau

L'approche serveur de noms logiquement centralisé offre une vision unique du système; elle permet la structuration de l'espace des noms selon les besoins administratifs et non de localisation. Les implantations actuelles prouvent que cette transparence est possible même dans des réseaux de taille importante.

Au contraire, l'approche SDS explicitement réparti suggère une structuration de l'espace des noms calquée sur la répartition des entités par les différents serveurs ou par les différents sites. Bien que l'on puisse dépasser cette limitation à l'aide de mécanismes de pré-traitement de noms (comme ceux introduits par les serveurs de préfixes du V-System ou ceux suggérés dans [Welch 86]), l'administration du système est subordonnée à la répartition des entités par serveurs ou par sites. Ceci peut être sans problème si le réseau est de petite taille et l'administration très intégrée, ou si les différents serveurs dupliquent les ressources (ceci rend indifférent le serveur utilisé).

Modularité et uniformité

L'approche serveur de noms logiquement centralisé isole la fonction de désignation des autres fonctions du système. Cette séparation est en général un gage de modularité. Elle garantit aussi une cohérence de la sémantique des opérations concernant la désignation. D'autres propositions qui suivent aussi cette option, telles que Clearinghouse [Oppen 81] ou ECMA [ECMA 85], partiellement inspirées de Grapevine, poussent plus loin la généralité des SDS conçus selon ce modèle: elles introduisent des méthodes permettant d'étendre le nombre de niveaux de hiérarchie et de paramétrer et typer les descripteurs des entités. Cette approche est donc particulièrement adaptée aux

réseaux de systèmes hétérogènes faiblement intégrés

Au contraire, l'approche SDS explicitement réparti, en éclatant la fonction de désignation sur l'ensemble des serveurs concernés par les autres fonctions du système, introduit la duplication de fonctions et un problème supplémentaire d'uniformité. L'uniformité de la sémantique des opérations est alors supportée par un ensemble de conventions auxquelles tous les serveurs doivent obéir. Ce problème n'est négligeable que si tous les serveurs sont sous la même autorité administrative - ils appartiennent donc à un système fortement intégré.

Généralité d'application

Un service spécialisé dans la désignation, comme celui qui est suggéré par l'approche serveur de noms, est particulièrement adapté à la désignation des entités qui ne sont pas matérialisées par des serveurs: les usagers, les sites, les services, les serveurs eux-mêmes et des groupes de ces entités. Cette aspect est particulièrement intéressant parce qu'il permet une base commune pour l'administration décentralisée mais intrinsèquement cohérente du système; ceci facilite des problèmes de protection et d'authenticité des usagers.

L'absence de cette base commune est une limitation importante de l'approche SDS explicitement réparti. Selon cette approche, ou bien les serveurs se méfient mutuellement les uns des autres, ou bien l'on présuppose qu'il y a une cohérence d'administration du système qui n'est imposée que manuellement. C'est souvent ce qui se passe dans les réseaux de petite taille où tous les serveurs partagent des copies cohérentes des fichiers de définition des usagers.

Réutilisation du logiciel

L'approche SDS explicitement réparti éclate la gestion des noms sur les différents serveurs qui gèrent les ressources: elle est plus adaptée à l'interconnexion de systèmes préexistants dans la mesure où elle est plus flexible et n'impose pas une base commune. La construction de systèmes répartis de gestion de fichiers bâtis sur la notion de montage distant suit pour l'essentiel cette approche.

Au contraire, les systèmes construits selon l'approche serveur de noms logiquement centralisé exigent la reécriture partielle des serveurs, du moins à cause des besoins de duplication et de résistance aux pannes qui leur sont associés et des nouveaux protocoles introduits par ce service.

Extensibilité

L'approche serveur de noms logiquement centralisé part souvent de l'hypothèse qu'il est isolé. Cette hypothèse est de plus en plus fausse comme nous l'avons signalé auparavant. De même, l'interconnexion de systèmes préexistants exige souvent la coexistence côte à côte de conventions et syntaxes très différentes.

Au contraire, l'approche SDS explicitement réparti s'adapte mieux à cette interconnexion dans la mesure où, à la limite, un chemin d'accès peut contenir différentes parties interprétables selon différentes syntaxes. L'interprétation des noms n'est pas faite par un service unique mais déléguée vers les serveurs concernés.

Cohérence et fiabilité

Selon l'approche serveur de noms, la gestion des noms et des descripteurs est faite indépendamment de la gestion des entités qu'ils désignent. Ceci introduit un problème de cohérence. En effet, les clients d'un tel système doivent être préparés pour recevoir des informations fausses et doivent être capables de les détecter.

Par exemple, la suppression d'une entité exige la supression de son nom dans le serveur de noms. Si l'un des serveurs concernés entre en panne pendant l'opération, une incohérence peut rester quelque part dans le système et le serveur de noms peut considerer que l'entité existe toujours (ou vice-versa). De même, si le serveur de noms est en panne, il empêche l'accès aux entités qu'il désigne bien que les serveurs qui les gèrent soient disponibles.

Ces problèmes de cohérence sont diminués si les serveurs qui gèrent les entités gèrent aussi leurs noms comme le suggère l'approche SDS explicitement réparti.

Efficacité

Cet aspect est d'une grande importance. L'introduction dans le système d'un service spécialisé dans la désignation exige une interaction supplémentaire, donc un côut supplémentaire, entre un client et un autre serveur avant d'accéder à une entité. Ce côut supplémentaire est, dans beaucoup de situations, insupportable. Par exemple, à notre connaissance, très rarement les services de gestion de fichiers séparent complètement, par différents serveurs éclatés sur différentes machines, les fonctions de désignation des fichiers et des catalogues de fichiers et les fonctions de stockage.

7.4. Conclusions sur la comparaison des deux modèles

Les deux approches que nous venons de présenter pour l'interprétation des noms symboliques ont simultanément des avantages et des défauts. Elles sont complémentaires l'une de l'autre. Par exemple:

- Les systèmes de gestion de fichiers bâtis sur la notion de montage distant ont besoin de serveurs de noms "logiquement centralisés" pour désigner les usagers, les serveurs et les sites. Tel est, par exemple, le cas du service "yellow-pages" introduit avec NFS [Sandberg 85] ou du "Berkeley Name-server" [Terry 84]. Ces services de désignation offrent une base uniforme et cohérente pour résoudre les problèmes de désignation d'usagers, de sites, etc. et pour la protection.
- Souvent, les systèmes de gestion de fichiers répartis qui offrent la notion de "nom global" d'un fichier utilisent un serveur de noms "logiquement centralisé" pour gérer les catalogues de plus haut niveau de l'arbre et décentralisent ensuite la gestion de la désignation des différents sous-arbres par les différents serveurs de fichiers.

  Les seules règles semblent être les suivantes:

  1/ les serveurs de noms sont très adaptés aux réseaux généraux de systèmes hétérogènes et faiblement intégrés;

  2/ l'approche SDS explicitement réparti est plus adaptée aux systèmes intégrés et sous la même autorité administrative; en particulier, elle est, en général, d'un coût moins important et permet également une plus simple réutilisation du logiciel préexistant.

8. Résumé et conclusions sur la désignation dans les SER

Nous avons étudié dans ce chapitre les problèmes soulevés par la répartition en matière de désignation, à la lumière du cahier des charges présenté dans l'Introduction.

Du point de vue de la désignation, deux niveaux d'observation d'un système sont essentiels: le premier est caractérisé par le fait d'offrir des noms internes (grosso-modo des numéros calculés dynamiquement par le système) qui permettent l'accès aux ressources; le deuxième est caractérisé par le fait d'offrir des noms symboliques (grosso-modo des chaînes de caractères proposées par les clients) qui permettent la désignation de façon lisible, ainsi que la structuration de l'espace des entités présentes dans le système.

Deux sortes de noms internes sont en général disponibles dans les SER (systèmes d'exploitation répartis):

- les noms uniques et globaux (*Unique Identifiers*), des noms constants dans l'espace (globaux) et non réutilisés, i. e., constants dans le temps (uniques), tel que l'offre le V-System [Cheriton 84] par exemple; et
- les noms contextuels ou locaux (une sorte de capacité sur une ressource), des noms contextuels au processus que les utilise, tel que l'offre le système Accent [Rashid 81], par exemple.

Comme nous l'avons montré au §4, ces deux approches sont complémentaires; le choix de l'une ou de l'autre relève essentiellement des options prises:

- vis-à-vis de la protection et de la transmission et/ou héritage de noms,
- vis-à-vis de la simplicité et de la portabilité du système, et
- vis-à-vis du niveau des services offerts pour le traitement des exceptions dûes à l'inaccessibilté soudaine des ressources (un problème omniprésent en environnement réparti).

Au prochain chapitre, nous montrerons que ce choix a aussi des répercussions sur l'implantation de mécanismes d'édition de liens et de mise au point d'applications réparties. Le chapitre IV présentera nos options et les justifiera en face à cette analyse.

L'introduction d'un SDS (système de désignation symbolique) dans un SER, relève de deux problèmes distincts, mais interdépendants:

1/ Conventions de désignation symbolique

Quelles sont les conventions de désignation adoptées par le système: une "racine réseau" regroupant des graphes de désignation "locaux", une interconnexion arbitraire de graphes de désignation "locaux", ou un graphe unique qui n'est structuré que selon les besoins administratifs et non de localisation?

La notion de nom symbolique global (indépendant de la localisation) est souvent synonyme, ou du moins associée, à celle de SER dans la mesure où ces noms offrent la transparence du réseau. Néanmoins, pour que cette transparence soit effective, il faut que la notion de serveur soit aussi cachée aux usagers: les entités du système sont alors dupliquées dans plusieurs serveurs et des opérations atomiques (ou du moins concertées) permettent de maintenir la cohérence entre les différents serveurs qui gèrent une entité et/ou qui gèrent leur(s) nom(s).

Cet objectif étant difficile à atteindre, les SER actuels utilisent des approches intermédiaires dont le succès est surtout lié à une bonne définition de son cahier des charges, comportant un environnement d'application bien précis.

Notre analyse de ce problème (cf §6) nous permet de démontrer, au chapitre V, que l'approche intermédiaire prise par Chorus offre une grande souplesse de la configuration de son SDS. L'introduction de noms globaux, ou de n'importe quelles autres conventions de désignation, revient alors à un choix de paramétrisation du graphe de désignation en fonction du style de l'environnement que le système doit offrir.

2/ Structure interne du SDS

L'introduction d'un SDS dans un réseau de systèmes hétérogènes et faiblement intégrés est souvent réalisée par l'intermédiaire de serveurs de noms, tel que Grapevine [Birrell 82] par exemple. Ces services, spécialisés dans la désignation, introduisent une indirection supplémentaire entre les clients et les serveurs qui gèrent les ressources.

Une autre approche, qui est à l'opposé de la précédente, consiste à intégrer le sous-système de désignation symbolique des ressources aux serveurs qui les gèrent; cette approche est typiquement celle qui est prise par les serveurs de fichiers.

Au paragraphe 7 de ce chapitre nous illustrons, analysons et comparons ces deux approches. Au chapitre V nous montrerons que quelques extensions simples des serveurs de fichiers suffisent, dans certains environnements, pour qu'ils soient capables d'offrir quelques-uns des services caractéristiques des serveurs de noms. Telle est l'option de Chorus.

Après cette présentation caractérisant le système de désignation interne et le système de désignation symbolique d'un SER, nous allons traiter au chapitre suivant le problème de leur utilisation pour réaliser l'édition de liens en environnement réparti.

CHAPITRE II

L'EDITION DE LIENS
ENTRE PROCESSUS COMMUNICANT PAR MESSAGES

CHAPITRE II

L'EDITION DE LIENS

ENTRE PROCESSUS COMMUNICANT PAR MESSAGES

L'ensemble des entités auxquelles un processus peut accéder par l'échange de messages forme son environnement de communication ou contexte d'adressage pour la communication. De façon simple, ce contexte est l'ensemble des noms internes d'entités accessibles par l'échange de messages connus du processus. Sa construction, i. e., l'acquisition et le relâchement de ces noms par le processus, relève des méthodes d'édition de liens entre processus communicant par messages. Fondamentalement, ce contexte est de la même nature qu'un contexte d'adressage d'objets dans n'importe quel système d'exploitation. Quelles contraintes supplémentaires l'environnement réparti apporte à la caractérisation de l'environnement de communication d'un processus?

Dans des systèmes centralisés simples, toutes les interactions avec les ressources se font au travers du noyau du système. Celui-ci connaît toutes les entités présentes, contrôle et résout toutes les références mutuelles, centralise les informations nécessaires pour établir les liens. Dans ce cadre, s'il y a un arrêt du noyau, un processus ne peut plus faire évoluer son environnement; néanmoins, étant donné le caractère centralisé du système, la panne du noyau entraine la mort de tous les processus. Ainsi, en environnement centralisé, un processus peut généralement considérer que son environnement est stable et les pannes et autres erreurs dues à l'asynchronisme interne aux systèmes sont souvent ignorées. De même, le rechargement du système permet la ré-initialisation de l'état des processus et le nettoyage d'éventuelles inconsistances.

En environnement réparti il n'y a pas un noyau mais autant de noyaux que de sites; l'environnement d'un processus est formé de noms représentant des points d'accès à des ressources, à des services ou à des partenaires, dont le processus ignore la localisation. Cet environnement se caractérise par son instabilité et par l'impossibilité de calculer l'état exact des entités qui le composent à un moment donné. De même, on ne peut ni arrêter, ni redémarrer de façon synchrone tous les sites du système. L'environnement réparti suggère le besoin d'une sorte de "ramasse miettes" réparti, qui nettoye régulièrement toutes les inconsistances.

L'expérience des langages de programmation a montré que plus on retarde l'édition de liens, plus la reconfiguration, l'adaptation et la réutilisation du logiciel sont privilégiées. Comme nous le verrons par la suite, les caractéristiques de l'environnement réparti ne font que renforcer cette constatation.

Ce chapitre discute l'ensemble des méthodes d'édition de liens, leur rapport avec les méthodes de désignation utilisées et leur adéquation à la répartition. Cette discussion sera conduite en ayant en considération deux notions: la notion de groupe de processus

d'une application répartie et la notion de groupe de processus d'un service réparti.

1. Notion de groupe de processus d'une application répartie

Le langage CSP [Hoare 78] par exemple, est un langage d'expression de programmes où la notion essentielle qui permet la structuration d'un programme est celle de processus communicant par échange de messages. Dans ce langage, un programme est constitué par un groupe de processus qui s'échangent des messages.

Une vision informelle d'un groupe de processus d'un programme réparti, ou d'une application répartie, est celle d'un réseau de processus communicants s'exécutant sur plusieurs sites.

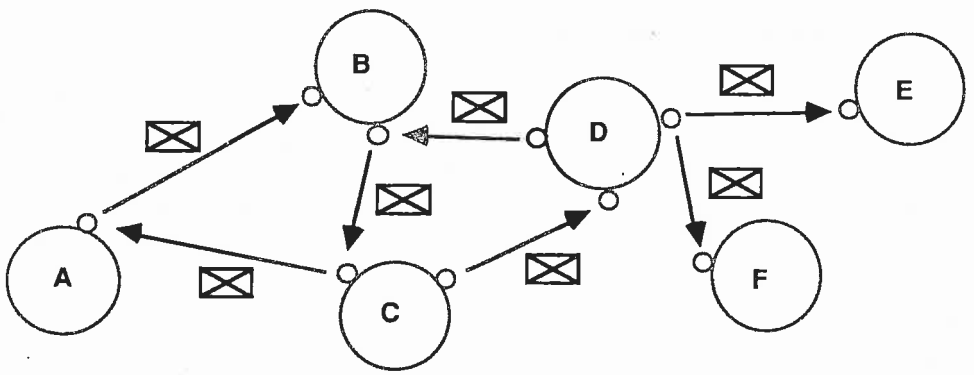


fig. 2.1 - Groupe de processus d'un programme réparti

L'environnement de communication de chaque processus du groupe est l'ensemble des points d'accès aux "opérations exportées" par les autres processus avec lesquels il communique.

Un autre exemple de composition de processus pour former un programme est celle suggérée par le *shell* d'Unix, cf annexe II. Son langage permet, par exemple, d'exprimer un *pipe-line* de processus.

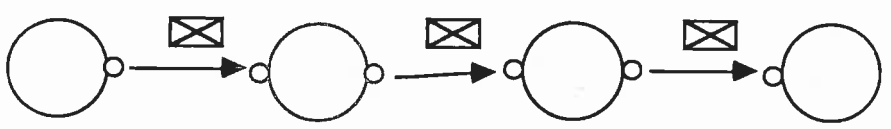


fig. 2.2 - Un *pipe-line* de processus

Un *pipe-line* s'exécutant simultanément sur plusieurs sites peut être assimilé à un groupe de processus d'un programme réparti.

Un SER (système d'exploitation réparti) doit offrir des mécanismes de base nécessaires pour l'exécution de tels groupes répartis de processus et pour réaliser

l'édition de liens entre ses différents composants. Il est indispensable que ces mécanismes répondent à des contraintes typiques de l'environnement réparti:

1/ La répartition des processus sur les différents sites doit pouvoir être ignorée et éventuellement calculée dynamiquement par le système.

2/ On doit pouvoir désigner le groupe par un seul identificateur et pouvoir appliquer des opérations à tout le groupe, comme par exemple la destruction de tous les processus qui l'intègrent quel que soient leurs localisations.

3/ L'application doit pouvoir être conçue et testée sur une configuration (un seul site par exemple) différente de celle où elle va s'exécuter plus tard.

4/ On doit disposer de mécanismes de mise au point ou de trace des interactions entre ses différents composants.

Analysons maintenant un cas particulier de groupe: le groupe de processus d'un service réparti.

2. Notion de groupe de processus d'un service réparti

Comme nous l'avons suggéré au paragraphe précédent, un processus peut être vu comme matérialisant une ou plusieurs ressources et "exportant" des points d'accès vers ces ressources. Si ce processus est généralement toujours disponible dans le système, c.a.d. s'il ne fait pas partie d'une application particulière, nous pouvons alors dire que ce processus rend de façon permanente des services aux autres processus.

Dans un SER existent en général plusieurs processus de ce type. Ils s'appellent serveurs. Pour des raisons de performance ou de résistance aux pannes, dans un SER, le même service est en général offert par plusieurs serveurs (un par site mais pas nécessairement). L'ensemble des processus qui rendent le même service constituent un cas particulier de groupe de processus réparti: ils constituent le groupe de processus d'un service réparti. Souvent, les composants d'un tel groupe exécutent le même code et, ce qui est encore plus important du point de vue de l'édition de liens, ils ne sont pas tous issus du même père. Ils sont chargés/arrêtés suivant l'arrêt/redémarrage des différents sites du système.

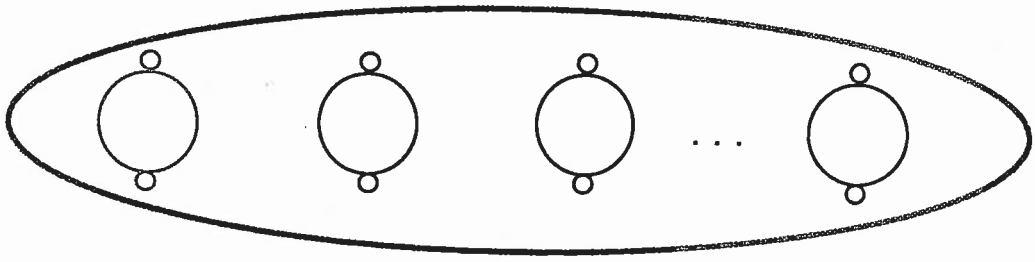


fig. 2.3 - Un groupe de processus d'un service réparti

Ce cas particulier de groupe réparti de processus possède des exigences spécifiques:

1/ Un processus intégrant le service doit à chaque moment être capable de s'adresser simultanément à tous ses partenaires. Cette forme d'adressage spéciale s'appelle générique; elle doit être possible sans que le processus soit obligé de connaître à tout moment la liste complète de ses partenaires parce

qu'elle peut varier dynamiquement.

Les SER mettent en oeuvre ces possibilités au travers des notions de groupe (symbolique, de portes ou de processus) et de diffusion de messages.

2/ Un client du service doit pouvoir s'adresser à celui-ci en contactant de façon transparente un de ses serveurs. De préférence le serveur qui lui est le plus proche. Cependant, le client doit pouvoir voir tous les serveurs comme une entité logiquement unique.

Cette forme d'adressage spéciale s'appelle fonctionnel. Elle doit être possible sans que le client soit obligé de connaître à tout moment la liste complète des partenaires qui rendent le service et sans qu'il soit obligé de choisir explicitement le partenaire qu'il veut contacter.

Nous pouvons assimiler cette forme d'adressage à l'utilisation des numéros de téléphone de la police ou des urgences. Le même numéro appelle différentes agences du service selon la localisation de l'appelant.

3/ On doit pouvoir changer dynamiquement l'ensemble des serveurs qui coopèrent pour rendre un service, sans avoir à arrêter les clients ou les serveurs qui ne sont pas concernés par la reconfiguration.

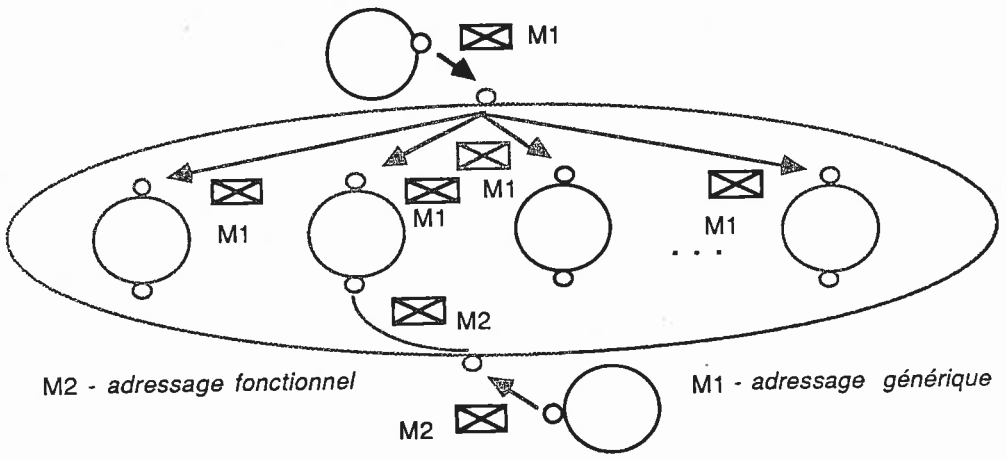


fig. 2.4 - Adressages générique et fonctionnel d'un service réparti

Souvent, quelques-uns des services offerts par les noyaux des SER sont aussi accessibles par le biais de l'échange de messages. Tel est le cas des noyaux des systèmes DEMOS/MP, Accent ou Chorus, par exemple. Dans un tel cas de figure, l'ensemble des noyaux d'un SER peut être aussi vu comme un service réparti implanté par un groupe d'entités.

Analysons maintenant, à partir de ces exemples, les différentes méthodes d'édition de liens en mettant en évidence leur prix et leur adéquation à l'environnement réparti†.

† En introduisant ces exemples de groupes répartis de processus communicants par messages, nous ne prétendons pas réduire tous les SER à cette vision. Leur introduction ne sert qu'à conduire cette discussion sur l'édition de liens. Il y a des SER qui cachent à leurs clients l'échange de messages comme par exemple les systèmes Eden [Almes 85] ou Cedar [Birrell 84]; ces systèmes ne l'utilisent, en général, qu'à un certain niveau de leur architecture interne.

Nous distinguons les méthodes suivantes: l'édition de liens statique, l'édition de liens au chargement et l'édition de liens à la demande.

3. Edition de liens statique

L'édition de liens statique est caractérisée ainsi: les noms internes des partenaires d'un processus communicant sont des constantes définies au moment de la compilation. On peut aussi l'appeler édition de liens au moment de la compilation.

Cette méthode présente les propriétés suivantes:

- elle est implicite du point de vue des processus;
- elle est aussi la moins chère dans la mesure où il n'y a aucune indirection à résoudre;
- étant donné que les noms sont définis au moment de la compilation, elle ne permet pas la reconfiguration des applications car elle exige la recompilation des composants qui l'utilisent;
- l'assimilation des noms à des constantes suggère leur dispersion dans les données des processus; cette méthode n'est donc compatible qu'avec la vision directe par les processus des UID (*unique identifiers*) ou des adresses réseau des partenaires.

En effet, même si le système cache les adresses réseau derrière des UID, l'édition de liens statique ne peut se faire qu'avec des UID pré-alloués ou prédéfinis.

Toutes ces raisons font de l'édition de liens statique une méthode souvent interdite aux clients du système. Elle est en général réservée à des niveaux très internes du système; par exemple, pour l'accès aux services rendus par le noyau ou par des serveurs dont la présence sur un site est indispensable à l'exécution d'un processus. Son utilisation doit être considérée comme exceptionnelle dans la mesure où elle limite la portabilité. En effet, le changement de la topographie du réseau, ou de la configuration du système, peut invalider les composants qui utilisent des noms internes connus statiquement.

4. Edition de liens au chargement

L'édition de liens au chargement est caractérisée ainsi: les noms internes des partenaires d'un processus lui sont transmis à sa naissance par le biais de ses paramètres. Ces noms sont donc assimilables à des variables initialisées au chargement.

Cette méthode présente les propriétés suivantes:

- elle est aussi implicite du point de vue des processus;
- mais elle est plus chère que l'édition de liens statique car elle implique le passage de paramètres au chargement des processus;
- elle permet qu'une application répartie soit conçue et codée indépendamment de la configuration où elle va s'exécuter, car l'établissement des liaisons concrètes sous-jacentes au réseau logique de processus qui compose l'application ne se fait qu'au moment de son chargement.
- elle permet la reconfiguration statique des applications, i. e., avant leur chargement, sans obliger la recompilation de leurs composants.

Cette méthode d'édition de liens suggère des possibilités intéressantes que nous décrivons ci-dessous.

4.1. Relations entre désignation interne contextuelle et édition de liens

Dans les systèmes où l'espace d'adressage pour la communication d'un processus est contrôlé par le système, i. e., où l'adressage se fait indirectement par des noms internes contextuels ou locaux, l'ensemble des noms internes connus d'un processus est répertorié par le système dans une table d'adressage ou contexte d'adressage pour la communication, inclus dans le contexte d'exécution du processus (cf §4 chapitre I).

Dans ces systèmes, le créateur d'un processus, i. e., son père, peut spécifier à la création quel est le contexte initial du fils. On dit alors qu'il y a héritage de noms (ou de capacités d'adressage) de père en fils.

Par exemple, dans le système Accent (cf §4.6 chapitre I) les capacités et les droits sur une porte peuvent être passés de père en fils. La primitive *fork* de ce système:

Fork (..., Liste de Portes) → {Père, Fils}

permet au père de spécifier quelles sont les portes initiales du fils, ainsi que les droits associés.

Cette fonctionnalité réalise intrinsèquement l'édition de liens au chargement dans ce système. Par exemple, un processus initial peut commencer par créer toutes les portes nécessaires au réseau de processus de l'application et ensuite charger chacun des processus qui la composent. Ces processus reçoivent du processus initial les portes et les droits (soit de réception, soit d'émission) caractéristiques de la visibilité qu'ils ont, ou qu'ils offrent, à leurs partenaires. Le rôle du processus initial est le chargement du réseau, il peut, donc, se terminer une fois ce chargement réalisé.

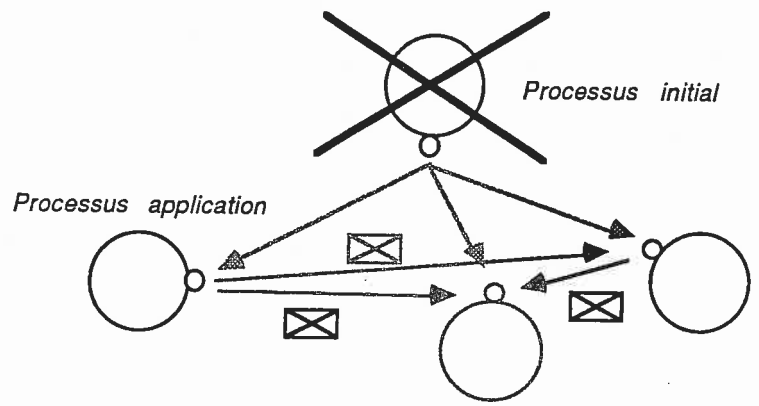


fig. 2.5 - Chargement d'un réseau de processus communicants

Le partage des fichiers ouverts entre un processus père et ses fils, tel que le permet le système Unix en environnement centralisé, par exemple (cf annexe II), est aussi mis en oeuvre par une sorte d'édition de liens au chargement. Cet héritage, suppose aussi une méthode de désignation contextuelle des fichiers ouverts par un processus.

Les *pipe-lines* de processus que l'interpréteur de commandes du système Unix (le *shell*) est capable de charger en environnement centralisé, sont aussi construits par cet utilitaire en utilisant l'héritage de noms au chargement. L'interpréteur de commandes d'Unix prend, dans ce cas de figure, le rôle que nous venons d'attribuer au processus initial.

Mais, en utilisant la même téchnique, i. e., le chargement d'une application répartie par un processus initial, un autre scénario peut être envisagé:

Pour chaque porte P1 du réseau, le processus initial crée deux portes: la porte P1 et une porte P1'. Chaque processus ayant besoin d'adresser P1 reçoit la capacité en émission sur P1'. Le processus initial garde la capacité en réception sur P1'. La capacité en réception sur P1 est effectivement passée au processus qui rend le service accessible au travers de cette porte.

Au lieu de se détruire à la fin du chargement des fils, le processus initial intègre le réseau et chaque message qu'il reçoit sur P1' est redirigé sur P1 au nom de la porte émettrice qui représente chez lui l'émetteur, par exemple P2.

Tous les messages échangés entre P1 et P2 passent alors par le processus initial qui peut fonctionner comme une sorte d'outil de trace et d'aide à la mise au point.

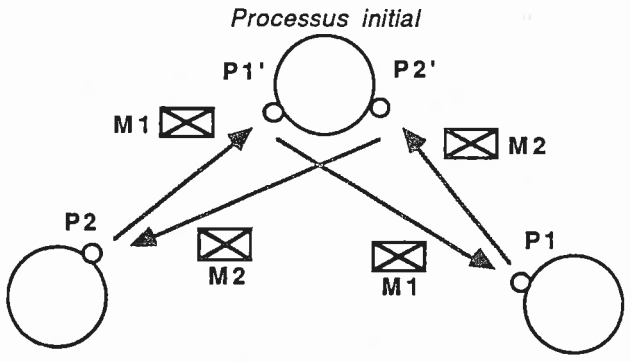


fig. 2.6 - Le processus initial contrôle les communications entre les portes P1 et P2

L'édition de liens au chargement peut donc servir de base à des fonctionnalités intéressantes. Le complément naturel de ces fonctionnalités est celui de langage de configuration de programmes répartis.

4.2. Langage de configuration de programmes répartis

L'établissement de liens au chargement des composants d'une application répartie suggère qu'une application puisse être configurée interactivement, au moment de son chargement, à partir d'un ensemble de composants choisis parmi ceux disponibles dans une sorte de "boîte à outils". On trouve dans [Sloman 85] une approche de ce type.

Cette idée est matérialisée par le *shell* d'Unix pour un cas particulier de réseaux de processus communicants. Sa généralisation est intéressante. Ce langage devrait permettre:

- de spécifier un réseau réparti de processus communicants;
- d'encapsuler un réseau qui ne serait accessible que par les "points d'accès exportés"; ce nouveau réseau devrait être assimilable de l'extérieur à un processus unique;
- de charger un réseau de processus;

- de spécifier (explicitement ou implicitement) la répartition des différents processus par le système.

Naturellement, l'ensemble des processus qui intègrent une application répartie doit constituer un groupe de processus; ce groupe, doit permettre l'application d'opérations de destruction, d'arrêt (*pause*) ou de redémarrage (*resume*) de l'ensemble des processus de l'application.

Après cette remarque nous passons à la discussion des méthodes d'édition de liens qui se caractérisent par l'acquisition des noms pendant l'exécution des processus, en fonction de leurs besoins d'adressage. Ces méthodes sont aussi appelées d'édition de liens dynamique.

5. Edition de liens à la demande

L'édition de liens à la demande peut être caractérisée ainsi: l'acquisition du nom interne d'un partenaire, ou du point d'accès à une ressource, se fait par demande explicite du processus.

Cette méthode est celle qui est typiquement utilisée pour l'accès aux ressources offertes par un serveur. Par exemple, l'ouverture d'un fichier passe par une édition de liens à la demande.

Elle présente les propriétés suivantes:

- elle est explicitement déclanchée par les processus;
- elle est plus chère que les méthodes précédentes dans la mesure où elle exige une indirection entre un nom, en général symbolique, et un nom interne;
- elle est plus adéquate à l'environnement réparti dans la mesure où l'édition de liens est retardée jusqu'au moment d'accéder à la ressource ou au partenaire.

Nous en illustrons ensuite son utilisation dans le cadre du projet Cedar.

5.1. Illustration: édition de liens à la demande dans le cadre du projet Cedar

Le projet Cedar [Teitelman 84] a implanté un environnement de programmation réparti sur le même réseau où s'exécute Grapevine (cf §7.1 chapitre I). Cedar permet l'appel de procédures à distance (*Remote Procedure Call* ou RPC) afin de permettre à un processus d'appeler des procédures s'exécutant dans un autre processus d'une autre machine [Birrell 84].

Le langage Mesa [Mitchell 78], un des langages disponibles dans Cedar, possède la notion de module d'interface qui est la base de la compilation séparée sûre (tel que Ada ou Modula2 possèdent la notion de module de spécification). Ce module contient l'ensemble des déclarations exportées par le module d'implémentation. Il est suffisant pour que l'importateur et l'exportateur réalisent indépendamment la compilation de façon sûre.

Associé à chaque service accessible par des RPC il y a un module d'interface qui décrit son interface. Cedar fournit les outils pour qu'à partir d'un module d'interface deux autres modules soient générés: le *User-stub* et le *Server-stub*.

Module d'interface → *User-stub*, *Server-stub*

Le code d'un serveur associé au service est lié avec le module *server-stub*; celui d'un client du service avec le *user-stub*; le *server-stub* émule dans le serveur les appels des clients; le *user-stub* émule dans le client les procédures du serveur.

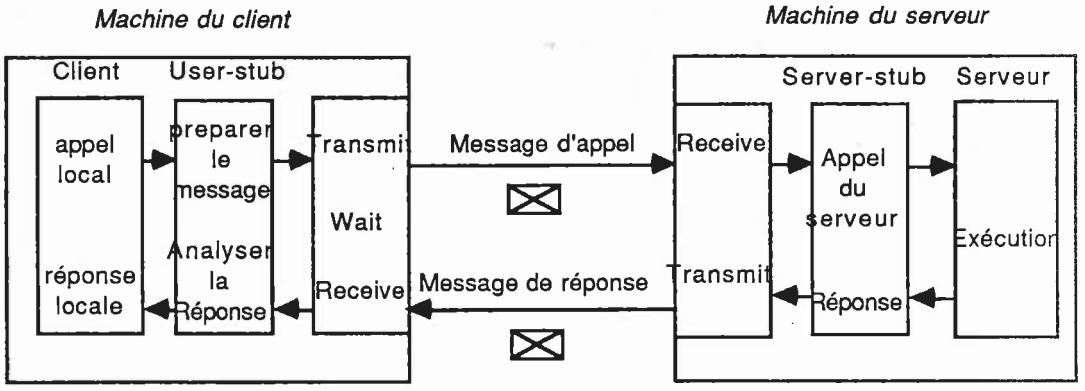


fig. 2.7 - Appels distants dans Cedar

Le processus client appelle la procédure distante en l'appelant dans son *stub*; celui-ci transforme cet appel dans un envoi de message vers le serveur; ce message est traité par le *server-stub* qui appelle le serveur en simulant un appel procédural conventionnel; finalement, le *server-stub* renvoie la réponse du serveur, dans un message, au *user-stub* qui retourne normalement dans le code du client avec la réponse à l'appel. Ces deux modules, le *user-stub* et le *server-stub*, jouent donc également le rôle d'une sorte de table de références externes entre processus.

L'établissement de liaisons entre le client et le serveur concerne, d'une part, la désignation symbolique du service et des serveurs, et d'autre part, l'obtention d'un point d'accès à un serveur particulier.

Pour résoudre la première, Cedar utilise Grapevine: chaque interface exportée accessible par des RPC possède dans Grapevine une entrée du type groupe, associée (cf §7.1 chapitre I). Ce groupe regroupe les noms de tous les serveurs qui exportent le service correspondant à l'interface. Il désigne donc le service. En général, un service est rendu par plus d'un serveur: chaque serveur est désigné et décrit à l'aide de Grapevine par une entrée du type *individu* (cf §7.1 chapitre I). Parmi les attributs de chaque *individu* on trouve son adresse réseau. Par conséquence, associé au nom de chaque serveur il y a son adresse réseau.

Par exemple, le service de gestion de fichiers est désigné par le groupe "fs.inria" qui contient la liste des noms des serveurs de fichiers: "fs-bat11.inria", "fs-bat34.inria", "fs-gip.inria", etc. Associé à chaque serveur il y a son adresse réseau; par exemple, "fs-bat11.inria" est à l'adresse: "5#86#".

Quand un serveur commence son exécution, il appelle la procédure standard (incluse automatiquement dans le *server-stub*):

ExportInterface (Interface, Serveur, @dispatcher)

par exemple:

ExportInterface ("fs.inria", "fs-bat11.inria", @dispatcher);

Cette procédure déclenche une invocation de Grapevine pour réaliser des contrôles de protection et pour mettre à jour l'adresse actuelle du serveur; elle enregistre aussi, dans une table de la machine où le serveur s'exécute, un UID de l'instance actuelle du serveur ainsi que l'adresse de la procédure *dispatcher*. Cette procédure, incluse automatiquement dans le *server-stub*, a pour rôle de transformer les messages adressés à cet UID en un appel d'une procédure exportée par le serveur. L'UID est généré en concaténant l'adresse actuelle du serveur avec l'heure de l'appel de "ExportInterface"; par exemple: "5#86#0459290034".

Un processus qui désire utiliser un service accessible par des RPC doit avoir été lié avec le *user-stub* correspondant à l'interface du service. Avant de faire le premier appel au service, il doit appeler la procédure standard (incluse automatiquement dans le *user-stub*):

ImportInterface (Interface, Serveur)

par exemple:

ImportInterface ("fs.inria", "fs-bat11.inria");

Cette procédure déclenche une consultation de Grapevine pour obtenir l'adresse du serveur; ensuite, avec cette adresse, elle contacte directement un module de la machine où s'exécute le serveur pour obtenir son UID actuel. Cet UID est mémorisé par le *user-stub* parce qu'il sera utilisé par la suite.

Une fois l'édition de liens à la demande accomplie, chaque appel d'un service distant est transformé par le *user-stub* correspondant en un envoi de message vers l'adresse du serveur associé; ce message contiendra comme premier paramètre l'UID de l'instance actuelle du serveur et le numéro d'ordre de la procédure appelée. L'UID sert pour détecter si l'on continue toujours à utiliser la même instance du serveur; le numéro d'ordre de la procédure appelée sert pour que la procédure *dispatcher* du *server-stub* puisse choisir la procédure correspondante qui doit être appelée dans le serveur.

Cette illustration fait la synthèse de plusieurs aspects que nous venons de discuter des le début de cet ouvrage: on trouve plusieures couches de désignation, une symbolique, une autre interne (les UID) et les adresses réseaux; les UID cachent les adresses réseau pour rendre sûre la désignation du récepteur d'un message; l'édition de liens est faite à la demande juste avant l'appel des services; un système de désignation symbolique est utilisé pour rendre possible la désignation, l'édition de liens à la demande, l'administration et la structuration des services; la notion de groupe permet aux clients d'avoir une vision unique d'un service qui est désigné par un seul nom ("fs.inria" par exemple) - ce nom représente le collectif de tous les serveurs participant au service.

En effet, Grapevine et cette méthode d'édition de liens permettent que le service soit reconfiguré de façon cachée aux serveurs et clients qui ne sont pas concernés. D'autre part, les règles de protection de Grapevine servent pour protéger l'accès aux noms des serveurs et pour protéger la configuration du service en général. Ces contrôles empêchent, par exemple, qu'un processus utilisateur ne se déclare comme serveur de messagerie afin d'espionner les messages des autres usagers.

Finalement, il reste à signaler que la procédure qui effectue l'édition de liens chez le client peut être appelée sans spécifier quel serveur de l'interface le client veut utiliser; par exemple, l'appel:

ImportInterface ("fs.inria", "");

signifie que le client ne veut pas se préocuper de l'identité du serveur qu'il veut utiliser c'est le système qui en choisit un, de forme implicite, parmi ceux qui appartiennent au groupe "fs.inria". Il s'agit d'une sorte d'adressage fonctionnel dont l'introduction est faite à l'aide de la notion de groupe.

Avec la présentation de cette illustration nous avons terminé la présentation des méthodes d'édition de liens. Le paragraphe suivant revient à la discussion de la notion de groupe et à ses implications sur l'édition de liens.

6. La notion de groupe et l'édition de liens

Il y a trois manières d'introduire la notion de groupe dans les SER. Elles sont présentées ci-dessous selon la couche de désignation où elles s'intègrent:

1/ La notion de groupe symbolique comme l'offre Grapevine. Un groupe symbolique est un ensemble de noms symboliques. Cette notion sert à l'adressage générique symbolique (diffusion de courrier éléctronique), à la constitution de listes d'entités qui partagent un droit (listes de contrôle d'accès) et à la désignation symbolique de services (comme nous l'avons illustré ci-dessus).

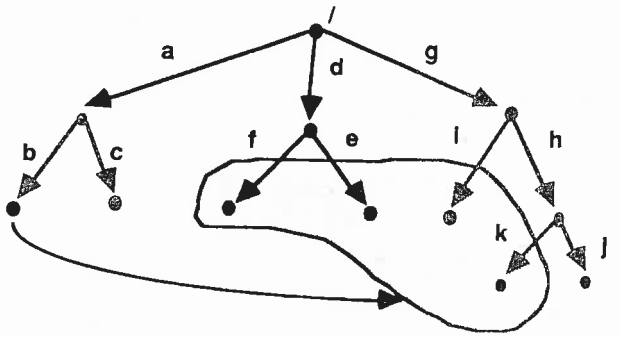
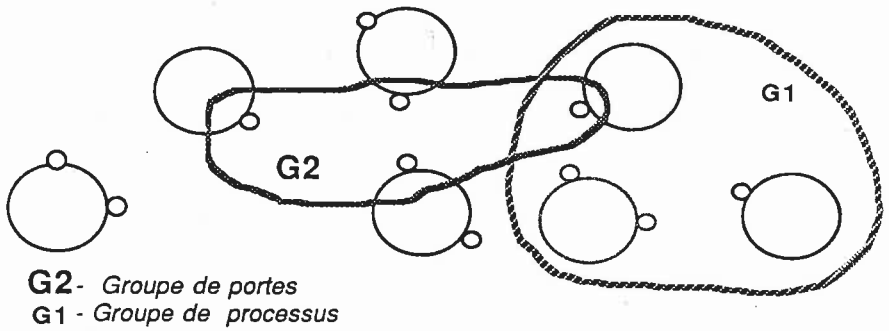


fig. 2.8 - Un groupe symbolique

2/ Les notions de groupe de processus ou de groupe de portes comme l'offrent le V-System [Cheriton 85] ou le système Chorus (cf chapitre IV). Ces groupes constituent un ensemble homogène d'entités avec un nom interne unique. Ces fonctionnalités servent pour la diffusion de messages (adressage générique), l'édition de liens (adressage fonctionnel) et l'application d'opérations génériques (comme par exemple la destruction en une seule opération de tous les membres d'un groupe de processus).



G2 - *Groupe de portes*
G1 - *Groupe de processus*

fig. 2.9 - Groupes de processus et groupes de portes

3/ La notion d'adresse de ***Brodcasting*** ou de ***Multicasting*** comme l'offrent les réseaux locaux. Ce niveau, même s'il est souvent directement visible des clients directs d'un réseau, ne sert, dans les SER, qu'à l'implantation des niveaux précédents.

Nous avons vu ci-dessus, à-propos de l'édition de liens à la demande, que la notion de groupe symbolique peut servir à désigner symboliquement une fonctionnalité, un service ou une interface. Elle introduit une indirection supplémentaire dans le processus d'édition de liens: le client peut choisir explicitement, ou le système implicitement, un membre arbitraire du groupe afin d'obtenir le service. Cette approche est intéressante dans la mesure où elle rend plus souple l'accès et la configuration du service.

La diffusion de messages à un groupe est en général importante pour l'implantation de services répartis (comme par exemple les bases de données réparties) ainsi que pour l'implantation de services ou d'applications exigeant la diffusion de messages (téléconférences, calcul parallèle réparti, etc.).

Ce type d'adressage est aussi particulièrement intéressant si le groupe comporte des entités offrant la même fonctionnalité, ou gérant le même objet, mais le client ne connaît pas un membre particulier à qui s'adresser.

Par exemple, un client voulant ouvrir un fichier F dont il ignore la localisation, peut diffuser sa requête d'ouverture à l'ensemble des serveurs de fichiers ; seul le serveur qui gère F répond au client. Il s'agit d'un protocole de demande de 1 à N avec une seule réponse.

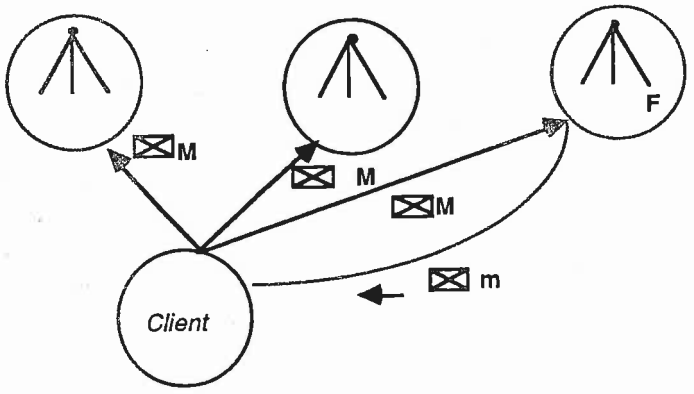


fig. 2.10 - Localisation du fichier F par adressage générique

Du point de vue de la duplication des ressources, la diffusion ou adressage générique peut s'avérer aussi intéressante. Si un fichier F est dupliqué par N serveurs, un client voulant ouvrir F peut adresser l'ensemble des serveurs, exécuter un vote majoritaire sur les réponses reçues, et choisir parmi l'ensemble des serveurs un de ceux qui gèrent la version la plus récente.

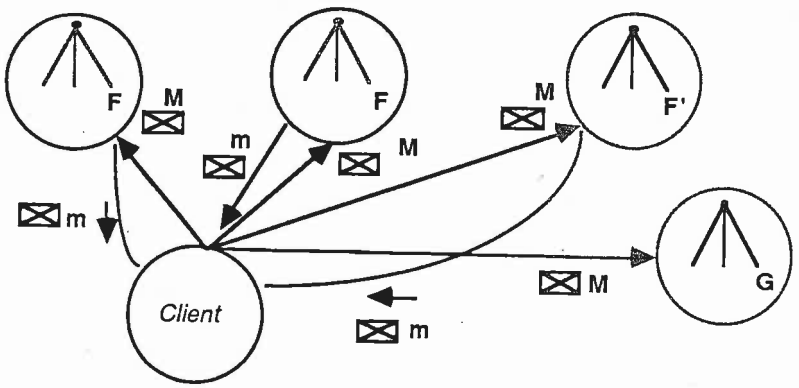


fig. 2.11 - Vote majoritaire pour l'ouverture d'un fichier dupliqué

Finalement, un client peut vouloir obtenir un service tout en ignorant l'identité du serveur qui le rend, à condition que ce serveur réalise le service. Par exemple, le client veut imprimer un fichier mais l'imprimante sélectionnée lui est indifférente. Nous appelons cette forme d'adressage spécial, l'adressage fonctionnel ou adressage de "1 à (1 parmi N)".

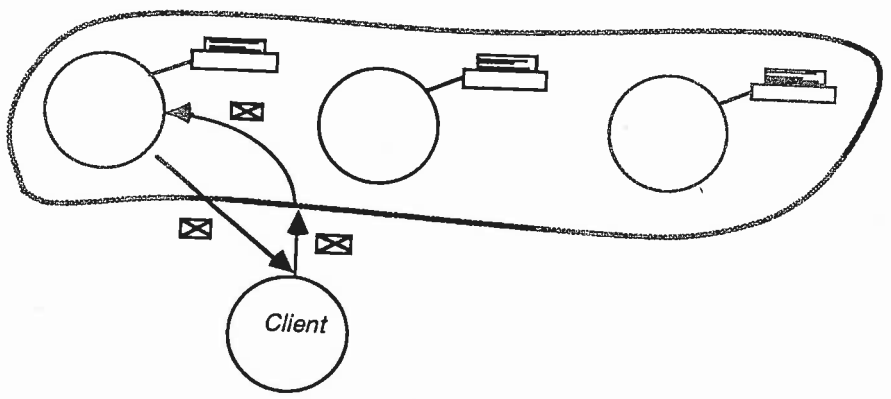


fig. 2.12 - Impression d'un fichier par adressage de 1 à (1 parmi N)

En guise de conclusion, nous pouvons résumer l'intérêt des groupes pour l'édition de liens par sa capacité à représenter la notion de fonctionnalité avec des points d'accès multiples.

7. Récapitulatif

L'environnement réparti est caractérisé par l'omniprésence de l'asynchronisme qui, plus qu'en environnement centralisé, rend difficile l'évaluation de l'état exact du système à un instant donné. L'asynchronisme des changements d'état des différents sites, dûs aux pannes ou aux reconfigurations, et les temps de propagation des messages dans le réseau empêchent un algorithme faisant ce calcul d'exister. L'environnement réparti doit donc privilégier les méthodes dynamiques d'édition de liens.

Ce chapitre passe en revue les différentes méthodes d'édition de liens entre processus communicants, en les classant selon le moment où l'édition de liens prend effet, et met en évidence les rélations existantes entre ce moment et les méthodes de désignation utilisées.

Il montre que l'édition de liens au chargement est la méthode la plus adéquate au chargement, contrôle et mise au point d'applications réparties; celles-ci étant caractérisées, dans ce chapitre, comme un réseau de processus communicants qui est chargé pour réaliser l'pplication.

L'édition de liens dynamique, ou à la demande, est la méthode caractéristique de l'obtention par les clients de l'accès aux services répartis; ces derniers étant caractérisés, dans ce chapitre, comme des réseaux de processus communicants dont le chargement et reconfiguration sont essentiellement dictés par des besoins administratifs, d'éfficacité ou de résistance aux pannes.

Finalement, nous avons mis en évidence les rélations étroites qu'il y a, en environnement réparti, entre la notion de groupe et l'édition de liens. En effet, la répartition met en valeur la notion de groupe d'une manière nouvelle. Les groupes permettent de matérialiser la duplication des ressources et donc de rendre réelles des

avantages de la répartition, tels que la fiabilité et l'éfficacité. Du point de vue de l'édition de liens, le problème de l'accès à un service dupliqué est, pour l'essentiel, celui de trouver un point d'accès à une des instances du service.

Chorus offre directement, ou permet l'implantation, de tous les mécanismes identifiés dans ce chapitre. La présentation de ses options est l'objet des chapitres suivants.

CHAPITRE III

DESCRIPTION GENERALE DU SYSTEME CHORUS

CHAPITRE III

DESCRIPTION GENERALE DU SYSTEME CHORUS

Ce chapitre donne un aperçu des caractéristiques essentielles du système Chorus. Le premier paragraphe introduit les notions de base qui caractérisent son architecture interne; le deuxième donne un aperçu de l'organisation du système d'exploitation bâti sur cette architecture; finalement, le troisième présente les aspects essentiels des options de Chorus pour la protection.

1. Notions de base de l'architecture Chorus

Selon l'architecture Chorus un système réparti est un ensemble de sites interconnectés par un réseau. Sur chaque site, les entités actives sont des acteurs. Un acteur est, en première approximation, l'équivalent d'un processus séquentiel. Un acteur peut créer et détruire d'autres acteurs sur son site ou sur d'autres sites; il peut communiquer avec d'autres acteurs, locaux ou distants, en échangeant avec eux des messages à travers des portes. Un acteur peut aussi créer et détruire des portes; acteurs et portes peuvent être associés dynamiquement. Nous présentons ensuite les éléments fondamentaux de cette architecture.

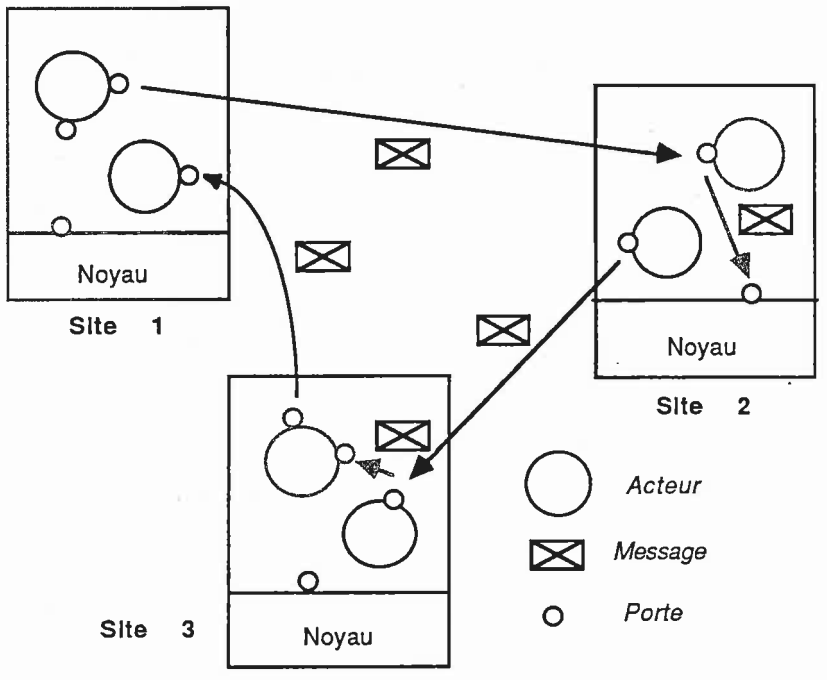


fig. 3.1 - L'architecture Chorus: acteurs, portes et messages

1.1. Les acteurs

Un acteur, tel un processus séquentiel, contient du code, des données et un contexte d'exécution. Son exécution est déclenchée sur réception d'un message. Le seul moyen de synchronisation entre acteurs est l'échange de messages.

Les acteurs sont structurés en étapes de traitement, délimitées par un point d'entrée et un point de retour. Ce point de retour est un point d'entrée dans le noyau du système. Cette structuration permet, par exemple, d'assimiler un acteur à un "type de données abstrait actif", avec une étape d'initialisation exécutée automatiquement à sa naissance, et d'autres étapes de traitement déclenchées par l'arrivée de messages.

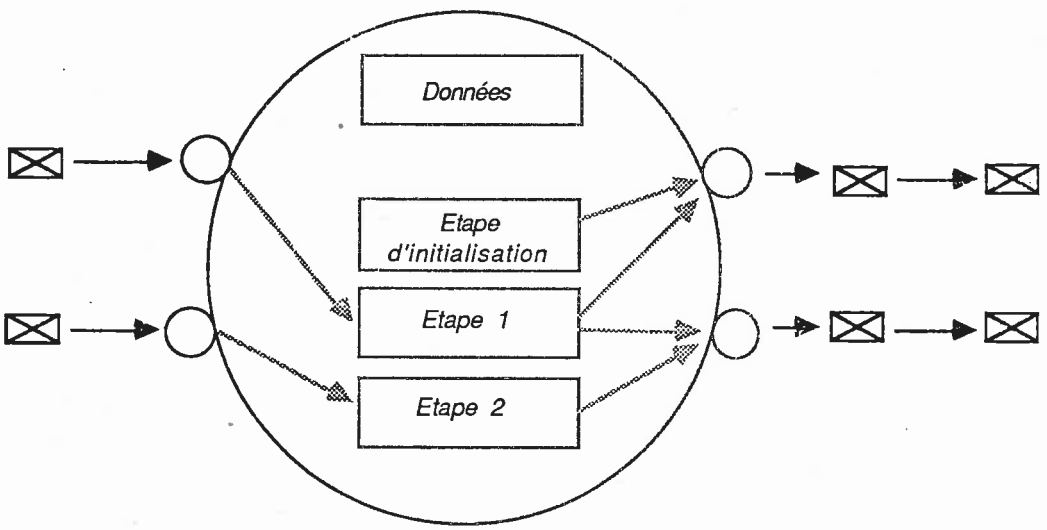


fig. 3.2 - Structuration interne d'un acteur

Pour permettre l'implantation de services résistants aux pannes, un acteur évolue comme une séquence d'étapes de traitement. Il ne peut prendre en compte d'autres messages tant que l'étape de traitement en cours n'est pas terminée. Symétriquement, les messages préparés pendant l'étape de traitement ne sont transmis que si l'étape de traitement se termine correctement. Ainsi, le récepteur d'un message est sûr que l'étape de traitement qui l'a émis s'est bien passée.

Pendant l'exécution d'une étape de traitement l'acteur peut invoquer les services caractéristiques de la machine logique Chorus.

Envoi	pour demander l'émission d'un message
Selection	pour positionner des conditions de sélection des prochains messages à traiter
Aiguillage	pour associer une porte à un point d'entrée d'une étape de traitement
Temporisation	pour armer une temporisation

fig. 3.3 - Opérations de la machine logique Chorus

La succession des étapes de traitement d'un acteur dépend d'une part des messages qu'il reçoit sur ces portes et d'autre part des sélections et aiguillages définis par l'acteur: les sélections sont une liste de liaisons représentées chacune par un couple (porte émettrice, porte réceptrice locale); seul un message reçu sur une de ces liaisons peut déclencher l'étape de traitement suivante. Le choix entre plusieurs messages candidats à la sélection respecte les priorités des portes réceptrices et, sur chaque liaison, les messages sont traités dans leur ordre d'arrivée.

L'acteur définit également des aiguillages (porte locale P, étape de traitement) qui spécifient quel traitement sera exécuté si le message sélectionné a été reçu sur la porte P. Ces aiguillages peuvent être modifiés dynamiquement.

Par exemple, si un acteur A sélectionne le couple (Q, P) et aiguille P vers le traitement T, un message émis de la porte Q et reçu sur la porte P de l'acteur A peut être sélectionné et il déclenchera l'exécution de T.

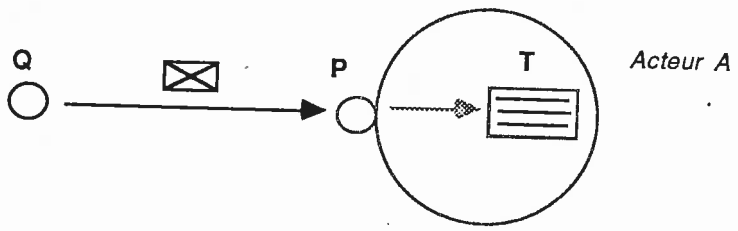


fig. 3.4 - Sélection et aiguillage

1.2. Les portes

Au niveau de l'architecture Chorus la seule interface de communication entre les acteurs est faite par le biais de portes. Afin de recevoir un message tout acteur doit posséder au moins une porte. Inversement, l'émission d'un message par un acteur s'effectue depuis l'une de ses portes.

Les acteurs ne se connaissent pas en tant que tels dans le système: un acteur ne connaît du système que ses portes et un ensemble de portes sur lesquelles il peut envoyer des messages. La désignation de l'émetteur et du récepteur est donc indirecte.

Le désignation des portes est indépendante de sa localisation et le Service de Transport assure la transmission des messages vers les portes de façon transparente, quelle que soit leur localisation.

Il n'y a pas de liaison pré-établie entre les portes; les messages sont, à un niveau élémentaire, échangés en mode "sans connexion". Le service de transport fourni aux acteurs par le système, est indépendant de leurs localisations: il n'y a aucune différence, pour un acteur, entre une communication locale et une communication distante.

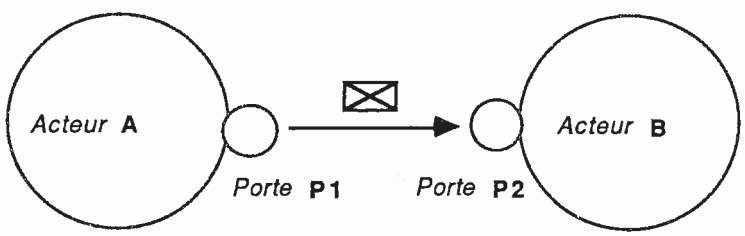


fig. 3.5 - Les messages sont échangés entre des portes

L'association entre les acteurs et les portes est dynamique: lorsqu'un acteur A ouvre

une porte P, il s'attache cette porte et peut, alors seulement, l'utiliser pour émettre et recevoir des messages. Lorsque la porte P est fermée, un autre acteur B peut l'ouvrir à son tour (aux restrictions de protection près). Une même porte peut ainsi être successivement associée à différents acteurs (un seul à la fois).

Ces choix de Chorus offrent plusieurs avantages:

- L'implantation des services disponibles derrière les portes peuvent être changés, sans aucun effet sur leurs clients, à condition que l'interface et le protocole d'accès ne changent pas.
- Pour des raisons de performance ou de résistance aux pannes, la reconfiguration des services est possible de façon transparente aux clients.
- La localisation est transparente et les communications locales ou distantes sont uniformes.

La gestion, la désignation et l'utilisation des portes, ainsi que des groupes de portes, est discutée en détail au chapitre IV.

1.3. Les messages

Un message est un ensemble de données qui sort de l'espace d'adressage de l'émetteur quand il est émis, et qui entre dans l'espace d'adressage du récepteur à la réception. Ils sont des objets identifiés et protégés.

Le cycle de vie d'un message ne se limite pas à son émission et à sa réception: un message peut être créé par un acteur A, qui peut l'envoyer à un acteur B, qui peut le rediriger vers un acteur C, qui finalement peut, par exemple, le renvoyer de nouveau à l'acteur A. Le message n'est détruit que par appel explicite de l'acteur qui le possède à un moment donné.

A la création d'un message, il reçoit un identificateur global, unique et non réutilisé: le numéro de séquence du message; ce numéro l'identifiera tant qu'il existe. A tout moment, un acteur ayant accès à un message peut obtenir ce numéro. Cet estampillage unique et global a comme but l'implantation de protocoles. En effet, comme le message conserve toujours son identificateur, même quand il transite d'un acteur à l'autre, cet identificateur peut servir à identifier une transaction entre plusieurs partenaires.

Un service de temporisation permet à un acteur de limiter le temps d'attente d'un message. L'arrivée d'une temporisation à échéance est perçue par l'acteur comme l'arrivée d'un message spécial qui lui est envoyé directement par le système.

2. Description sommaire du système d'exploitation

Le système Chorus est bâti sur les concepts de l'architecture que nous venons de présenter. Ce paragraphe décrit son interface et sa structure interne, ainsi que quelques aspects de son implantation sur la machine SM90.

2.1. Les services offerts par le système

L'interface système

L'interface du système est totalement indépendante de la localisation. Elle est un sur-ensemble de celle du système Unix; elle comporte des fonctions système permettant:

- la gestion des conditions de sélection,
- l'aiguillage des portes ouvertes vers les étapes de traitement,
- la gestion des temporisations,
- la gestion des portes et des groupes de portes,
- la communication au travers des portes et des groupes de portes,
- la création et la destruction d'acteurs (un cas particulier d'acteur est l'équivalent d'un processus Unix),
- l'accès à un système de gestion de fichiers répartis,
- la création de groupes répartis de processus et l'envoi de signaux aux processus et aux groupes de processus.

L'interface interactive de l'utilisateur

L'interface interactive de l'utilisateur de Chorus est semblable à celle d'un système Unix étendu. L'utilisateur dispose d'un interpréteur de commandes - *shell* - étendu qui permet:

- de contrôler les sites d'exécution des acteurs lancés par l'usager,
- d'accéder à tous les fichiers locaux ou distants,
- de manipuler des messages,
- d'accéder à tous les services Chorus.

De plus, l'utilisateur dispose d'un environnement suffisant pour développer ses applications, constitué d'utilitaires Unix transportés sur Chorus.

2.2. Structure interne du système

Le système est composé d'un noyau élémentaire et d'acteurs système.

Le noyau gère les sélections, les aiguillages et les temporisations; il assure la transmission locale des messages et l'enchainement des étapes de traitement; il est au coeur de la protection. Enfin, le noyau transforme les interruptions et les E/S de la machine en une interface de type porte/message: un acteur lié à un niveau d'interruption reçoit chaque interruption sous forme d'un message sur la porte qu'il a désigné; de même, chaque périphérique physique est représenté par une porte du noyau.

De manière générale, l'interface du noyau se présente comme un ensemble de portes et toutes les interactions entre les acteurs et le noyau se font par échange de messages.

Ce sont les acteurs système (construits sur le même modèle que tous les autres acteurs) qui réalisent toutes les autres fonctions de Chorus (gestion des acteurs, des

portes, gestion des communications distantes, gestion des périphériques, etc.). Les services réalisés par ces acteurs sont donc demandés par des messages envoyés sur leurs portes.

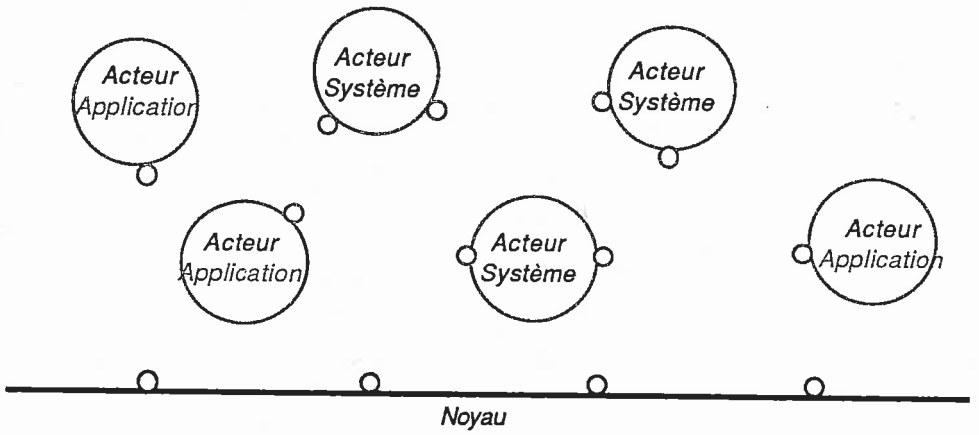


fig. 3.6 - Le noyau, les acteurs système et les acteurs application

Cette structure interne, ainsi que les protocoles d'accès aux services, sont cachés aux acteurs application par le biais de la bibliothèque d'interface du système. Cette bibliothèque invoque les services système en exécutant des protocoles du type demande/réponse. L'interface vue par les clients est donc une interface procédurale.

Le noyau et les serveurs système d'un site coopèrent étroitement entre eux d'une part et avec leurs partenaires des autres sites d'autre part. Cependant, leurs clients ne sont pas concernés par cette coopération, ni par la localisation des serveurs capables de rendre un service particulier.

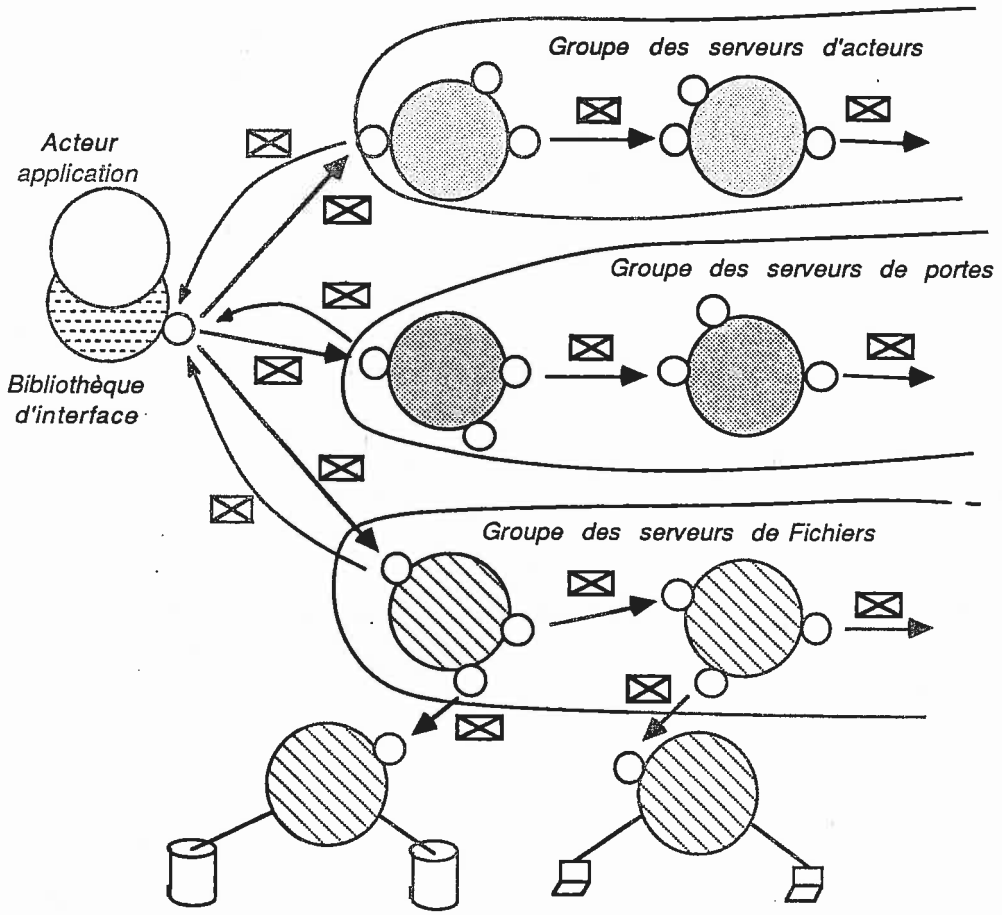


fig. 3.7 - Structure interne du système

2.3. L'implantation sur la SM90

Chorus est implanté sur un ensemble de SM90 reliées par un réseau Ethernet. La SM90 [Finger 81] est un multi-processeur avec une structure modulaire. Cet ordinateur est organisé autour d'un *bus* de communication global sur lequel peuvent être branchés plusieurs processeurs banalisés (modules de traitement - MT), plusieurs processeurs spécialisés (pour les entrées/sorties) et de la mémoire globale. Chaque MT peut aussi avoir sa mémoire locale et son propre *bus* local.

Chorus est implanté sur cette structure selon sa nature modulaire. Bien que le partage de ressources soit permis par la machine, il s'y exécute comme sur un réseau: chaque MT correspond à un site Chorus. La communication intra sites de la même SM90 est basée sur l'échange de messages, optimisé par l'usage de la mémoire commune.

Dans la SM90 chaque MT possède un noyau, un acteur serveur de portes et de groupes de portes et un acteur serveur d'acteurs. Un acteur serveur de transport réalisant le transport entre machines, un acteur serveur de fichiers et des acteurs contrôleurs de périphériques sont partagés par tous les sites de la même machine.

Pour terminer cette présentation du système Chorus nous décrivons ses options pour la protection.

3. Principes de la protection dans Chorus

Dans sa réalisation actuelle, la protection dans Chorus est bâtie sur une hypothèse préliminaire: toutes les machines liées par le réseau sont des machines controlées par Chorus (car le transport se fait sans cryptage ni authentification mutuelle des machines); spécifiquement, tous les messages émis par un acteur ont une entête construite par le système du site de l'émetteur, à laquelle le système du site du récepteur peut faire confiance.

Après cette remarque préliminaire, nous présentons les principes de la protection dans Chorus. Ces principes sont, dans certains de leurs traits, inspirés de ceux du système Unix†. Ils sont, grosso modo, les suivants:

- Un système Chorus (i. e., un ensemble de machines interconnetées par Chorus) est un domaine, au sens de la protection, dont l'administration est cohérente. Des administrateurs "humains" assurent que tout utilisateur et groupe d'utilisateurs possède une identification unique au domaine.
- L'identification interne (au sens de la protection) des usagers et des entités est représentée par une paire (identificateur d'usager, identificateur de groupe) noté aussi *(uid, gid)*, identificateur de protection ou simplement id-protection (IDP).
- La protection distingue deux classes d'entités: les entités actives et celles sur lesquelles elles agissent. Une entité active est un acteur; elle peut agir sur d'autres acteurs, sur les portes, sur les groupes de portes ou d'acteurs et sur les fichiers. Le contrôle de la validité d'une requête se fait en vérifiant que l'identificateur de protection au nom duquel elle est émise ainsi que l'identificateur de protection et les règles de protection de l'entité quelle concerne, donnent au demandeur les droits nécessaires pour satisfaire sa demande.

La mise en oeuvre de ces principes élémentaires est faite de la façon suivante:

- Les identificateurs de protection des usagers et les groupes d'usagers sont définis par des fichiers uniques au domaine (en fait ils sont dupliqués dans le domaine, cf chapitre V).
- Chaque entité (acteur, porte, groupe de portes ou d'acteurs et fichier) reçoit à sa création un identificateur de protection et des règles de protection pour sa manipulation. Grossièrement, cet identificateur est l'IDP au nom duquel a été émise la requête de création de l'entité.
- Toute action sur une entité se traduit par un message envoyé au serveur qui gère l'entité; chaque message est estampillé avec l'IDP de la porte émettrice lorsqu'il est émis. Cet identificateur est l'IDP au nom duquel la requête est émise. Lorsqu'un acteur reçoit un message, il a accès à cet identificateur pour pouvoir appliquer, éventuellement, les règles de protection attachées à l'entité qu'il gère.

---

† Dans Unix les usagers sont identifiés de façon interne par un numéro, *l'uid* de l'usager, et chaque usager appartient obligatoirement à un groupe d'usagers, identifié aussi de façon interne par un numéro, *le gid* de l'usager; ces identificateurs sont définis dans des fichiers standard ("/etc/passwd" et "/etc/group"), cf annexe II.

- Un acteur peut avoir des portes ouvertes avec des identificateurs de protection différents. L'identificateur au nom duquel sont émises ses requêtes peut ainsi varier selon la porte qu'il utilise.

La protection dans Chorus généralise celle d'Unix dans le sens suivant: si un acteur a des portes ouvertes avec des identificateurs de protection différents, ses droits peuvent varier en fonction de la porte qu'il utilise pour émettre ses requêtes, comme l'illustre la fig. 3.8.

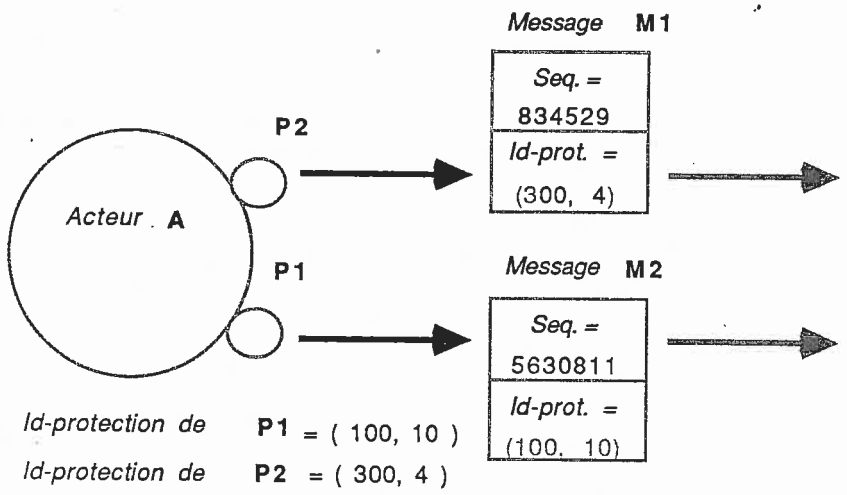


fig. 3.8 - Identificateurs de protection des portes et estampillage des messages

CHAPITRE IV

GESTION ET DESIGNATION DES PORTES ET DES GROUPES DANS CHORUS

CHAPITRE IV

GESTION ET DESIGNATION

DES PORTES ET DES GROUPES DANS CHORUS

Le but initial du projet Chorus était de définir une architecture répartie d'exécution et de communication, simple, générale et fournissant l'ensemble minimal de concepts nécessaires pour l'implantation d'applications réparties. Ce but a été atteint avec succès comme l'atteste la Version 1 du système. Chorus est une architecture simple et générale mais, telle qu'elle est offerte en V1, d'une facilité d'emploi peu souple. Elle s'apparente beaucoup plus à une structure d'exécution d'applications réparties embarquées, développées selon une méthode croisée, qu'à un système d'exploitation réparti (SER) complet.

Dans cette version, quelques choix ont été retenus en matière de désignation et d'édition de liens [Senay 83]: les portes étaient désignées par des UID (*Unique Identifiers*) directement visibles des clients; les opérations sur les portes et l'adressage de messages n'étaient pas pour l'essentiel protégées; il n'y avait aucun moyen de désignation symbolique des portes; les moyens disponibles pour réaliser l'édition de liens étaient élémentaires. C'était aux couches au-dessus de l'architecture de base (niveaux bibliothèque d'interface ou outils langage) de combler ces insuffisances en s'adaptant au niveau des services requis par une utilisation particulière.

Pour faire évoluer cette architecture vers un système complet, plusieurs choix étaient possibles: développer un système réparti temps réel, développer un système à usage général du style "Unix réparti", développer un système "orienté objets" (du style d'Eden [Almes 85] par exemple), etc. Le choix qui a été retenu a été celui de construire un système réparti complet qui enrichit l'architecture Chorus, tout en conservant sa généralité, et capable de récupérer les acquis du système Unix du point de vue du logiciel disponible (applications, langage de commandes et outils de développement). Le résultat de ce choix est le système Chorus Version 2.

Ce chapitre commence par préciser les choix fondamentaux de Chorus V2 pour la gestion et désignation des portes, des groupes de portes et des groupes d'acteurs, ainsi que pour l'édition de liens entre acteurs. Ensuite, les paragraphes 2 et 3 présentent respectivement la gestion et désignation des portes et des groupes de portes. Le paragraphe 4 introduit l'IPC (*Inter Process Communication facility*) de Chorus. Le paragraphe 5, les groupes d'acteurs. Le paragraphe 6 détaille l'héritage des portes et des groupes entre acteurs père et fils. Nous terminons par une présentation de l'implantation, paragraphe 7, et des leçons de notre travail, paragraphe 8.

1. Choix fondamentaux de Chorus

Le système Chorus†, comme plusieurs autres SER (Accent, V-System, ...), est un système orienté communication par messages. La plupart des appels système, même s'ils sont réalisés localement, sont obtenus par l'échange de messages au travers de portes de communication. Chorus offre aussi à ses clients un IPC puissant et performant.

En effet, le système et les applications peuvent être assimilés à un ensemble de gestionnaires de ressources - ou d'objets - dont l'interface est clairement définie par un ensemble de portes de communication - les points d'accès aux objets - et par l'ensemble des opérations accessibles derrière ces portes. Typiquement, les clients invoquent les opérations sur les objets par des protocoles "demande/réponse".

Cette façon de concevoir la coopération des différents composants d'une application ou d'un service réparti a été particulièrement enrichie par l'intégration, dans un ensemble homogène, des fonctionnalités suivantes:

- Les portes peuvent être successivement associées à différents acteurs pour permettre la reconfiguration dynamique des applications et la répartition de la charge. Le *droit de réception* des messages reçus sur une porte peut ainsi transiter dynamiquement d'un acteur à un autre, même s'ils sont sur des sites différents. Si c'est le cas, la porte migre. Le système se charge dynamiquement de la localisation des portes.
- Chorus offre le concept de groupe de portes pour permettre la diffusion de messages et faciliter l'adressage générique et fonctionnel (cf chapitre II) d'un service ayant de multiples points d'accès.
- Pour permettre la réalisation d'opérations portant sur un ensemble d'acteurs, le système fournit aussi la notion de groupe d'acteurs (implanté à l'aide de la notion de groupe de portes).
- Des fonctionnalités d'édition de liens au chargement et à la demande sont aussi disponibles. Les portes (et les groupes de portes) peuvent être associées avec des noms symboliques. Le père d'un acteur peut contrôler l'environnement de communication initial du fils et spécifier les portes qu'il aura automatiquement à sa naissance.
- Toutes les opérations sur les portes et les groupes sont protégées.

Au niveau de l'architecture interne du système les entités sont désignées par des UID. Ce niveau de désignation n'est accessible qu'au noyau et aux acteurs système. Les avantages de l'utilisantion des UID, mises en évidence au chapitre I, ont guidé ce choix.

La visibilité directe des UID par les clients présente plusieurs insuffisances (en ce qui concerne la protection, la portabilité, la détection d'exceptions et l'édition de liens, cf §4 chapitre I). Afin de combler ces insuffisances, à un coût supplémentaire minimal, une autre couche de désignation interne a été introduite. Cette nouvelle couche est matérialisée par le biais du contexte d'un acteur. Elle cache les UID aux clients du système sous des noms contextuels ou locaux. Ces derniers se présentent comme des entiers, variant de 0 à N, et calculés dynamiquement, qui ne sont valides que dans le contexte de l'acteur qui les connaît. Nous les appellons descripteurs de portes/groupes ou PGD (pour *Port/Group Descriptors*).

Cette nouvelle couche de désignation interne, la seule accessible aux clients, présente plusieurs avantages:

† A partir de ce paragraphe Chorus est synonyme de Chorus V2.

- Elle rend naturelle l'introduction de méthodes d'édition de liens au chargement; elle permet ainsi au père d'un acteur de spécifier le contexte de communication initial du fils.
- Elle permet le contrôle de la visibilité des noms; en effet, pour qu'un acteur puisse appliquer une opération à une entité ainsi désignée, il faut qu'il connaisse son nom. Elle joue donc un rôle important vis-à-vis de la protection.

Finalement, une porte (ou un groupe de portes) peut également être associée avec un nom symbolique. Cette autre couche de désignation est introduite par un système de désignation symbolique (SDS), cf chapitre I, construit comme une extension du SDS des fichiers: une porte (ou un groupe de portes) est désignée par un nouveau type de noeud de l'arbre des fichiers. Cette fonctionnalité sert à l'édition de liens dynamique et joue un rôle important dans la construction du système de gestion de fichiers réparti de Chorus (cf chapitre V).

Par la suite nous allons introduire les détails de ces fonctionnalités et discuter les raisons qui ont présidé à nos choix. Nous commencerons par présenter les portes de Chorus.

2. Caractérisation, gestion et désignations des portes dans Chorus

Dans un système basé sur la communication par messages, ceux-ci sont envoyés vers des "réceptacles à messages". La façon dont ces réceptacles sont associés aux processus introduit différentes méthodes d'adressage de messages. Parmi celles-ci, les plus répandues sont l'adressage direct ou de processus à processus (exemple: V-Kernel) et l'adressage au travers de portes (exemple: Accent).

Dans Chorus la communication se fait au travers de portes. Un acteur peut avoir plusieurs portes. Une porte désigne un point d'accès à un *service,* ou à une *opération,* et non nécessairement un *serveur;* en particulier, dans Chorus, les portes existent indépendamment des acteurs et la même porte peut être, successivement et dynamiquement, associée à différents acteurs. Bien que plus lourde du point de vue du système que l'adressage direct, cette méthode est plus souple du point de vue du client parce que, d'une part, elle permet la reconfiguration dynamique (une porte peut être dynamiquement associée à différents acteurs) et, d'autre part, elle facilite la synchronisation (comme un acteur peut consommer des messages au travers de plusieurs portes, il n'est pas obligé de consommer un message pour se rendre compte qu'il doit reporter à plus tard son traitement).

2.1. Caractérisation des portes dans Chorus

Logiquement, dans Chorus, une porte est un réceptacle à messages, i. e., une file d'attente de messages. Cette entité système peut être dans l'un des deux états suivants:

Fermée: la porte est inactive; la file d'attente est vide; on ne peut ni déposer, ni retirer des messages de cette porte; les messages émis vers une porte fermée sont perdus.

Ouverte: elle est active; elle peut recevoir des messages; un seul acteur, celui qui l'a ouverte, peut retirer des messages de la file d'attente.

Quand un acteur est chargé, le système lui associe un certain nombre de portes dans l'état "ouverte" (cf §6). En plus de ces portes un acteur peut créer dynamiquement d'autres portes. A la suite de sa création, une porte est fermée. Afin de pouvoir se servir d'une porte, un acteur doit l'ouvrir, et quand il n'a plus besoin d'elle, il doit la fermer. Néanmoins, pour permettre la reconfiguration, une porte peut être créée

par un acteur et successivement ouverte/fermée par d'autres.

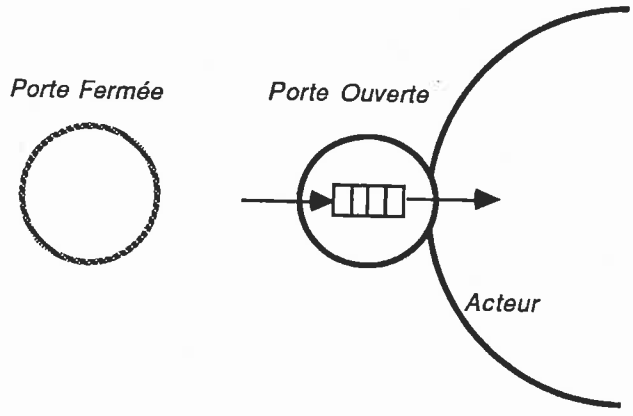


fig. 4.1 - Etats d'une porte

Pour que l'identification de l'émetteur et du récepteur soit symétrique, un acteur doit aussi émettre ses messages à partir d'une porte ouverte: les envois se font donc de porte à porte et non d'acteur à porte. L'environnement de communication de tout acteur (émetteur ou récepteur) se résume ainsi à un ensemble de portes.

2.2. Manipulation des portes

L'échange de messages à travers les portes permet la coopération entre acteurs. Il n'y a pas de relation établie a priori entre les portes. Un message peut être adressé de n'importe quelle porte émettrice vers n'importe quelle porte réceptrice. Ainsi, comme les portes sont le seul moyen d'interaction entre les acteurs, il est essentiel que les acteurs puissent acquérir dynamiquement le nom des portes de leurs partenaires. Pour un acteur, les noms des portes sont des noms qui lui sont contextuels, qu'il peut acquérir par héritage, dynamiquement ou parce qu'il est le créateur de la porte. Comme nous l'avons déjà dit, nous appelons ces noms des *pgd*.

Les opérations qui permettent la manipulation des portes sont les suivantes:

creatport → pgd

crée une porte dans l'état "fermée" et rend son nom. Pour pouvoir se servir de la porte, l'acteur doit l'ouvrir;

openport (pgd, priorité)

ouvre la porte désignée par *pgd;* la *priorité* d'une porte caractérise la priorité à accorder au traitement des messages qu'elle reçoit.

closeport (pgd)

ferme la porte désignée par *pgd;* seul l'acteur qui l'a ouverte peut la fermer;

dlport (pgd)

détruit la porte désignée par *pgd;* cette porte doit être préalablement fermée. Si l'appel réussit, le nom *pgd* est désalloué du contexte de l'appelant. Celui-ci peut être un acteur différent de celui qui a créé et/ou ouvert la porte.

Dans un environnement de développement de logiciel, comme l'est la présente version de Chorus, certains acteurs matérialisent des commandes interactives et leurs portes doivent donc être détruites lorsque leur exécution prend fin. Seules les portes qui donnent accès à des services permanents ou dynamiquement reconfigurables doivent être conservées par le système.

A cette fin, quand un acteur est détruit, toutes les portes qu'il avait ouvertes sont fermées et parmi ces portes, celles qui ont l'attribut *destruction automatique* positionné sont en plus détruites à ce moment là. Cet attribut d'une porte est positionné par son créateur comme nous verrons plus tard.

2.3. Transparence de la localisation et migration

Les noms des portes sont indépendants de la localisation des portes qu'ils désignent: un acteur ne peut pas déduire le site de résidence d'une porte à partir de son nom. De plus, pour un acteur, il n'y a pas de différence entre une communication locale ou distante. Quand une porte est fermée par un acteur, un autre acteur peut l'ouvrir: le *droit de réception* (cf §2.7) peut ainsi transiter d'un acteur à un autre, même s'ils sont sur des sites différents. Si c'est le cas, la porte migre mais cette migration est cachée à tous les acteurs qui la connaissent.

Quand une porte migre d'un site A vers un site B, son descripteur transite de A vers B et la file d'attente de messages qui lui est associée est recréée sur B. Pendant cette opération, la porte ne perd, ni n'acquiert aucun nouvel attribut. En particulier, elle continuera à appartenir au même ensemble de groupes de portes (cf §3). Le système se chargera dynamiquement de la localiser (cf §7.4).

Notre implantation actuelle de la migration des portes est réalisée de telle sorte que les messages reçus pendant la migration, ou non encore consommés avant la migration, sont perdus. Cependant, même en présence de cette limitation, il a été démontré que la migration des portes peut être utilisée pour construire des serveurs dupliqués qui se reconfigurent dynamiquement, en présence de pannes, de façon transparente à leurs clients [Banino 85].

2.4. Expérience d'utilisation des portes

Le système Chorus est bâti sur les concepts qu'il offre à ses clients. Sa réalisation nous a donc permis d'apprécier les qualités de ses concepts de base. Dans le système il y a plusieurs serveurs: le serveur gestionnaire des portes et des groupes de portes, le serveur gestionnaire d'acteurs, le serveur de désignation et de fichiers, plusieurs serveurs contrôleurs de périphériques, etc.

Ces serveurs ont des politiques différentes quant à l'utilisation des portes: une porte par opération, une porte par groupe de fonctionnalités de même priorité, une porte par groupe de serveurs avec lesquels un dialogue est établi, etc. Cependant, dans tous les cas, nous avons constaté que la possibilité d'avoir différents points d'accès au même serveur (i. e., plusieurs portes par serveur) s'avère être un moyen très souple de leur structuration.

D'autre part, l'association dynamique des portes aux acteurs et la possibilité que les portes ont de migrer sont aussi des fonctionnalités très intéressantes. Par exemple, dans Chorus, comme nous verrons dans les détails au §5, un acteur est identifié du point de

vue du système par une porte spéciale: la *porte des signaux*. Des signaux (fonctionnalité analogue à celle du système Unix) envoyés d'un acteur A vers un acteur B sont transformés en l'envoi de *messages de signal* vers la *porte des signaux* de B. Ces signaux peuvent, par exemple, solliciter la destruction de B. L'appel système *exec* du système Unix permet qu'un processus change de code et de données tout en conservant les ressources qu'il manipulait ainsi que son identification et interface. Cette fonctionnalité est aussi disponible sur Chorus, mais elle est étendue à la répartition: l'acteur peut migrer en exécutant un exec sur un site distant. Au cours de cette opération, les portes de l'acteur peuvent migrer de site et sont dynamiquement associées à un autre code et données. En particulier, sa *porte des signaux,* est dynamiquement associée à un nouveau contexte, dans un autre site, mais ses partenaires peuvent toujours lui envoyer des signaux en l'adressant.

Nous venons de voir comment sont caractérisées et manipulées les portes. Nous décrivons ci-dessous comment elles sont désignées.

2.5. Désignation des portes

Comme dans d'autres systèmes répartis (cf chapitre I) les portes (et les groupes de portes) sont désignées dans Chorus, de façon interne, par des UID. Ces noms ont la structure indiquée par la figure 4.2.

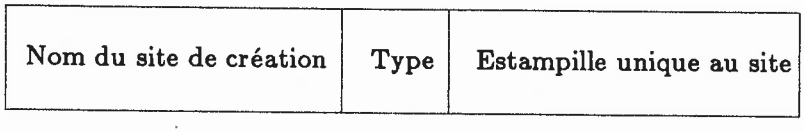


fig. 4.2 - Structure d'un UID Chorus

Les noms des sites sont des numéros uniques alloués statiquement. L'estampille est calculée à partir de l'heure (logique) de création de l'entité.

Dans les versions précédentes de Chorus, les UID des entités étaient directement visibles des clients. Cependant, comme Chorus a évolué d'une structure d'exécution d'applications réparties embarquées vers un système de développement et d'exécution d'applications réparties, les noms des portes (et des groupes) connus des clients sont, maintenant, des noms contextuels aux acteurs. Un acteur ne peut manipuler une entité que si le système a contrôlé l'acquisition de son nom par l'acteur. Chorus, tel qu'Accent [Rashid 81] ou DEMOS/MP [Powell 83] implante donc un espace d'adressage pour les messages qui est protégé. Les couches de désignation présentes dans le système sont celles de la figure 4.3.

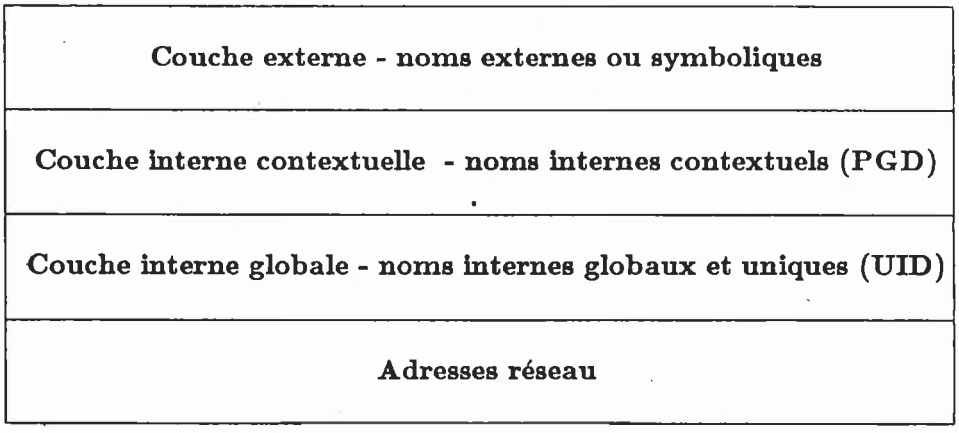


fig. 4.3 - Couches de désignation des portes et des groupes dans Chorus

Les adresses réseau et les UID ne sont visibles que des serveurs système. Les noms contextuels et les noms symboliques sont les seuls accessibles aux clients†.

Les noms contextuels, locaux à l'acteur, peuvent désigner des portes ouvertes par l'acteur, ouvertes par un autre acteur, ou fermées (ainsi que des groupes de portes); ils sont calculés dynamiquement par le système et ne sont valides que dans le contexte d'un acteur. En particulier, ils ne peuvent pas être passés d'un acteur à l'autre sans être traduits entre contextes††.

Dans des systèmes où les messages sont typés, comme Accent par exemple, ce typage est utilisé pour réaliser le passage de noms entre processus: le système peut reconnaître la présence dans un message d'un nom contextuel et faire une double traduction, **nom contextuel → nom global,** à l'émission, et **nom global → nom contextuel,** à la réception. Dans Chorus, comme les messages ne sont pas typés, nous offrons d'autres mécanismes pour réaliser l'édition de liens entre les acteurs. Nous décrivons ces mécanismes ci-dessous.

Identification automatique de l'émetteur

Quand un acteur reçoit un message, il peut acquérir un nom qui lui est contextuel et qui désigne la porte émettrice du message par le biais de l'opération:

msgsrc (message) → pgd

car le système assure la transmission automatique du nom de l'émetteur (en fait c'est son UID qui est transmis) dans l'entête du message (qui est une zone protégée du message).

---

† Les UID ont l'avantage de pouvoir être passés d'un contexte à l'autre sans traduction. Pour cette raison, nous avons laissé aux acteurs système la visibilité de ces noms. Nous leurs épargnons ainsi la relative lourdeur de la traduction de noms entre contextes.

†† Fondamentalement, ces noms sont des entrées dans une table du contexte de l'acteur - le contexte d'adressage pour la communication - qui traduit les noms contextuels (PGD) en des noms globaux (UID). Dans cette table, il n'y a qu'une entrée par UID, donc par entité connue de l'acteur.

Héritage de noms contextuels entre père et fils

Un des avantages les plus intéressants de la désignation contextuelle est la possibilité de l'introduction de schémas d'édition de liens au chargement des processus. Ainsi, dans Chorus, quand un acteur est créé, il hérite de son père les noms contextuels des portes et des groupes de portes qu'il connaît. Le système fournit aussi la possibilité au père de cacher au fils certains de ces noms. Cette fonctionnalité, conjuguée avec la possibilité offerte au père de créer des portes que ses fils ouvriront, peut être utilisée pour la configuration d'applications réparties (cf §6).

Finalement, le système permet aussi l'acquisition dynamique de noms de portes et groupes de portes au travers de la désignation symbolique.

Edition de liens dynamique par le biais des noms symboliques

Les serveurs de fichiers de Chorus permettent la désignation symbolique des portes (et des groupes) par le biais de noeuds spéciaux de l'arbre de désignation de fichiers que nous appelons *noeuds du type porte/groupe*. Ces noeuds permettent l'association d'une porte (ou d'un groupe de portes) avec un nom symbolique (chemin d'accès).

Par exemple, le serveur chargé de gérer une imprimante *laser* a une porte qui reçoit les demandes d'impression; cette porte (qu'il désigne contextuellement par le PGD *p1*) peut être désignée de façon symbolique par */printers/laser*. Quand le serveur est chargé, il associe sa porte à son nom symbolique (en réalité c'est l'UID de sa porte qui est associé au noeud) au travers de l'appel:

symbbind (SETNAME, "/printers/laser", p1)

un client de ses services peut obtenir un nom contextuel de cette porte en appelant:

symbbind (GETNAME, "/printers/laser") → pgd

et il pourra plus tard émettre des messages vers *pgd*.

L'association d'un nom contextuel (PGD) à un nom symbolique est indépendante de l'état de l'entité qu'il désigne. Par exemple, un acteur peut créer une porte et l'enregistrer; un autre acteur peut obtenir le nom de la porte et l'ouvrir. C'est aux clients de cette fonctionnalité d'assurer la cohérence entre les attributs du nom symbolique et l'état de l'entité ainsi désignée. Cependant, comme tous ces mécanismes de traduction de noms sont bâtis sur l'utilisation des UID, le système fonctionne toujours de façon non ambigue et les clients peuvent détecter l'incohérence a posteriori.

Associés aux *noeuds de type porte/groupe* il y a un ensemble de droits (tel qu'aux autres noeuds de l'arbre Unix). Fondamentalement, si un acteur a le *droit écriture* sur un *noeud de type porte/groupe,* il possède le *droit d'associer* une porte (ou un groupe) à ce noeud; le *droit d'acquérir* un nom contextuel de l'entité désignée symboliquement par un noeud, n'est accordé qu'aux clients qui ont le *droit lecture* sur ce noeud de l'arbre.

Finalement, nous détaillons ci-dessous un aspect complémentaire qui concerne la désignation par le biais des noms contextuels.

2.6. Nature des noms contextuels dans Chorus

Dans Chorus, un nom contextuel (PGD) est une référence contextuelle vers une entité (ce nom cache l'UID de l'entité). Si ce nom référence une porte ouverte par l'acteur, il peut être sûr que la porte existe; sinon le système garantit seulement que cette entité a existé auparavant. Si un acteur ferme et/ou détruit une porte (ou détruit un groupe de portes), il n'y a pas de notification automatique des autres acteurs qui la

connaissent. Ils ne le détecteront qu'a posteriori. Par exemple, si un acteur demande l'ouverture d'une porte qui n'existe plus, le système lui rend une erreur.

Cette caractéristique de la gestion de l'espace d'adressage contextuel est due au fait que Chorus ne traite pas l'excéption "avoir accès à un objet qui n'existe plus" ni dispose d'aucun "ramasse-miettes" qui libère les noms contextuels qui ne sont plus valides. Des fonctionnalités de ce style seraient trop lourdes. Néanmoins, la désignation par le biais des UID diminue l'impact de leur absence.

Les acteurs disposent d'un appel système spécial, prévu pour la libération (par demande explicite de l'appelant) de noms contextuels. Il s'agit de l'appel **relpgd**, voir l'annexe III. L'appel **pgdtype**, cf annexe III, retourne le type de l'entité référencée par un nom contextuel.

2.7. Les protections associées aux portes

La protection dans Chorus est bâtie sur le modèle suivant (cf chapitre III): chaque acteur s'exécute au nom d'un usager (un ***principal*** au sens de la protection); chaque entité passive (fichier, porte et groupe de portes) appartient aussi à un usager, celui qui l'a créée; chaque entité passive reçoit à sa création un ensemble d'attributs de protection spécifiant qui peut la manipuler.

En particulier, chaque porte reçoit à sa création un ensemble d'attributs de protection spécifiant qui peut l'ouvrir et qui peut la détruire. Les noms contextuels des portes et des groupes ne sont donc pas des capacités. En effet, leur connaissance est une condition nécessaire, mais pas suffisante, pour pouvoir manipuler l'entité qu'ils désignent. Comme nous montrerons par la suite, ces noms ne partagent les propriétés d'une capacité que du point de vue du droit d'émission de messages.

En effet, la protection des opérations associées aux portes se fait par le biais de plusieurs mécanismes; d'une part il y a les contrôles intrinsèques à la méthode de désignation interne utilisée:

- Le *droit de créer* une porte est public. Le créateur acquiert toujours un nom contextuel qui désigne la porte.
- Le *droit d'émission* de messages vers une porte P, n'est accordé que si l'acteur connaît un nom contextuel de P. Ce privilège peut donc être acquis dynamiquement ou par héritage. La connaissance d'un PGD désignant P, est aussi une condition nécessaire, mais non suffisante, pour pouvoir ouvrir ou détruire P, voir ci-dessous.
- Seul l'acteur qui a une porte ouverte peut la fermer.

D'autre part, il y a les contrôles associés aux identificateurs de protection (IDP), cf §3 chapitre III. Ces contrôles sont tous exécutés selon le principe suivant: une porte P reçoit à sa création l'identificateur de protection au nom duquel elle a été créée - IDP1; lorsqu'un acteur se présentant au système avec l'IDP2, désire ouvrir ou détruire P, les contrôles de protection sont fonction de IDP1, IDP2 et des règles ou attributs de protection de P (cf ci-dessous).

Les attributs de protection et *l'attribut destruction automatique* sont implicitement donnés par le créateur au travers d'un "masque", conservé dans son contexte, voir l'appel **pmask** dans l'annexe III. Ces attributs, ainsi que l'identificateur de protection de la porte sont statiques.

Les *droits d'émission* et de *réception de messages* sont donc dissymétriques: le contrôle du *droit d'émission* vers une porte (ou un groupe de portes) est reporté au contrôle de la visibilité d'un nom contextuel qui les désigne - ce droit peut être acquis

dynamiquement ou au chargement; par contre, le contrôle du *droit de réception* des messages reçus par une porte P est reporté au contrôle du *droit d'ouverture* de P, un droit spécifié statiquement.

Les attributs de protection des portes sont de deux types: *droit d'ouverture* et *droit de destruction*. Comme pour les fichiers d'Unix, ils sont structurés en trois ensembles logiques: les droits de l'usager qui a créé la porte, les droits des usagers du même groupe d'usagers et les droits des autres usagers. Un acteur ne peut ouvrir (respectivement détruire) une porte que si l'identificateur de protection avec lequel il se présente au système lui donne le *droit d'ouverture* (respectivement de *destruction*).

Un dernier aspect sur la protection est le suivant: parmi les portes ouvertes par un acteur, l'une d'elles constitue sa *porte des appels système*, voir **syscallport** dans l'annexe III. Cette porte a deux rôles:

- Elle sert à la bibliothèque d'interface du système pour réaliser les échanges de messages nécessaires pour obtenir les services système.
- L'identificateur de protection de cette porte est *l'IDP effectif* de l'acteur dans la mesure où il se présente au système avec cet identificateur; en effet, les messages qu'il émet par la *porte des appels système* sont estampillés avec cet IDP, voir §3 chapitre III.

2.8. Résumé, comparaison et conclusions

Les portes Chorus sont des entités système qui peuvent être créées et détruites dynamiquement. Leur association aux acteurs est dynamique et elles peuvent aussi migrer.

Les portes sont désignées par plusieures couches de désignation, ayant chacune un rôle spécifique. Au niveau interne du système, les portes sont désignées par des UID qui cachent leur localisation et permettent leur désignation de façon non ambigue. Les acteurs les désignent par des noms contextuels, calculés dynamiquement en fonction des besoins, les PGD. Cet espace de désignation contextuel est protégé et permet l'héritage de noms de père en fils. Finalement, les portes peuvent aussi être associées à des noms symboliques. Ces noms s'adressent à l'édition de liens à la demande et à la structuration de la vision externe offerte par le système.

En ce qui concerne la désignation, protection et la manipulation d'entités système, servant à la communication par messages entre processus, deux approches opposées sont possibles.

L'approche du V-System

Dans ce système la communication se fait directement de processus à processus et les processus sont désignés de façon interne par des UID directement visibles des clients. Ce système reporte aux clients les problèmes suivants:

1/ la protection de l'espace d'adressage de messages;

2/ la transmission de noms en général, et leur héritage en particulier;

3/ le contrôle de la cohérence entre la vision qu'un processus a de son environnement et l'état exact de cet environnement.

Les clients peuvent s'échanger librement des noms internes dans des messages. Le système résultant est assez simple (telles étaient aussi, grossièrement, les options de Chorus V1 pour la désignation et la protection des portes en ce qui concerne les points 1/, 2/ et 3/).

L'approche d'Accent

Dans ce système la communication se fait de porte à porte. Un processus peut avoir plusieurs portes et les portes peuvent transiter dynamiquement d'un processus à l'autre. Les portes sont désignées de façon interne par des noms contextuels. Ces noms sont une sorte de capacité (logiciel) pour leur adressage et manipulation, cf §4 chapitre I. Ce système offre les mécanismes suivants:

1/ la protection de l'espace d'adressage de messages et de l'accès à toutes les ressources en général;

2/ des messages typés permettant la transmission de capacités sur les portes et des opérations de création de processus permettant aussi leur héritage;

3/ l'envoi aux processus, au travers de messages d'urgence, des capacités que les partenaires ont relâchées (ou perdu suite à une panne) afin de permettre le traitement d'exceptions.

Chorus choisit une approche intermédiaire

Ainsi, dans Chorus, la communication se fait aussi de porte à porte. Un acteur peut avoir plusieurs portes et les portes peuvent aussi transister dynamiquement d'un acteur à l'autre. Mais, dans Chorus:

1/ les noms contextuels des portes ne sont pas des capacités, même s'ils partagent avec celles-ci leurs propriétés du point de vue du droit d'émission de messages;

2/ des mécanismes spécifiques sont prévus pour l'acquisition dynamique et pour l'héritage de noms;

3/ Chorus ne signale pas aux acteurs que le nom contextuel d'une entité qui vient d'être détruite, ou qui devient soudainement inaccessible, n'est plus valide.

Mises à part les actions sousjacentes à l'héritage, le coût supplémentaire que représentent les options de Chorus par rapport à la visibilité directe des UID, se résume au prix des traductions des noms contextuels en UID, et plus rarement, au prix de la traduction inverse (en effet, celle-ci n'est faite qu'à la demande).

Accent est un système qui s'est revélé une élégante architecture dont la lourdeur est, néanmoins, non négligeable. Par exemple, le contrôle des capacités et leur transmission par le réseau est mis en oeuvre par le mécanisme suivant: pour chaque porte P1 d'un site S1, ayant la capacité en émission sur une porte P2 d'un site S2, une porte P1' existe dans S1 qui représente P2; et pour que P2 puisse répondre à P1, une porte P2' doit exister dans le site S2 qui représente P1, cf [Rashid 81]. P1' et P2' sont des portes des serveurs de transport (*network-servers*). P1'/P2 et P2'/P1 sont implicitement des "liaisons réseau" entre portes. Ainsi, chaque fois qu'une capacité sur une porte transite dans le réseau, il y a la création/destruction de ces liaisons par les serveurs de transport. Ce sont ces serveurs qui supportent tout le poids de l'extension au réseau de l'adressage par capacités et de la transmission des capacités dans des messages (qu'ils reconnaissent automatiquement parce que les messages sont typés).

Comme un processus Accent ne peut connaître une porte distante que par le biais d'une porte locale qui lui est "liée", si la porte distante disparaît, les serveurs de transport font aussi disparaître les portes des autres sites correspondant aux liaisons où elle figurait; en l'occurence, chaque noyau transmet l'exception aux processus de son site.

Dans Chorus, l'implantation du transfert des noms entre sites est trivial car le noyau et les serveurs système ont la visibilité directe des UID. Fondamentalement, c'est la lourdeur des options d'Accent qui justifie celles de Chorus.

Accent n'offre aucune notion de groupe de portes ou de processus, au contraire du V-System ou de Chorus. Ces notions sont présentes dans Chorus et sont introduites à partir du prochain paragraphe.

3. Caractérisation, gestion et désignation des groupes de portes dans Chorus

La notion de groupe est une notion qui se révèle souvent nécessaire dans un système. Par exemple, cette notion permet la destruction de l'ensemble des processus d'un *pipe line* en une seule opération (*job control*). La répartition met en valeur la notion de groupe pour deux autres raisons: la diffusion de messages, ou adressage de 1 à N, qui remplace partiellement l'absence de mémoire commune dans les SER, et la possibilité d'encapsuler la notion logique de service réparti avec de multiples points d'accès équivalents, cf chapitre II.

Chorus offre deux notions de groupe, la notion de groupe de portes et la notion de groupe d'acteurs. Les groupes de portes sont introduits dans ce paragraphe, les groupes d'acteurs au §5.

3.1. Caractérisation des groupes de portes dans Chorus

Un groupe de portes est une liaison logique sur un ensemble arbitraire de portes: il peut être vide ou il peut contenir des portes ouvertes ou fermées. Les portes d'un groupe peuvent être réparties sur plusieurs sites. Une porte peut appartenir à zero, un ou plusieurs groupes, mais un groupe ne peut pas appartenir à un autre groupe (afin d'éviter la récursivité). Comme pour les portes, les acteurs ne connaissent que des noms contextuels des groupes; ils peuvent acquérir ces noms par héritage, dynamiquement, ou parce qu'ils sont les créateurs du groupe.

Cette liaison ne prend effet que du point de vue de la communication car la vie d'une porte n'est pas liée à celle du groupe (ou des groupes) auquel elle appartient. Le rôle des groupes de portes est de permettre des mécanismes de communication et d'adressage plus puissants que le point à point déterministe offert par les portes.

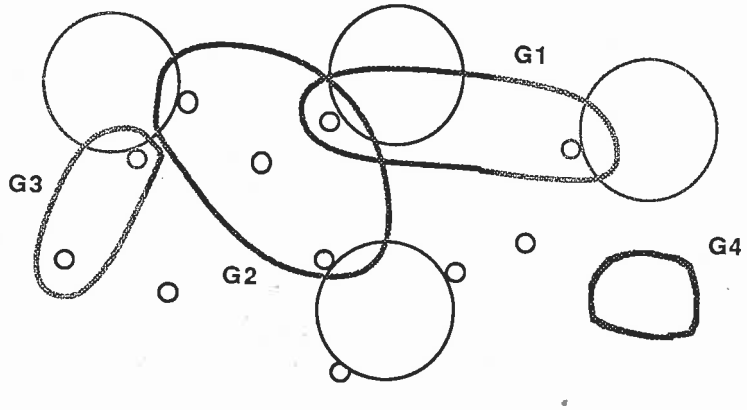


fig. 4.4 - Portes et groupes de portes

3.2. Manipulation des groupes de portes

Un groupe de portes peut être créé, modifié et détruit. L'opération:

creatgroup → pgd

crée un groupe vide de portes et rend son nom; à sa création le groupe reçoit aussi un attribut *destruction automatique* comme les portes. Si un groupe n'a pas l'attribut *destruction automatique* positionné, il ne peut être détruit que par l'opération:

dlgroup (pgd)

cependant, si le groupe est créé avec cet attribut, il est automatiquement détruit quand il redevient vide (et c'est alors la seule façon de le détruire).

Le choix de la forme que prend la destruction d'un groupe G est donc laissé à son créateur. En effet, si G regroupe un ensemble de portes d'une application répartie, il est naturel qu'il soit implicitement détruit quand l'application se termine; mais si G représente un service réparti S, il doit continuer à exister indépendamment du fait qu'il existe ou non un représentant de S couramment disponible. Ici nos options sont différentes de celles prises par le V-System [Cheriton 85]; dans ce système, les groupes qui deviennent vides sont toujours implicitement détruits.

Les opérations:

groupctl (PORTISIN, pgd-porte, pgd-groupe) → { oui, non }

groupctl (PORTINSERT, pgd-porte, pgd-groupe)

groupctl (PORTREMOVE, pgd-porte, pgd-groupe)

ont respectivement pour effet de tester l'appartenance, insérer ou retirer la porte *pgd-porte* du groupe *pgd-groupe*. Une porte peut être insérée ou retirée d'un groupe indépendamment de son état (fermée ou ouverte) et les changements d'état d'une porte (fermeture, ouverture ou migration) n'affectent pas sa qualité de membre d'un groupe de portes. Naturellement, une porte nouvellement créée n'appartient à aucun groupe, et quand elle est détruite, elle est retirée de tous les groupes auxquels elle appartenait.

3.3. Désignation des groupes de portes

Les groupes de portes sont désignés de la même manière que les portes. Par des UID à l'intérieur du système et par des noms contextuels (des PGD banalisés avec ceux des portes) au niveau des acteurs. Les noms contextuels des groupe de portes peuvent être aussi hérités ou associés avec des noms symboliques au travers de l'opération **symbbind.**

3.4. Attributs de protection d'un groupe de portes

Comme pour les portes, la protection des groupes est réalisée par plusieurs mécanismes; d'une part il y a les contrôles intrinsèques à la méthode de désignation par des noms contextuels.

- Le droit de créer un groupe est public. Le créateur acquiert toujours un nom contextuel qui désigne le groupe.

- Le contrôle du *droit d'émission* sur un groupe est reporté au contôle de la visibilité d'un nom contextuel qui le désigne. Ce droit est acquis, par un acteur, dynamiquement ou par héritage. Il est aussi condition nécessaire, mais non suffisante, pour pouvoir détruire ou modifier le groupe, voir ci-dessous.

D'autre part, Chaque groupe de portes reçoit aussi à sa création un ensemble d'attributs de protection spécifiant qui a le *droit de modification* et le *droit d'auto-modification* sur le groupe. Ces attributs de protection sont statiques et, ainsi que *l'attribut de destruction automatique* du groupe, donnés implicitement par le créateur au travers d'un masque conservé dans sont contexte (comme pour les portes), voir l'appel **gmask** à l'annexe III.

Le sens de ces attributs est le suivant: si un acteur a le *droit de modification* sur un groupe, il peut insérer ou retirer n'importe quelle porte du groupe et il peut aussi détruire le groupe (à condition qu'il connaisse un nom de la porte et du groupe comme pour toutes les autres opérations sur les portes et les groupes); si un acteur a le *droit auto-modification,* il ne peut qu'insérer ou retirer du groupe les portes qu'il a ouvertes.

Ces règles de protection ont été inspirées de celles des listes de distribution de courrier éléctronique: si un acteur a le *droit modification* sur un groupe, il est le maître du groupe, s'il n'a que le *droit auto-modification,* il ne peut que s'abonner ou se retirer du groupe. Par exemple:

- Dans un SER, on peut introduire un sous-système qui surveille les sites actifs et diffuse périodiquement à ses clients la liste de ces sites: cette diffusion utilise un groupe pour lequel le *droit auto-modification* est public, autrement dit, tout le monde peut s'abonner au service.
- Chaque Serveur de Fichiers a une porte inscrite dans le groupe du service Gestion de Fichiers : Gf. Un acteur voulant s'adresser à tous les serveurs de fichiers peut s'adresser à Gf. Ce groupe est publiquement connu mais seuls les serveurs de fichiers ont le *droit auto-modification* sur Gf.
- Afin de soumettre un groupe G à des règles de protection spéciales, il est possible d'en assurer la gestion par un serveur privé: chaque acteur qui a besoin d'introduire ou retirer une de ses portes de G doit envoyer un message au serveur. Le serveur a le *droit de modification* sur G, mais aucun autre acteur ne connaît le nom de G.

3.5. Groupes prédéfinis

Dans certaines circonstances, un problème de synchronisation est associé à l'utilisation groupes. En général, l'ensemble des serveurs qui participent à un service réparti S ne sont pas tous issus du même père; supposons que, quand un de ces serveurs s'initialise, il doit inscrire une de ses portes dans un groupe Gs, associé à S. A l'initialisation, si le serveur constate que Gs n'existe pas (en consultant un serveur de noms par exemple), il peut décider de le créer. Cependant, si deux serveurs s'initialisent en même temps, deux groupes distincts pourraient être créés.

En absence d'un service capable de réaliser atomiquement l'opération "obtention du nom d'un groupe, en le créant s'il n'existe pas" (qui pourrait être réalisé par un serveur de noms) le système doit offrir d'autres moyens pour résoudre ce problème†. L'opération:

† Pour résoudre ce problème, le V-System offre la notion de groupe statique: un groupe logiquement toujours existant et préalablement affecté à un service à la génération du système.

getpgroup (groupe-prédéfini) → pgd

retourne un PGD référençant un *groupe prédéfini de portes*. Un *groupe prédéfini de portes* est un groupe qui logiquement existe toujours. Il n'a à être ni créé ni détruit. Les opérations que l'on peut appliquer à un de ces groupes sont: **getpgroup**, pour obtenir un nom contextuel qui le référence, **groupctl**, pour insérer ou supprimer une porte du groupe, et les opérations d'envoi de messages comme pour un groupe explicitement créé par **creatgroup.**

Chorus offre (UidMax * PGroupMax)† groupes prédéfinis. A chaque identificateur interne d'usager ou *uid*, cf §3 chapitre III, est attribuée une tranche de PGroupMax groupes prédéfinis. Un acteur ne peut obtenir que le descripteur d'un groupe prédéfini de la tranche correspondante à l'*uid* avec lequel il se présente au système. Le paramètre *groupe-prédéfini* désigne un groupe prédéfini relativement à la tranche de groupes auxquels l'appelant a accès. L'accès au nom d'un groupe prédéfini par **getpgroup** est ainsi intrinsèquement protégé.

Les groupes de portes ont un rôle important dans l'implantation du système et sont une fonctionnalité très souple à la disposition des clients. Nous reviendrons sur notre expérience de son usage après la présentation des primitives de communication de Chorus.

4. La communication au travers des portes et des groupes

La communication par échange de messages au travers des portes et des groupes est le seul moyen d'interaction entre les acteurs. Il n'y a aucune interconnexion pré-établie entre les portes. Les messages sont échangés en mode non connecté.

Dans Chorus un message en transit est constitué de quatre éléments:

Pe, Pr, Ns, Données

où Pe désigne la porte émettrice, Pr la porte réceptrice et Ns le numéro de séquence du message, un numéro unique et global qui l'identifie pour toujours. Ns est accessible (en lecture) au récepteur du message. Celui-ci peut aussi obtenir des noms contextuels référençant Pe et Pr au travers d'appels système spéciaux (**getsrc** et **getdest**, cf annexe III).

Nous distinguons deux types fondamentaux d'échanges: l'échange asynchrone et l'échange demande/réponse, ainsi que deux points de vue: celui de l'émetteur et celui du récepteur.

4.1. Point de vue de l'émetteur : échange asynchrone

putmsg (message, source, destination, mode d'adressage)

envoie de façon asynchrone *message* de la porte *source* (qui doit être une porte ouverte de l'émetteur) vers la porte ou le groupe *destination*. Si *destination* désigne un groupe, le paramètre *mode d'adressage* est pris en considération; fondamentalement, il peut prendre les valeurs:

- *diffusion*: le message est envoyé à toutes les portes du groupe - en aucun cas la porte émettrice ne recevra une copie du message même si elle appartient au groupe.
- *fonctionnel*: le message est envoyé à une seule porte du groupe, n'importe laquelle -

† UidMax et PGroupMax sont des constantes de génération du système Chorus.

en aucun cas la porte émettrice n'est séléctionnée même si elle appartient au groupe.

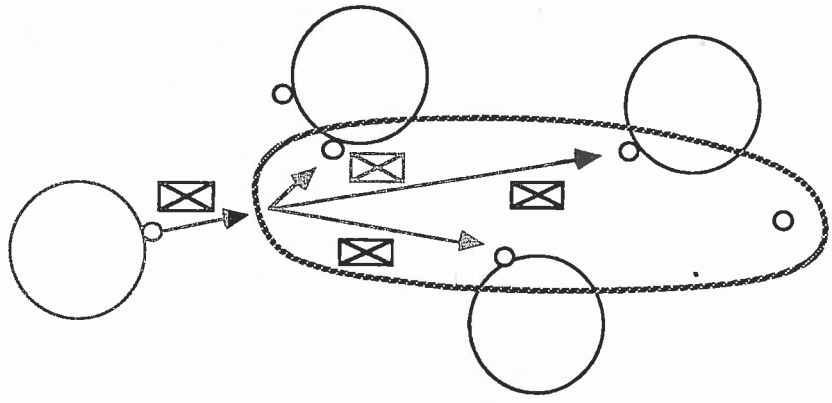


fig. 4.5 - Diffusion d'un message sur un groupe de portes

Si un message M = {Pe, Gr, Ns, Données} est adressé à un groupe (Gr), chaque porte réceptrice Pr qui reçoit le message M, reçoit le message {Pe, Pr, Ns, Données} i. e., le mode d'adressage et les groupes sont cachés aux récepteurs.

4.2. Point de vue de l'émetteur : protocole demande / réponse

call (message, source, destination, mode d'adressage, délai) → message

implante un protocole de demande/réponse. *Message* est envoyé de la porte *source* vers la porte ou le groupe *destination;* si *destination* désigne un groupe, le mode d'adressage doit être le mode fonctionnel de telle sorte que le message ne puisse être reçu que par une seule porte. Cet appel système retournera dans l'un des deux cas suivants:

- le message de réponse à l'appel - un message ayant le même numéro de séquence que le message émis (voir ci-dessous) - a été reçu par l'émetteur,
- ce message n'est pas revenu à l'émetteur mais *délai* est échu.

4.3. Point de vue du récepteur

Dans Chorus il n'y a pas d'appel pour faire la demande explicite de l'attente d'un message. Les acteurs sont structurés en étapes de traitement et, une fois une étape de traitement terminée, l'acteur se trouve toujours implicitement *en attente de réception* d'un message. Ainsi, les acteurs désirant recevoir des messages, autres que ceux de réponse à **call,** doivent avoir une structure adéquate, voir [Guillemont 84, 86] ou le chapitre III. L'état *en attente de réception* est bloquant; des chiens de garde sont disponibles pour éviter une attente infinie.

Le récepteur d'un message peut lui appliquer l'opération **putmsg** mais, s'il est engagé dans un protocole demande/réponse, il utilise souvent les deux opérations suivantes:

putfwd (message, destination)

qui retransmettra le message vers une autre porte, et

putreply (message)

qui le renverra à son émetteur initial. **Putreply** modifie les attributs porte émettrice et porte réceptrice et **Putfwd** l'attribut porte réceptrice du message mais elles ne modifient pas son attribut numéro de séquence.

4.4. Expérience d'utilisation de l'IPC Chorus

Le protocole demande/réponse (**call**) s'est avéré une fonctionnalité importante pour l'implantation du système. Il représente le point de vue du client dans un échange client/serveur. **Putreply** et **putfwd** représentent le point de vue du serveur dans le même échange. En effet, presque tous les appels système sont transformés par des procédures d'interface (*stub procedures*) en des échanges de demande/réponse avec les serveurs système ou le noyau.

L'envoi asynchrone est surtout utilisé par les clients qui veulent déclencher un évènement asynchrone, ainsi que par les serveurs qui se multiplexent: ils ne restent pas bloqués s'ils doivent coopérer avec d'autres serveurs pour accomplir le service demandé par un client. Pour d'autres détails sur le contrôle d'exécution dans Chorus voir [Guillemont 86] et [Rozier 86].

Nous présentons ci-dessous notre expérience de l'utilisation des groupes de portes pour la communication.

4.5. Expérience d'utilisation des groupes de portes

Chorus n'implante pas une diffusion fiable: quand un groupe est adressé, rien ne garantit que tous ses membres recevront une copie du message; il a été montré par d'autres, cf [Cheriton 85], qu'il est possible de construire des protocoles de diffusion fiable bâtis sur des groupes ne supportant pas la diffusion fiable.

Les serveurs de portes et groupes de portes (cf §7) et les serveurs de fichiers de Chorus, utilisent des groupes de portes et la diffusion pour localiser des entités nomades et pour dupliquer des catalogues (voir le chapitre V). Un autre aspect de l'utilisation de la diffusion par Chorus est illustré au §5, qui présente les groupes d'acteurs.

4.6. Interêt du mode d'adressage fonctionnel

L'interêt de cette fonctionnalité de Chorus est illustré ci-dessous par deux exemples.

Répartition de la charge à l'aide du mode d'adressage fonctionnel

Une boîte à lettres permet la répartition de la charge entre plusieurs serveurs: les serveurs peuvent tous lire, alternativement, la même boîte à lettres. Dans Chorus, cette utilisation des boîtes à lettres peut être réalisée par un groupe de portes que les clients adressent en mode fonctionnel. Chaque fois qu'un serveur ayant une porte dans le groupe est sélectionné et reçoit un message, il retire sa porte du groupe, réalise le service, répond au client et finalement, il réinsère à nouveau la porte dans le groupe.

Edition de liens dynamique à l'aide du mode d'adressage fonctionnel

Un client qui veut établir un dialogue suivi avec un serveur quelconque parmi un ensemble de serveurs équivalents (par exemple pour imprimer un fichier) commence par adresser le groupe associé au service en mode fonctionnel puis, par la suite, il n'adresse que la porte à partir de laquelle la réponse à son premier appel lui a été renvoyée.

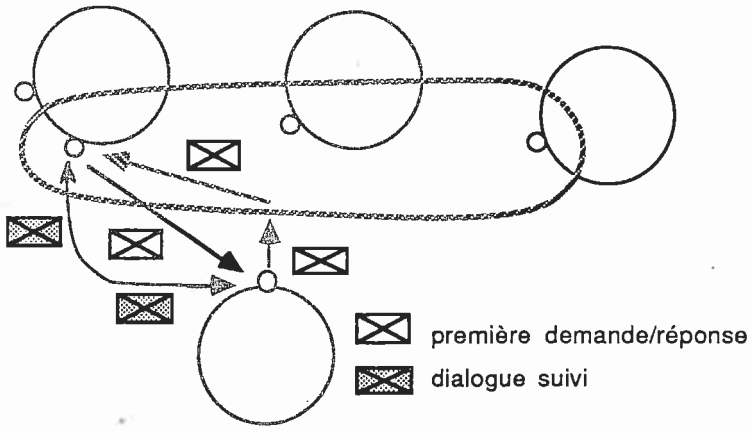


fig. 4.6 - Etablissement d'un dialogue suivi

4.7. Limitations actuelles des groupes de portes

Un des interêts des groupes est la possibilité de réaliser un protocole demande/réponse de type 1 à N avec M réponses (cf chapitreII). Notre implantation actuelle de l'appel système call ne permet pas la réalisation d'un tel protocole. Cette limitation a deux origines: l'absence explicite de demande de réception (ce qui complique la réception successive de plusieurs réponses au même appel) et l'absence de garantie après un appel call qu'une réponse arrivée après que *délai,* cf §4.2, soit échu, ne pourra plus être consommée par le client.

Le paragraphe suivant présente comment les groupes de portes servent à l'implantation des groupes d'acteurs dans Chorus.

5. Les groupes d'acteurs

L'interface offerte par Chorus pour manipuler les acteurs est une extension de l'interface offerte par Unix (cf annexe II) pour la manipulation de processus. Cela concerne les opérations de création/destruction d'acteurs, l'opération de création d'un groupe d'acteurs (*setpgrp*) et l'opération d'envoi de signaux aux acteurs et aux groupes d'acteurs (*Kill*).

L'envoi d'un *signal Unix* est transformé par Chorus en l'envoi (ou la diffusion) d'un message à un acteur (ou à un groupe d'acteurs). Chaque acteur possède une porte spéciale, toujours ouverte, la *porte des signaux,* qui traite ces *messages de signal.* Cette porte est complètement cachée du code de l'acteur et est associée à un traitement standard protégé, que les acteurs ne peuvent pas modifier directement.

Un acteur est identifié par l'UID de sa *porte des signaux.* Comme cet identificateur est global et unique, l'acteur peut migrer dans le système sans problèmes (par exemple,

en exécutant un exec à distance). Le traitement standard, associée à la *porte des signaux,* déclenche les actions prévues par les différents signaux reçus en fonction du code du signal présent dans le message et des actions associées par l'acteur au signal correspondant. Elle exécute aussi les contrôles de protection.

Dans Unix un processus peut appartenir à quatre groupes de processus. Le groupe des "processus application" du système, le groupe des processus travaillant au nom du même usager, le groupe des processus attachés à la même console et le groupe des processus issus de la hiérarchie de processus, groupe de processus créé par l'appel *setpgrp* (cf annexe II), qui crée un nouveau groupe de processus auquel l'appelant appartient automatiquement. Il devient le *process leader* de ce groupe (c.a.d., la racine du nouveau sous-arbre de processus). Tous ses fils appartiendront (par défaut) à son groupe. Toutes ces fonctionnalités sont aussi disponibles sur Chorus.

Ainsi, dans Chorus, la *porte des signaux* de chaque acteur appartient, en permanence, à plusieurs groupes de portes, correspondant chacun à un groupe d'acteurs. Les groupes de portes correspondant aux groupes des consoles sont créés par les acteurs contrôleurs des consoles. Les groupes de portes correspondant aux groupes d'acteurs explicitement créés par **setpgrp**, sont créés suite à cet appel. Les autres groupes sont des groupes prédéfinis spéciaux. Les UID de ces groupes sont maintenus dans le contexte de l'acteur. C'est le système qui, suite à la création (destruction) de l'acteur, introduit (retire) automatiquement la *porte des signaux* de l'acteur dans les groupes de portes correspondants.

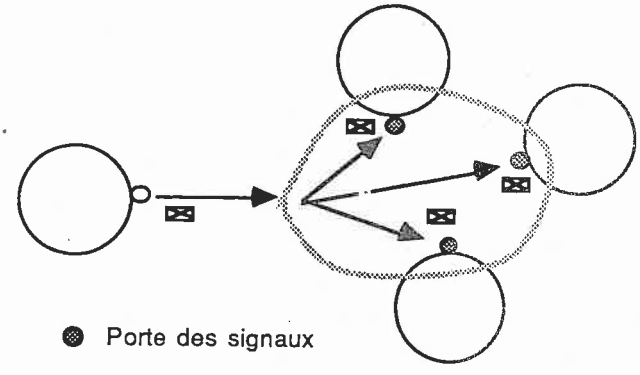


fig. 4.7 - Diffusion d'un signal sur un groupe d'acteurs

Chorus offre aussi les opérations:

suspend (acteur ou groupe d'acteurs)

qui envoie un message "signal de suspension" de l'acteur récepteur (ou groupe d'acteurs), et

resume (acteur ou groupe d'acteurs)

dont l'effet est contraire. Ces opérations ont été prévues pour le contrôle et la mise au point d'applications réparties, cf chapitre II. Elles sont implantées en utilisant des *messages de signal* spécifiques.

Comme nous venons de voir, Chorus offre deux concepts de groupe: les groupes de portes et les groupes d'acteurs. Les deuxièmes étant implantés sur des groupes des *portes des signaux,* des portes spéciales des acteurs. L'interface offerte pour la manipulation des groupes d'acteurs est une extension de celle offerte par Unix pour le traitement des groupes de processus et les signaux.

Les groupes ont d'importants rôles (cf chapitre II). Pour le contrôle de l'exécution, pour la diffusion de messages et pour matérialiser la notion de service réparti avec des points d'accès dupliqués. Dans Chorus, les groupes d'acteurs s'adressent au premier de ces objectifs et les groupes de portes aux deux autres.

A notre connaissance, les seuls systèmes qui offrent à leurs clients des notions de groupe de processus (et/ou de portes), autres que Chorus, sont Unix (en environnement centralisé ou réparti - systèmes "Unix répartis") et le V-System (en environnement réparti). Les groupes de processus d'Unix servent, pour l'essentiel, au contrôle de l'exécution; les groupes de processus du V-System réalisent toutes les fonctionnalités citées ci-dessus.

Le V-System n'offre que la notion de groupe de processus parce que la communication dans ce système se fait directement de processus à processus. Son interface de manipulation des groupes présente aussi d'autres différences mineures avec celle qui est offerte par Chorus. Cependant, en ce qui concerne la communication par messages au travers des groupes, Chorus et le V-System diffèrent sensiblement. Chorus offre la notion de mode d'adressage, notion absente du V-System. Cependant, ce deuxième permet la réalisation, de manière souple, d'un protocole de 1 à N avec M réponses, ce qui n'est pas le cas de Chorus, cf §4.7.

Pour terminer la présentation des fonctionnalités offertes par Chorus en ce qui concerne les portes et les groupes, il nous reste à introduire l'héritage de noms en général et des portes en particulier.

6. Contexte initial de communication d'un acteur

Chorus permet la création d'acteurs par le biais des opérations:

- **fork:** le fils résultant est une copie du père,
- **exec:** qui métamorphose l'appelant en un acteur avec un code, pile et données nouveaux, mais avec le même contexte et identification, et
- **fexec:** dont la sémantique est en première approximation équivalente à l'exécution d'un **fork,** suivi de l'exécution d'un **exec** par le fils.

Ces opérations peuvent être exécutées aussi bien en local, qu'à distance. L'appelant spécifie dans une variable d'environnement le nom symbolique du site de chargement du fils.

En ce qui concerne l'héritage du contexte d'exécution (sans considérer l'environnement de communication), Chorus obéit aux spécifications d'Unix. Cela inclut aussi l'héritage des fichiers ouverts (cf annexe II). Pour définir les règles d'héritage du contexte de communication d'un acteur nous avons pris en considération deux sortes de contraintes:

- respecter, dans la mesure du possible, le "style Unix" sousjacent à l'usage Unix de l'héritage des fichiers ouverts (cf annexe II);

- offrir des fonctionnalités de configuration d'applications réparties en utilisant l'héritage des noms contextuels (PGD), comme nous l'avons introduit au chapitre II.

Ces deux contraintes sont partiellement contradictoires. En effet, comme les fichiers ouverts sont des ressources externes aux processus, le père et le fils peuvent les partager par héritage; le *droit d'émission* sur une porte ou sur un groupe de portes sont aussi partageables; cependant, le *droit de réception* ne l'est pas.

Du point de vue de l'héritage des droits sur les portes pour la configuration d'applications réparties, les situations envisageables sont les suivantes:

- le père veut faire partager (sélectivement) par le fils les droits en émission qu'il possède,
- le père veut passer (sélectivement) au fils le droit en émission sur les portes qu'il a ouvertes, tout en gardant le droit de réception,
- le père veut passer (sélectivement) au fils le droit en réception sur des portes à lui, tout en gardant le droit en émission.

Après ces préliminaires, nous introduisons ci-dessous les règles d'héritage du contexte de communication.

6.1. Héritage des noms des groupes et des noms des portes non ouvertes par l'acteur

Chaque nom contextuel de porte ou groupe possède un attribut appelé *attribut de transmission*. Cet attribut est associé au nom et non à l'entité qu'il désigne. Il est donc contextuel.

Un acteur peut positionner cet attribut par le biais de l'opération (remarquer l'analogie avec l'appel *fcntl* d'Unix):

pgdcntl (pgd, transmit)

pgdcntl (pgd, no-transmit)

A l'issue d'un **fork**, le fils est une "copie" du père, donc, par analogie, il voit exactement les même noms que son père. Ces noms ont l'attribut de transmission positionné comme dans le père.

Cependant, a l'issue d'un **exec** ou d'un **fexec**, il y a la création d'un contexte nouveau (partiellement dans le cas d'un exec). Le fils ne voit dans ce cas, que les noms que son père lui a transmis. Le père peut ainsi contrôler les droits d'émission initiaux de son fils. Si les portes ou les groupes vus par le père représentent des points d'accès à des ressources, le fils peut les partager avec le père.

6.2. Transmission du contexte concernant les portes ouvertes

Quand un acteur exécute l'appel **fork**, son fils aura autant de portes ouvertes que lui, avec les mêmes noms contextuels et les mêmes attributs, mais ces portes sont nouvelles. Si le père dialogue avec des serveurs, le fils disposera de portes ouvertes qui lui permettront aussi de continuer ce dialogue. En particulier, le fils aura une *porte des appels système* et une *porte des signaux* nouvelles.

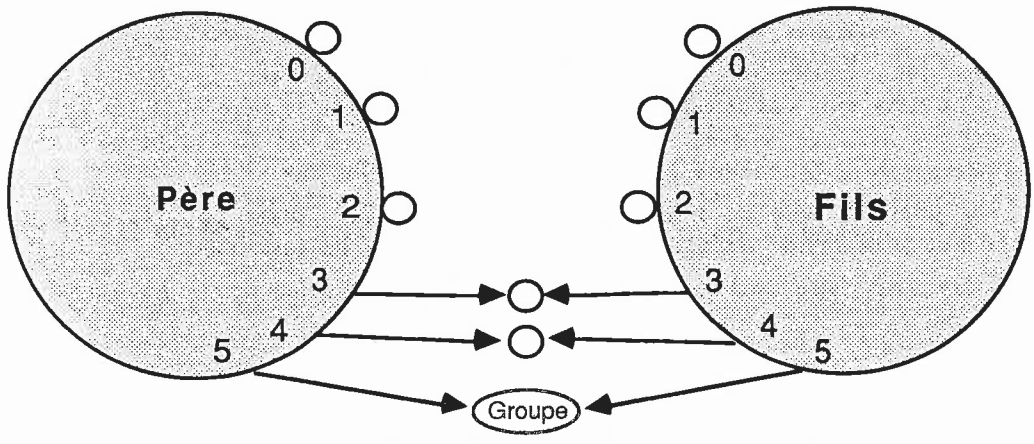


fig. 4.8 - Transmission du contexte de communication au **fork**

Quand un acteur exécute un **exec**, toutes les portes de l'appelant dont le nom contextuel a *l'attribut transmission* positionné sont transmises au nouveau contexte, et toutes les portes ouvertes sans cet attribut sont automatiquement fermées (et éventuellement détruites si elles ont *l'attribut destruction automatique* positionné). Si l'exec est à distance, les portes transmises migrent automatiquement. La *porte des appels système* et la *porte des signaux* sont toujours transmises pour que l'acteur puisse continuer à s'exécuter.

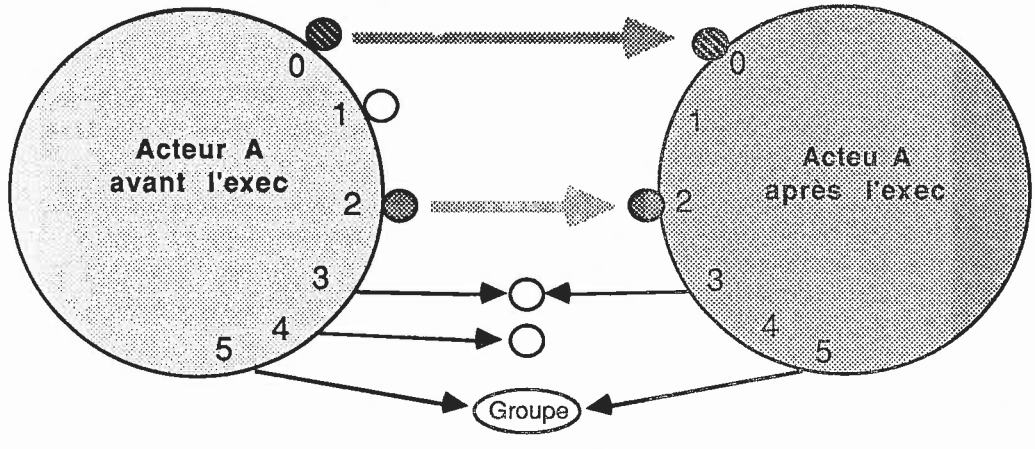


fig. 4.9 - Transmission du contexte de communication à l'exec

Finalement, quand un acteur exécute l'appel **fexec**, son fils aura autant de portes ouvertes que celles que son père a voulu lui transmettre, avec les mêmes noms contextuels et les mêmes attributs, mais ces portes sont nouvelles. Naturellement, la *porte des appels système* et la *porte des signaux* sont toujours *transmises*.

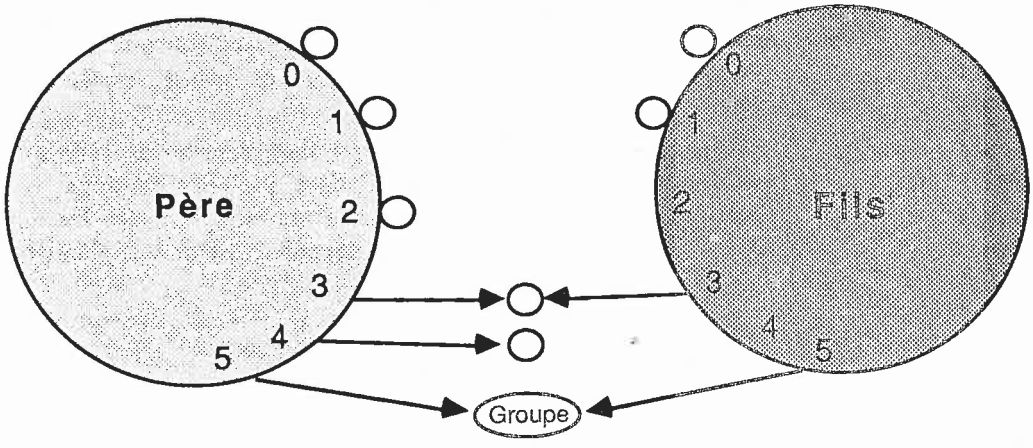


fig. 4.10 - Transmission du contexte de communication au **fexec**

Ces règles respectent pour l'essentiel "l'esprit Unix" mais, elles ont comme repercussion que ni le père, ni le fils, n'ont connaissance mutuelle des portes l'un de l'autre; ceci est en contradiction avec les besoins de chargement d'applications réparties. Nous avons rémédié à ce problème par l'introduction d'une porte avec un rôle spécial.

6.3. Protocole d'édition de liens entre père et fils

Dans chaque acteur, une porte ouverte (et une seule) a l'attribut *ombilicale :* cette porte sera utilisée pour la communication entre le père et le fils après la création de ce dernier. L'opération :

ubport (pgd)

permet à l'appelant de spécifier quelle est sa porte dédiée à cette fonction. La *porte ombilicale* ne peut pas être cachée au fils donc, quand celui-ci naîtra, il aura une nouvelle porte ouverte avec le même nom contextuel que celle du père qui sera sa *porte ombilicale.* Le système permet aux deux acteurs d'obtenir des noms contextuels désignant les *portes ombilicales* de l'un et de l'autre :

getchild → **pgd** (de la *porte ombilicale* du fils)
getparent → **pgd** (de la *porte ombilicale* du père)

6.4. Conclusions sur l'héritage

En partant des deux contraintes auparavant énoncées, les règles de transmission du contexte de communication présentées ci-dessus, ont été choisies pour Chorus.

Bien que la contrainte "maintenir le style Unix" l'emporte partiellement sur la contrainte "configuration de réseaux répartis de processus communicant par messages", ces règles permettent aussi l'installation de ces derniers.

Supposons, par exemple, qu'une application répartie soit constituée par un acteur "producteur" qui importe le service "put", par un acteur "consommateur" qui importe le service "get" et par un acteur "buffer" qui exporte les services "put" et "get". Ces deux services sont représentés par des portes ouvertes de l'acteur "buffer".

Un acteur "configurateur" pourrait alors créer les portes "put" et "get", passer le nom de "put" à l'acteur "producteur", le nom de "get" à l'acteur "consommateur" et ces deux noms à l'acteur "buffer". Une fois que "buffer" aura ouvert les portes "put" et "get", l'application peut démarrer.

La même méthode peut aussi servir pour configurer l'application de telle sorte que l'acteur configurateur puisse fonctionner comme "outil de trace" des échanges entre les participants dans l'application comme il a été illustré au chapitre II. Des signaux **suspend** et **resume** serviraient alors pour contrôler le déroulement de l'application.

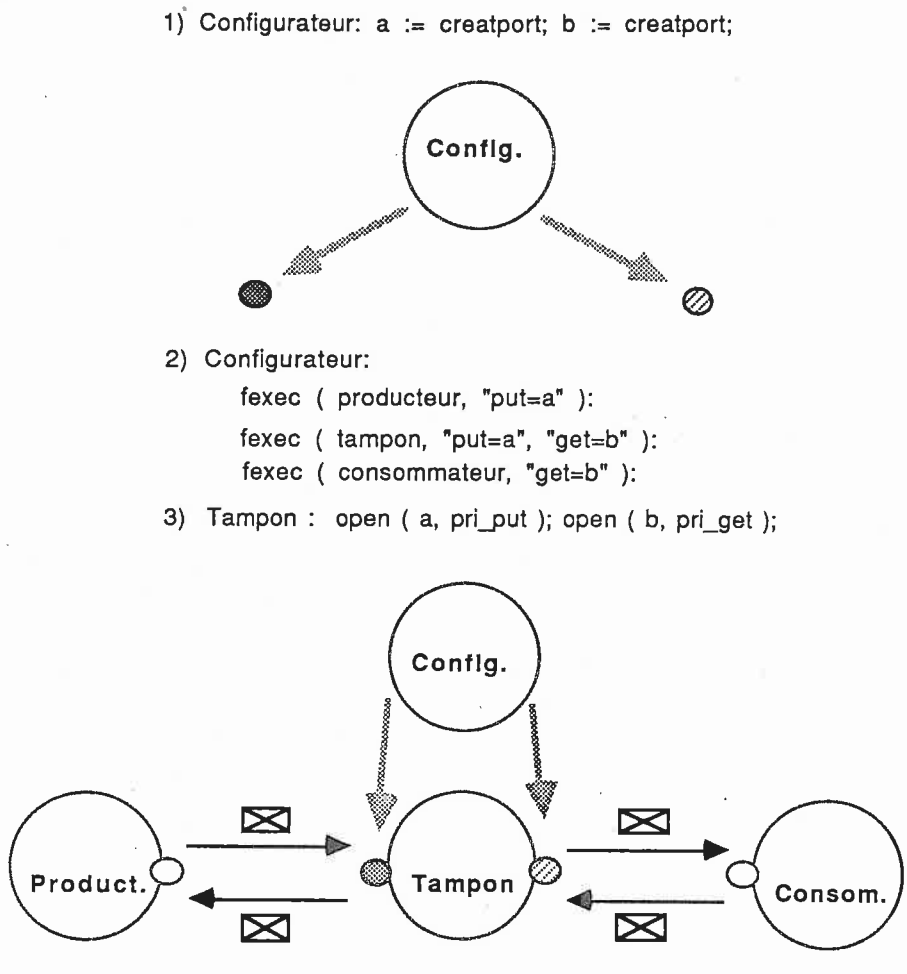


fig. 4.11 - Configuration par héritage d'un réseau réparti d'acteurs

Il est possible d'introduire des extensions dans le langage de commande d'Unix (le *shell,* cf annexe II) de telle sorte que son interpréteur puisse prendre le rôle de l'acteur "configurateur" de l'exemple ci-dessus.

7. Réalisation

La gestion des portes et des groupes de portes est réalisée par le service réparti de gestion des portes et groupes que nous avons réalisé. Une agence de ce service, le serveur gestionnaire de portes et groupes, est présente par site. Ces serveurs épargnent au noyau une grande partie de la complexité de la gestion des portes et des groupes de portes: le noyau ne connaît que les portes ouvertes par les acteurs de son site car c'est la seule information qui lui est nécessaire pour délivrer des messages localement; il n'est concerné ni par la migration, ni par la création/destruction des portes ou des groupes, ni par la localisation des portes distantes.

Les opérations de manipulation des portes et des groupes sont transformées par la bibliothèque d'interface en des envois de messages vers les portes du serveur de portes du site de l'appelant. La bibliothèque se donne accès au contexte de l'acteur pour transformer les noms contextuels (PGD) en des noms globaux (UID) et mettre à jour ce contexte en fonction de l'opération réalisée.

Le service de gestion des portes et des groupes gère des catalogues répartis de portes et de groupes, ainsi qu'un ensemble de listes d'appartenance des portes aux groupes. Ces serveurs coopèrent aussi:

- avec le noyau à l'ouverture/fermeture des portes,
- avec les acteurs serveurs gestionnaires d'acteurs pour traiter le contexte d'un acteur lors d'une création ou d'une destruction d'acteur,
- et avec les serveurs gestionnaires du réseau lors de la localisation des portes.

La réalisation de ces services est détaillée ci-dessous.

7.1. Portes et groupes statiques

Pour des raisons de performance et pour permettre l'édition de liens à la compilation, les services système sont réalisés derrière des portes qui ont des noms globaux, mais *statiques,* i. e., des noms dont les estampilles sont connues à la compilation. Les serveurs système ont ainsi des portes dont les noms sont prédéfinis. Les procédures d'interface qui donnent accès à ces services envoient des messages vers ces portes.

Ces portes ne peuvent pas migrer et ne sont accessibles qu'aux acteurs système. Des groupes statiques sont associés aux portes statiques: un groupe statique contient toutes les portes statiques qui ont la même estampille prédéfinie.

Les groupes statiques sont des groupes prédéfinis spéciaux qui caractérisent les services système. Ces groupes et les portes statiques forment une base simple et puissante pour l'implantation du système. La diffusion sur un groupe statique (seul mode d'adressage accepté sur ces groupes) est implantée par coopération directe entre le noyau et les serveurs réseau.

7.2. Gestion des portes et des groupes

Les serveurs de portes gèrent un catalogue réparti de portes; ce catalogue ne contient pas de duplications: une porte P n'est répertoriée que par un et un seul serveur.

Ces serveurs sont multiplexés et coopèrent les uns avec les autres. Par exemple, l'ouverture ou la destruction d'une porte non locale, passe par la localisation du serveur qui la gère, soit afin de faire migrer cette porte vers le site de l'appelant, soit afin de retransmettre la requête du client vers le serveur qui la gère. La migration d'une porte consiste à transférer son descripteur (UID, attributs de protection et liste des groupes

auxquels elle appartient) entre les serveurs des deux sites et à mettre à jour les tables de localisation (cf §7.4).

Les serveurs de portes et groupes implantent aussi un catalogue réparti de groupes de portes; ce catalogue contient des descripteurs de groupes (UID et attributs de protection) et peut contenir des duplications parce que ces attributs ne changent pas pendant la vie du groupe.

Suite à sa création, un groupe G n'est répertorié que par le serveur qui sert le site du créateur; néanmoins, au fur et à mesure qu'il y a des portes qui sont insérées dans G, ou que ces portes migrent, son descripteur est dupliqué sur les différents serveurs.

Toutes les requêtes de modification d'un groupe de portes, concernant une porte P, sont toujours redirigées vers le serveur qui la gère. A cause des contrôles de protection, ce dernier ne peut exécuter les modifications de la liste des groupes attachée à P que s'il connaît le descripteur de G. L'acquisition du descripteur de G se fait en utilisant la diffusion sur un groupe statique des serveurs de portes et groupes.

La destruction d'un groupe ne peut être exécutée que par un serveur qui connaît son descripteur. Si le serveur qui reçoit la requête de destruction de G connaît son descripteur, il exécute les contrôles de protection, exécute la destruction en local, répond au client et diffuse la destruction à ses partenaires. S'il ne connaît pas le descripteur, il doit d'abord l'acquérir.

7.3. Héritage des noms contextuels des portes et des groupes

Le service de gestion des portes intervient aussi, suite aux appels de création et de destruction d'acteurs: les serveurs gestionnaires d'acteurs demandent aux serveurs de portes et groupes le traitement du contexte d'adressage pour la communication de l'acteur (héritage de noms contextuels, création des portes initiales - des *signaux, ombilicale, des appels système* -, traitement des groupes d'acteurs, etc.). S'il s'agit d'un exec à distance le service fait aussi migrer collectivement les portes ouvertes conservées par le nouveau code de l'acteur.

7.4. Acheminement de messages et localisation des portes

Le noyau Chorus n'est concerné ni par l'adressage des portes non locales, ni par l'adressage des groupes de portes. Ces deux fonctionnalités sont implantées par le biais de serveurs intermédiaires d'acheminement de messages . Les serveurs réseau sont les intermédiaires pour les portes distantes et les serveurs de portes et groupes pour les groupes de portes (cf ci-dessous).

Dans les versions précédentes de Chorus, l'algorithme de localisation des portes nomades était basé sur la diffusion, entre serveurs réseau, de demandes de localisation de ces portes. L'algorithme était accéléré par l'utilisation de caches qui étaient mis à jour dynamiquement par chaque serveur en fonction des adressages précédents [Guillemont 84a]. Cet algorithme était assez équivalent à celui qui est utilisé par le V-System pour localiser les processus nomades [Theimer 85].

L'utilisation de la diffusion pour la localisation soulève quelques problèmes (cf §8); afin de réduire la fréquence de son utilisation pour la localisation des portes, nous avons modifié cet algorithme dans la présente implantation. Chaque serveur de portes maintient une table qui indique le site de résidence actuel des portes qui ont été créées chez lui et qui ont migré par la suite. En cas de migration, ou de destruction de portes nomades, cette table est mise à jour.

Quand un message est adressé d'un site A vers une porte P non locale, une requête de localisation de P est dirigée vers une porte statique (qui n'a donc pas à être localisée) du serveur de portes du site où P a été créée (l'UID de P contient son site de création) et le message est ensuite acheminé vers le site où la porte réside. En même temps, l'information de localisation est introduite dans un cache du serveur réseau du site A, de telle sorte que la demande de localisation n'est ré-exécutée que si la porte migre à nouveau.

Une des caractéristiques intéressantes d'un algorithme de localisation est son comportement en cas de pannes. Dans DEMOS/MP, cf chapitre I, par exemple, les messages adressés aux processus nomades suivent partiellement leur chemin dans le système. Si un des liens de ce chemin est coupé par une panne, l'acheminement est impossible. Dans [Powell 83] sont discutées quelques stratégies compliquées pour éviter ce problème. Dans Chorus, si le site de création d'une porte nomade est en panne, la localisation immédiate de cette porte devient impossible. Face à cette situation, le serveur réseau du site de l'émetteur diffuse une demande de localisation de la porte et s'il obtient une réponse, il introduit cette information dans son cache de localisation comme dans le cas normal.

Les informations de localisation maintenues par les serveurs de portes doivent être considérées comme étant des suggestions de localisation, dans la mesure où leur mise à jour se fait de façon asynchrone lors de la migration/destruction des portes. En plus, les messages qui font cette mise à jour peuvent se perdre ou n'arrivent pas nécessairement dans l'ordre. En cas d'insuccès d'acheminement, le serveur ré-exécute un certain nombre de fois la demande de localisation et, comme solution extrême, il utilise alors la diffusion comme dans le cas d'une panne.

Un site ré-initialisé après une panne diffuse à tous les autres sites une demande asynchrone de mise à jour des tables de localisation de son serveur de portes. Celui-ci recevra le nom et le site de résidence actuel des portes nomades qui ont été créées auparavant sur son site.

7.5. Adressage des groupes de portes

Comme nous l'avons dit ci-dessus, les serveurs de portes et groupes sont les intermédiaires pour l'adressage des groupes de portes. La réalisation de cette fonction est la suivante:

Adressage en Diffusion

Quand le noyau reçoit un message M adressé à un groupe de portes G, selon le mode d'adressage *diffusion,* il diffuse M à un groupe statique de portes des serveurs de portes; chaque serveur reçoit une copie de M qu'il duplique et redirige sur chaque porte ouverte de son site qui appartient à G.

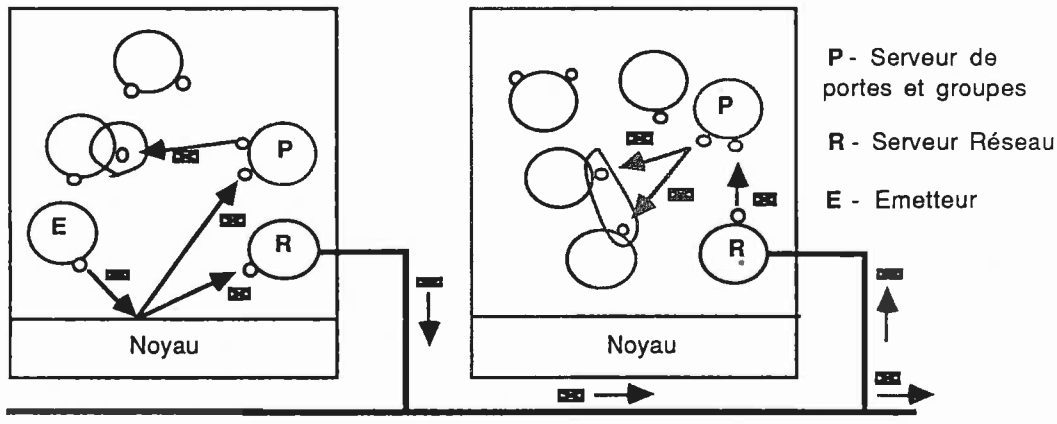


fig. 4.12 - Acheminement de messages lors de la diffusion sur un groupe de portes

Adressage Fonctionnel

Si le message M est adressé à un groupe G selon le mode *fonctionnel,* le noyau redirige M au serveur de portes de son site. Celui-ci teste si une porte ouverte appartenant à G existe sur son site, cas où il redirige M vers cette porte; sinon il diffuse à ses partenaires une demande de localisation d'une porte ouverte appartenant à G. M sera redirigé vers la porte indiquée par la première réponse reçue, ou détruit s'il n'y en a pas. Cette demande de localisation d'une porte ouverte appartenant à G se fait par diffusion sur un groupe statique de portes des gestionnaires de portes et groupes.

Dans tous les cas de figure, que le mode d'adressage soit *fonctionnel* ou *diffusion,* un serveur de portes n'envoie jamais le message M à sa porte émettrice initiale, même si celle-ci fait partie du groupe récepteur.

Comme nous venons de le montrer, le serveur gestionnaire de portes et groupes de portes est à la base de plusieurs fonctions du système. Ce serveur comporte 8000 lignes de code source, est écrit en Pascal et correspond à 32 K octets de code Motorola 68000. Configuré pour gérer 512 portes et groupes, 128 suggestions de localisation de portes et 1024 couples d'appartenance d'une porte à un groupe, il comprend 16 K octets de données.

8. Leçons et perspectives

Le système Chorus, après ses plusieures implantations, a suffisament mûri. Voilà quelques-unes des leçons tirées de notre expérience sur la gestion et la désignation des portes et des groupes et quelques perspectives envisagées.

Association dynamique des portes aux acteurs et migration

Ces possibilités se sont avérées, tout au long du projet, très intéressantes. Elles ont été utilisées pour l'étude de mécanismes de résistance aux pannes [Banino 85] et de configuration d'applications réparties, cf §6. En particulier, les opérations: **creatport, openport, closeport** et **dlport** ont toujours fait partie de l'interface du système. Nous pensons que ces fonctions résisteront à ses évolutions futures.

Utilisation des noms uniques et globaux

Notre expérience confirme que l'utilisation des noms uniques et globaux est une base solide et indispensable pour la construction de la désignation en environnement réparti. L'utilisation de noms réutilisés doit être limitée à des mécanismes de très bas niveau, jamais accessibles aux clients. Nous avons dû faire marche arrière à chaque violation de cette règle.

Nous avons utilisé différentes techniques de génération d'estampilles uniques. Nous pensons que la méthode la plus simple de les générer est celle qui repose sur l'utilisation de l'horloge de la machine (cf [Liskov 81]) mais, comme cette horloge est manipulable par les usagers, nous pensons qu'il est indispensable d'introduire dans les machines des horloges spéciales, prévues à cet effet.

Couches de désignation des portes et des groupes

Notre expérience confirme aussi que la visibilité directe des noms globaux et uniques par les clients du système n'est réaliste que dans des systèmes non protégés, dédiés et orientés temps réel. Cependant, ce qui est encore plus important à notre avis, les mécanismes d'édition de liens, de protection, de désignation symbolique et le rôle spécifique des couches langage, doivent être considérés comme déterminants dès le début du dessin du système. Nos hésitations à ce propos nous ont fait perdre beaucoup de temps.

Sur l'utilisation de la diffusion

Nous partageons le point de vue ([Cheriton 85], [Welch 86], ...) selon lequel la diffusion est un des moyens les plus simples de localiser une entité nomade, même si le problème des pannes est à considérer. Nous avons donc décidé de l'utiliser et nous avons aussi introduit la notion de groupe de portes dans le système pour l'offrir au niveau du client.

Malheureusement, la diffusion n'est implantable de façon performante que dans des réseaux locaux et de taille limitée et, ce qui est encore un problème plus délicat, elle n'est pas prévue par les implantations actuelles du standard ISO.

D'une certaine façon, nous pouvons assimiler notre implantation des services concernant les portes et les groupes, et leurs localisations, à l'implantation d'un service de gestion de noms tel que Grapevine [Birrell 82] par exemple. Nous savons donc que la seule façon de supprimer la diffusion de notre implantation passe par la duplication des catalogues et par l'introduction de protocoles de mise à jour de catalogues dupliqués, basés sur des communications de 1 à 1.

Ce problème est en effet très complexe dans la mesure où il y a une importante contradiction entre des objectifs tels que la généralité, la portabilité, l'indépendance du réseau et de la taille du système, d'une part, et la performance et la reconfiguration rapide en face des pannes, d'autre part.

En particulier, bien que réalisée sur le réseau Ethernet, notre implantation des groupes ne repose pas sur l'utilisation des adresses *multicasting* de ce réseau (son introduction s'avère assez simple). Nous pensons que cela limiterait la portabilité du système.

Nous venons de terminer la présentation des fonctionnalités concernant les portes et les groupes que nous avons introduits dans le système Chorus V2. Cette présentation est acompagnée de la justification de nos choix face à l'étude des problèmes que la répartition soulève en matière de désignation et d'édition de liens entre processus

communicant par messages (cf chapitres I et II).

Dans la version actuelle du système Chorus, toutes les fonctionnalités que nous venons de décrire sont disponibles, contribuant à former un système ouvert, complet et permettant, avec des performances qui s'annoncent correctes, une grande souplesse d'utilisation. Au chapitre suivant nous allons présenter la désignation symbolique dans ce système.

CHAPITRE V

LA DESIGNATION SYMBOLIQUE DANS CHORUS

CHAPITRE V

LA DESIGNATION SYMBOLIQUE DANS CHORUS

Une des caractéristiques les plus significatives d'un système réparti est la façon dont les entités du système sont désignées symboliquement. En effet, les systèmes répartis couvrent toute une gamme, allant des réseaux de systèmes fortement intégrés - SER - qui offrent à leurs usagers un environnement qui simule celui d'un système centralisé à temps partagé, jusqu'aux réseaux "grande distance" de systèmes faiblement intégrés, qui n'offrent que l'accès à un seul type d'entité - les boîtes à lettres.

Comme nous l'avons discuté au chapitre I, les choix et les solutions retenus pour résoudre le problème de la désignation symbolique en environnement réparti sont fonction des réponses apportées aux questions ci-dessous.

Environnement d'application

Quelles implications le style de l'environnement offert par le système et l'usage qui en est prévu ont-ils sur le sous-système de désignation symbolique (SDS):

1/ Quelles entités faut-il pouvoir désigner et quels sont les taux de mise à jour de leurs attributs?

Dans un réseau faiblement intégré, comme un réseau public, la désignation symbolique des ressources diffère suivant qu'elles sont locales ou distantes. Dans un tel environnement, le SDS réparti permet, pour l'essentiel, la désignation des sites et des usagers, i. e., des entités dont les attributs sont très stables (par des syntaxes telles que: *user.registry, user@host ou host1!host2!user,* ...)

Dans un environnement intégré, le SDS joue un rôle primordial: il doit permettre la désignation et l'accès à toutes les ressources, indépendamment de leur localisation et du taux de mise à jour de leurs attributs, par des syntaxes reflétant le caractère unique du système comme par exemple: */printers/laser, user, /users/jose/exemple.p,* ...

2/ Quel degré de compatibilité avec d'autres systèmes doit être offert afin de pouvoir récupérer leurs logiciels? Si le logiciel à récuperer provient d'un système centralisé, comme c'est le cas d'Unix, comment contourner les éventuelles limitations dues à l'origine centralisée de ce système?

Vision externe offerte par le système

Quelles sont les conventions de désignation offertes par le système? Un environnement intégré, tel qu'un système d'exploitation réparti (SER), exige un SDS offrant des noms symboliques, non seulement globaux (i. e., dont l'interprétation est indépendante de la localisation), mais aussi uniques (i. e., sans "suggestions logiques" de localisation), cf §6 chapitre I. Ces noms permettent la migration des entités et/ou leur duplication.

La mise en oeuvre d'un SDS offrant une telle vision externe du système exige, dans le cas général, la duplication de catalogues de désignation. Souvent, cette mise en

oeuvre exige également la duplication des ressources. Dupliquer des quantités peu importantes d'information, dont les taux de mise à jour sont faibles, est un problème pour lequel il existe des solutions. Cependant, dupliquer les ressources disponibles dans le système, et en particulier les fichiers, s'avère être un problème difficile qui déborde du domaine de la désignation symbolique.

Serveurs de noms vs Serveurs des ressources et de leurs noms

Le SDS est-il introduit par un service spécialisé (service de désignation ou *name service*), qui enregistre les noms et les attributs des ressources, comme le font la plupart des services répartis de courrier électronique, ou bien délègue-t-on vers les serveurs qui maintiennent les ressources la fonction de gérer leurs noms, comme par exemple, la plupart des serveurs de fichiers? (cf §7 chapitre I).

Accès aux ressources et édition de liens

Dans un SER, un nom symbolique désigne souvent une ressource, or l'application d'une suite d'opérations à la même ressource passe par l'établissement d'une sorte de transaction avec le serveur qui la gère (une forme implicite d'édition de liens). L'établissement de cette connexion doit aboutir à la traduction du nom symbolique en un point d'accès à la ressource. Quels moyens offre le système pour construire ces points d'accès de manière indépendante des localisations du client et du serveur?

L'interdépendance de ces choix étant importante, ils conditionnent de manière sensible l'adéquation des solutions introduites.

Dans la veine des SER axés sur la récupération de logiciel pré-existant, la solution retenue pour Chorus consiste, pour l'essentiel, à étendre les serveurs de fichiers de telle sorte qu'ils puissent offrir aussi quelques-uns des services caractéristiques d'un serveur de noms. Aucune des solutions prises par Chorus n'est en elle-même complètement nouvelle, c'est son ensemble, et plus particulièrement sa simplicité intrinsèque, qui conduisent à un résultat intéressant.

Ce chapitre commence par introduire le scénario, les objectifs et les problèmes que nous avons identifés pour la désignation symbolique dans Chorus. Le paragraphe 2 introduit une première solution que nous avons étudiée et explique pourquoi nous avons pris le choix de ne pas introduire un service spécifique de désignation. Le paragraphe 3 introduit la solution finalement retenue. Le paragraphe 4 montre comment, dans Chorus, se fait l'accès aux fichiers de manière indépendante de la localisation. Puis, le paragraphe 5 présente les nouvelles fonctionnalités qui ont été introduites dans les serveurs de fichiers Chorus pour qu'ils puissent rendre des services caractéristiques des serveurs de désignation. Le paragraphe 6 présente les conventions de désignation retenues pour Chorus et comment les usagers ont accès à des noms globaux. Les paragraphes 7 et 8 discutent respectivement de l'utilisation de ces conventions et de leur implantation. Finalement, nous dressons un bilan récapitulatif de notre solution et nous la comparons à des travaux similaires (paragraphes 9 et 10).

1. Scénario, objectifs et problèmes

Le scénario, les objectifs et les contraintes qui ont présidé à nos choix ont été les suivants:

Objectif 1

Chorus est un SER qui doit offrir un environnement intégré de développement de logiciel. Bien qu'implanté sur un réseau local de multi-processeurs, il doit présenter à ses usagers une importante transparence du réseau, de la communication et de la configuration du système. Les activités mises en oeuvre par le système sont celles qui caractérisent un environnement de développement: édition de programmes et de documents, compilation, d'importants taux d'accès aux fichiers, ... Cet environnement se caractérise aussi par un degré important de partage de fichiers. Le scénario typique est celui de quelques dizaines de machines interconnectées par un réseau local. Parmi ces machines, quelques-unes sont personalisées, d'autres sont partagées par plusieurs usagers, d'autres sont affectées primordialement, mais pas exclusivement, à la gestion commune des périphériques onéreux.

Objectif 2

Chorus doit offrir des moyens priviliégiés pour le développement et l'exécution d'applications réparties et aussi d'"applications réparties temps réel". En particulier, des moyens de désignation symbolique de services, de serveurs et d'opérations doivent être disponibles.

Contrainte

Le système doit étendre l'interface du système Unix et, en particulier, il doit permettre: de récupérer sa chaîne de développement (compilateurs et autres utilitaires d'aide au développement), de récupérer l'essentiel de sa chaîne d'utilisation (langage de commandes et commandes les plus souvent utilisées) et de récupérer, dans un premier temps, une bonne partie de sa chaîne de procédures et commandes administratives.

Ces deux objectifs et cette contrainte permettent de mettre en évidence un ensemble d'exigences.

Exigences qui découlent des objectifs

Ainsi, doit on pouvoir désigner, de façon indépendante de la localisation (i. e., par des noms symboliques globaux) et de préférence de façon unique (i. e., dont la syntaxe ne présente pas des exceptions dues au caractère réparti du système et sans "suggestions logiques" de la localisation), les usagers, les groupes d'usagers et les fichiers (Objectif 1), tout en gardant les conventions administratives caractéristiques du système Unix. Par exemple, la vision externe des ressources offertes par le système doit être indépendante du site de *login*.

D'autre part, il faut offrir aux usagers du système des moyens de désignation symbolique qui leur permettent le développement d'applications réparties (Objectif 2). Ceci passe par un SDS permettant la désignation de services et de serveurs. Ce système de désignation doit aussi permettre l'enregistrement des attributs de ces entités (leurs portes d'accès par exemple). Il doit être à la base des mécanismes d'édition de liens dynamique d'applications réparties, cf chapitre II.

Enfin, le caractère réparti du système exige que l'on puisse aussi désigner symboliquement ses sites et les serveurs qui le composent: l'administration du système et

le développement d'applications réparties (Objectif 2) soulèvent souvent le besoin de la visibilité explicite de la localisation.

Chorus doit donc se présenter comme un système à temps partagé unique, mais éclaté sur un ensemble de sites; les administrateurs, ou les usagers qui en ressentent le besoin, devant néanmoins pouvoir connaître de façon précise la configuration exacte du système. En plus, du point de vue de la désignation, le système doit présenter un degré important de compatibilité avec le système Unix.

Ces exigences soulèvent un ensemble de problèmes que nous décrivons ci-dessous.

Problèmes soulevés

Le seul choix important qui a été retenu dès le début de cet étude a été le suivant: pour des raisons de performance et de simplicité, les serveurs de fichiers de Chorus devraient aussi gérer les catalogues et les noms symboliques désignant ces ressources.

Naturellement, à cause des exigences ci-dessus, chaque serveur de fichiers est compatible avec les fonctionnalités (open, read, write, ...) et les conventions (arbre de désignation, syntaxe des noms, ...) du système Unix (cf annexe II).

Cependant, dans le système existent plusieurs serveurs de fichiers (avoir un seul serveur serait possible mais irréaliste des points de vue de la fiabilité et de la performance). Par conséquence, dans le système Chorus existent plusieurs arbres de fichiers compatibles Unix.

La présence de plusieurs arbres de fichiers pose un problème d'unicité de désignation des fichiers. En plus, Unix désignant les usagers et les groupes d'usagers à l'aide de fichiers (cf annexe II), un problème d'unicité de désignation des usagers et des groupes d'usagers se pose également.

D'autre part, il faut que l'on puisse désigner symboliquement les sites, les serveurs, les services, les opérations, ... (Objectif §2). Une solution pourrait consister à utiliser des fichiers banalisés pour contenir les noms et les attributs de ces entités (comme dans Unix pour les noms des usagers et groupes d'usagers). Cette solution est impossible dans Chorus parce que les UID ne sont pas directement visibles des clients du système. Un problème de désignation des sites, des services et des serveurs est donc également présent.

2. Première solution: analyse et critique

Pour résoudre ces problèmes nous avons analysé, spécifié et partiellement exprimenté, une première solution. Cette solution a du être abbandonnée par la suite à cause de ses défauts. Cependant, les leçons que nous avons tiré de cette expérience ont été assez riches et justifient partiellement les options qui sont à la base de la solution finalement retenue. Nous allons donc brièvement présenter et critiquer cette solution.

Elle consistait en l'introduction d'un service de désignation *(name service)* matérialisé par une base de données répartie mais logiquement centralisée. Cette base de données servait à enregistrer des noms et à leur associer des attributs [Legatheaux 85a]. Elle offrait le concept d'arbre de noms uniques† et globaux; cet arbre était structuré selon les besoins administratifs et non de localisation, comme celui de Grapevine

† Nous rappelons que nous utilisons le terme "nom symbolique unique" pour dénoter un nom symbolique interprété par un service réparti, mais logiquement unique. Ces noms ne contiennent pas d'exceptions syntaxiques dues à la répartition, ni de suggestion logique de la localisation de l'entité qu'ils désignent, cf chapitre I.