COURS

SYSTEMES ET APPLICATIONS REPARTIS

Laurence Duchien - Gérard Florin
Eric Gressier-Soudan - Claude Kaiser

Plan du document

Introduction : Construction de systèmes micro-noyaux. (E Gressier)

Chapitre I: **La gestion des communications**

- Mode message, (G Florin)
- Mode appel de procédure distante (G Florin)

Chapitre II: **La gestion des activités et la synchronisation répartie**

-Les techniques de tolérance aux pannes (G Florin)
- Ordres, état global dans les systèmes répartis (C Kaiser)
- Ordres, synchronisation (C Kaiser)
- Points de reprise (C Kaiser)

Chapitre III: **Le partage des données réparties**

- Les mémoires virtuelles partagées réparties:
  modèles de consistance mémoire, algorithmes (E Gressier).
- Les systèmes de fichiers distribués (E Gressier).
- La désignation dans les systèmes répartis (E Gressier)
- Les systèmes à objets répartis.(L Duchien)
- Le transactionnel réparti (L Duchien)

Cnam - Cedric

CONSTRUCTION DE SYSTEMES

-

MICRO-NOYAUX

Eric Gressier-Soudan

Plan

Introduction

Micro-Noyaux

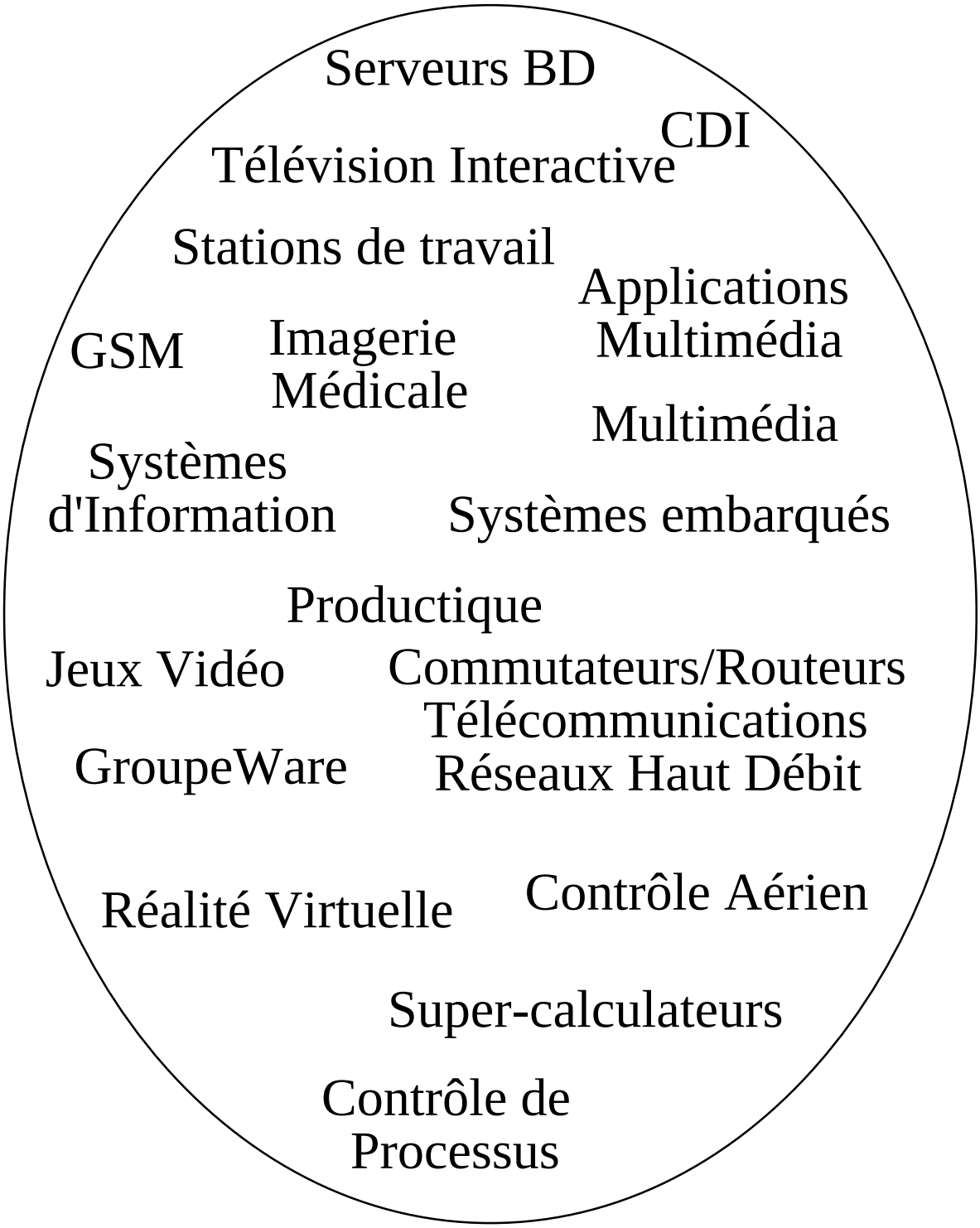
Sous-Systèmes

Systèmes à Objets

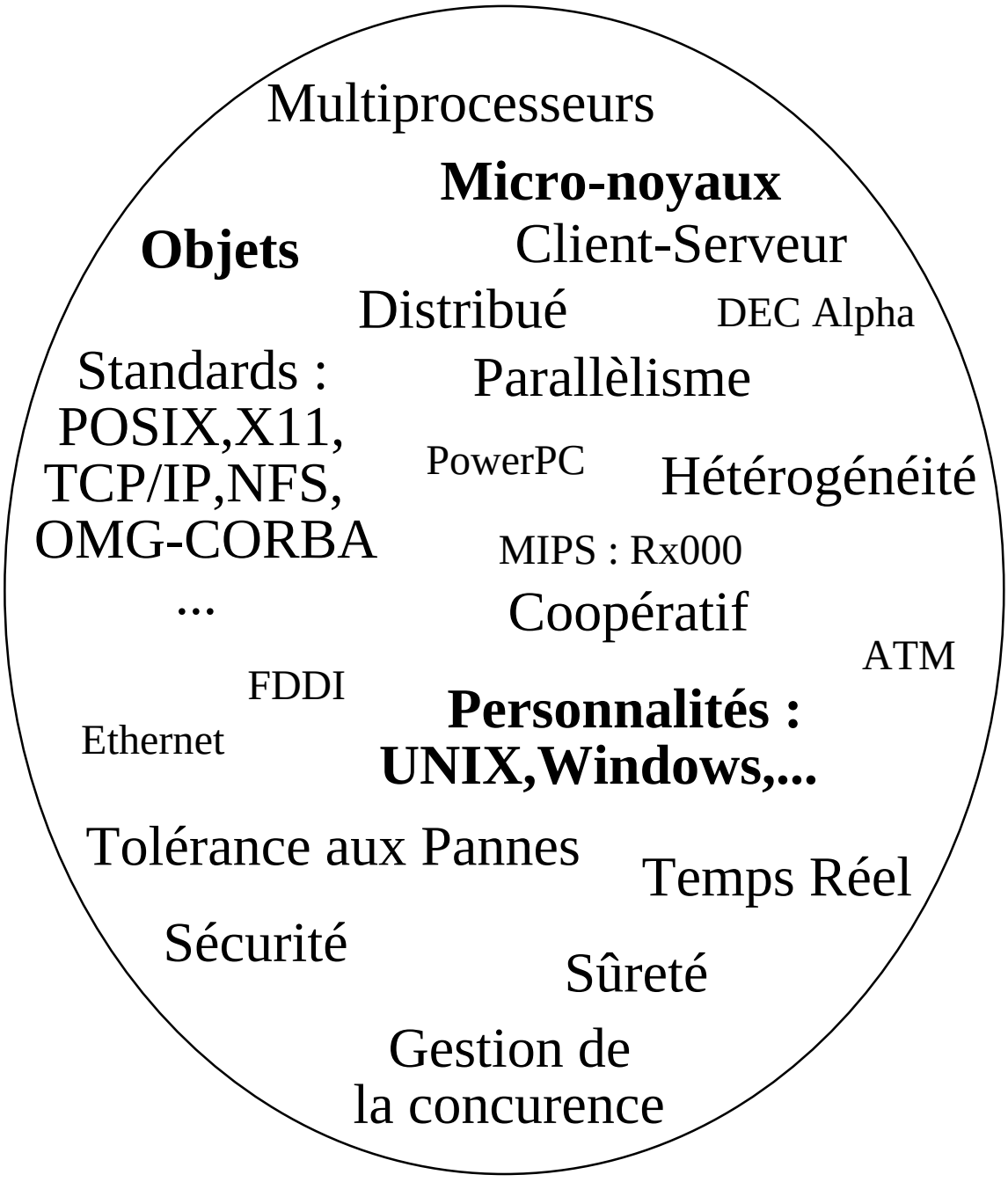
Abstractions de base

INTRODUCTION

APPLICATIONS



Caractéristiques des solutions



Objectifs en conception des systèmes répartis (1)

. **masquer** la répartition (**transparence**)

-> **localisation** des ressources *non perceptible*

-> **migration** des ressources possible sans transformation de l'environnement utilisateur

-> *invisibilité* de la **réplication** des ressources

-> **concurrence** d'accès aux ressources *non perceptible*

-> *invisibilité* du **parallèlisme** sous-jacent offert par l'ensemble de l'environnement d'exécution

Utilisateur = humain ou programme

Objectifs en conception des systèmes répartis (2)

. **Flexibilité** / Modularité

->Noyaux Monolithiques contre **Micro-noyaux**

Objectifs en conception des systèmes répartis (3)

. Fiabilité

N machines plus fiable qu'une seule ?

-> **Disponibilité** :

Leslie Lamport donnant une définition d'un système distribué :
"One on which I cannot get any work done because some machine I have never heard of has crashed"

Avec NFS, par exemple, on souhaiterait éliminer l'interdépendance des serveurs d'un réseau de stations pour éviter les attentes ou les blocages liés au montage de partitions distantes.

* indépendance des composants du système

* redondance logicielle et matérielle

Objectifs en conception des systèmes répartis (4)

. **Fiabilité**

-> **Tolérance aux pannes / Sûreté de fonctionnement** :

spécification : comportement vérifiable par **un ensemble de prédicats/des assertions** qui tient compte des entrées, des sorties du système et du temps qui lui est imparti pour exécuter son travail

mode normal : le système rend un service qui respecte à tout moment les spécifications sur lesquelles reposent sa conception

mode dégradé : la perte de fonctionnalité n'empêche pas les spécifications d'être respectées mais avec éventuellement une baisse de performance

mode panne : le comportement du système est hors spécifications

Objectifs en conception des systèmes répartis (5)

. **Fiabilité**

-> **Sécurité**

Protection contre les accès non autorisés

. propriétés visées :
contrôle d'accès,
authentification,
intégrité,
confidentialité,
non-répudiation

. techniques :
cryptographie à clefs privées ou à clefs publiques,
fonctions univoques ou hachage,
signature
protocoles de sécurité

Objectifs en conception des systèmes répartis (6)

. **Performance** malgré la transparence, la flexibilité, la fiabilité

. **Gérer le grand nombre**

beaucoup d'utilisateurs
de machines
de services
de trafic
=> centralisation impossible

hétérogénéité

intégrabilité et extensibilité

. **Maintenabilité** Pas pour les concepteurs ... mais pour les ingénieurs système !!!

Impossibilité des systèmes répartis

PAS D'ETAT GLOBAL
(facilement accessible)

. Pas de référence de temps (HORLOGE) commune entre les différents sites d'un système réparti !!!

. Impossibilité de fabriquer une image du système à tout instant ... pb des états globaux instantanés ("distributed snapshots")

Les prises de décisions se font sur la base d'informations locales à chaque site[1].

Les Univers Distribués sont foncièrement probabilistes

=> . Problèmes d'Algorithmique répartie

. Le Déverminage en réparti est une tâche complexe

[1] Le problème d'ouverture de connexion de transport est typiquement un problème d'algorithmique répartie résolu par une démarche probabiliste : au bout de 3 messages(three way handshake) on a un maximum de chances que la connexion soit ouverte entre deux sites ... mais on en a pas la certitude absolue (voir pb des deux armées).

Micro-noyaux

Paradigme

Point de vue de l'ingénierie des systèmes répartis :
Les micro-noyaux forment la base des nouvelles architectures de systèmes.

Démarche adoptée par :

. Novell/USL ... HP
. Microsoft
. Apple
. DEC
. Sun
. ALCATEL
. Siemens
. Cisco
. Unisys
. Cray
...

Génèse des micro-noyaux (1)

70's → 80's

. **Réseaux longues distances**

. **Protocoles de communication**

. **Applications dédiées :**

- **Transfert de fichiers**

- **Terminal Virtuel**

- **Messagerie Electronique**

Réseaux Locaux + Systèmes Monosites

. Systèmes de fichiers répartis (ex NFS-Sun)

. Environnement d'exécution réparti (ex ONC-Sun, NCA-Apollo)

Interconnexion

Services Répartis

→ **90's**

. Multi-calculateurs

. Système d'exploitation réparti et vision unifiée du système (Single System Image)

. Accès aux objets systèmes
- **Fichiers**
- **Processus**
- **Mémoire**
- **Périphériques**

. Objets Distribués
- **Groupes de processus**
- **Systèmes de fichiers**

. Objets Migrants
- **Processus**
- **Fichiers**

Distribution du système

Génèse des micro-noyaux (2)

Recherche sur les sytèmes (80's)

Grapevine (XEROX)

Accent (CMU)

Amoeba (Université d'Amsterdam)

Chorus (INRIA)

Mach (CMU)

V-System (Université de Stanford)

Sprite (Université de Berkeley)

Systèmes Commerciaux (90's)

Chorus (Chorus Systèmes)

Mach-OSF/1 (CMU-Open Software Foundation)

Amoeba (ACE)

Windows NT (Microsoft)

Système Monolithique qu'est ce que c'est ? (1)

Système Monolithique :

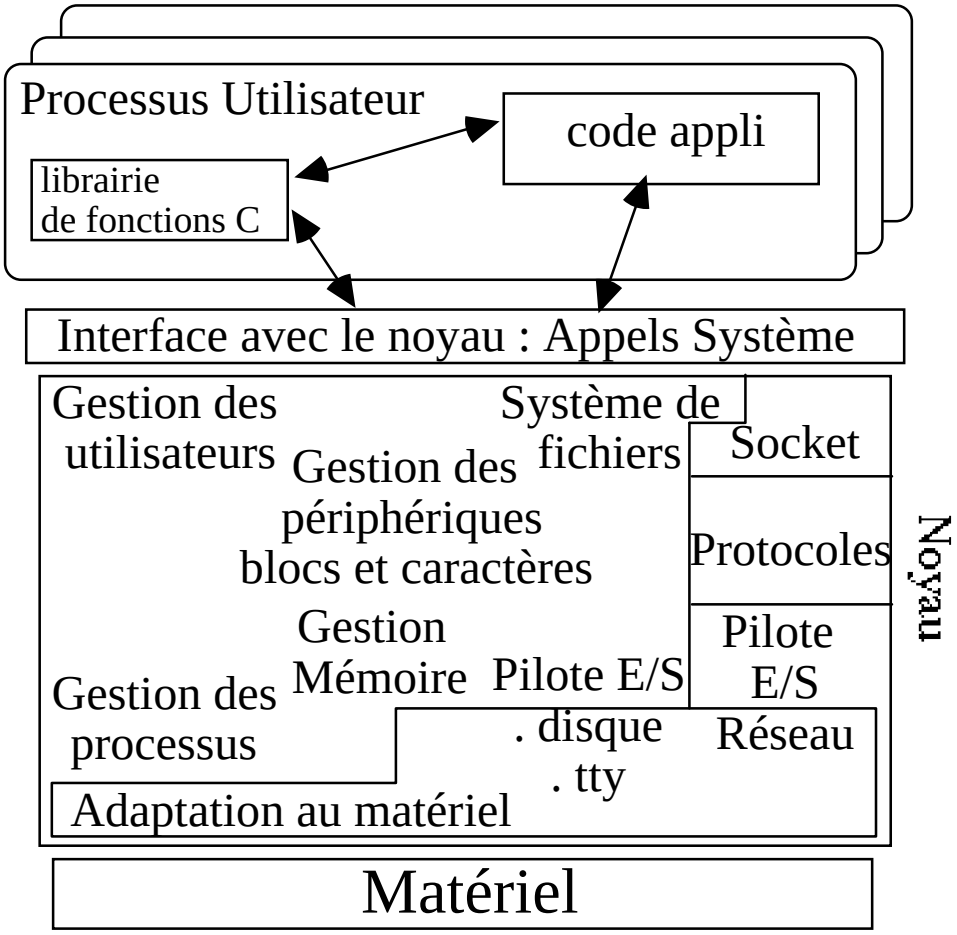
le système est en un seul morceau,

rien ne peut lui être enlevé sans mettre en péril son fonctionnement,

aucune possibilité de voir les ressources (ex fichiers) autrement que part la vue fournie par le système[2]

[2] Pas complètement vrai ... si on reprend Unix System V R4.0 ou 4.4 BSD, le système permet de voir des systèmes de fichiers de différents types : UFS (Berkeley -> rapide), S5 (traditionnel System V-> lent), NFS en particulier, grâce au mécanisme de Virutal File System et de Virtual node (VFS/V-node) empruntés à NFS.

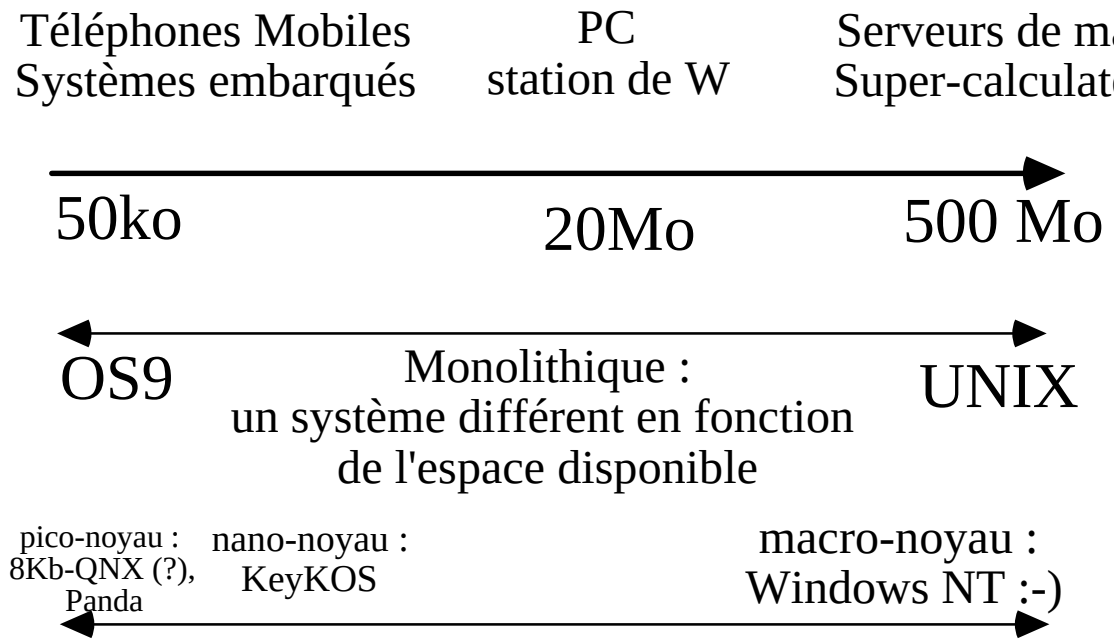
Système Monolithique qu'est ce que c'est ? (2)



Architecture qui ressemble à celle d'un Unix BSD ou d'un Unix System V

Contrainte Matérielle sur la conception du système

Place disponible en mémoire RAM pour héberger le système :



Micro-noyau modulaire :
le même mais avec plus ou moins de fonctions
un exemple : Chorus

Objectifs de conception d'un micro-noyau

. Transparence

- à la localisation
- à la migration
- à la réplication
- au parallèlisme

. Modularité

=>Extensibilité/Portabilité/Configurabilité

- implantation des services sous forme de gestionnaires ou de serveurs spécialisés
- réduction de la complexité
- développement et validation plus faciles

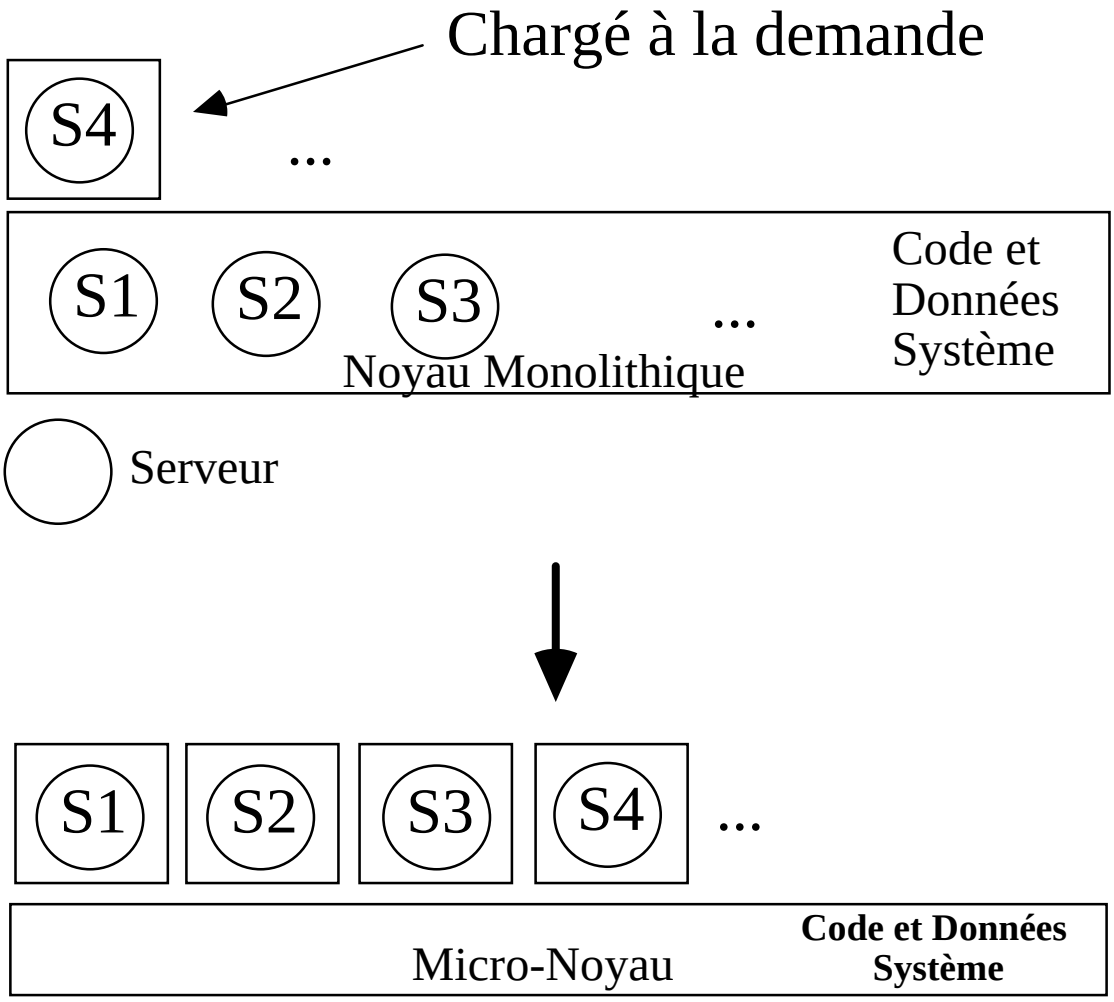
. Tolérance aux fautes/Disponibilité

. Performance

. Sécurité

et un Retour à la simplicité

Gain au niveau de l'architecture



Qu'en est-il de la performance ?

avantage au monolithique[3] ! ? !

[3] mais A. Tanenbaum cite une étude qui compare Sprite (Monolithique) à Amoeba (Micro-noyau) et qui indique qu'il n'y a pas de différence sensible ... il faut relativiser ce type d'affirmation ... il y a un vieux débat qui consiste à dire pour gagner de la performance il faut tout compacter le code du système, on y perd toute modularité donc tout les gains liés à l'ingéniérie des logiciels,
par ailleurs, les progrès technologiques, en particulier la baisse du prix des mémoires centrales et la vitesse des processeurs, ont permis de dédier une partie de

Organisation du système

. Micro-noyau

=> Fond de panier logiciel

. Serveurs

- de type système : superviseur
- de type utilisateur

. Sous-Systèmes ou machines virtuelles qui définissent des "Personnalités"

l'architecte système dispose d'un "patron"qu'il peut modifier/tailler en fonction de ses besoins

---

la bande passante des processeurs à cette modularité sans grande perte de performances, les micro-noyaux entrent dans cette logique

Sous-Système

Personnalités

Sous-système / Sur-système

1. survivre au standard Unix

pouvoir hériter de l'existant

-> compatibilité binaire des applications

éviter l'échec d'Apollo/Domain

2. coexistence d'applications spécifiques avec d'autres conçues pour des systèmes différents

-> temps réel

-> Mac

-> Windows

-> DOS

3. préparer la transition vers de nouveaux systèmes

systèmes à objets

Sous-système

Un sous-système :

. une librairie d'appels systèmes
. un serveur ou un ensemble de serveurs

qui permettent la compatibilité binaire pour des applications déjà écrites pour le système émulé.

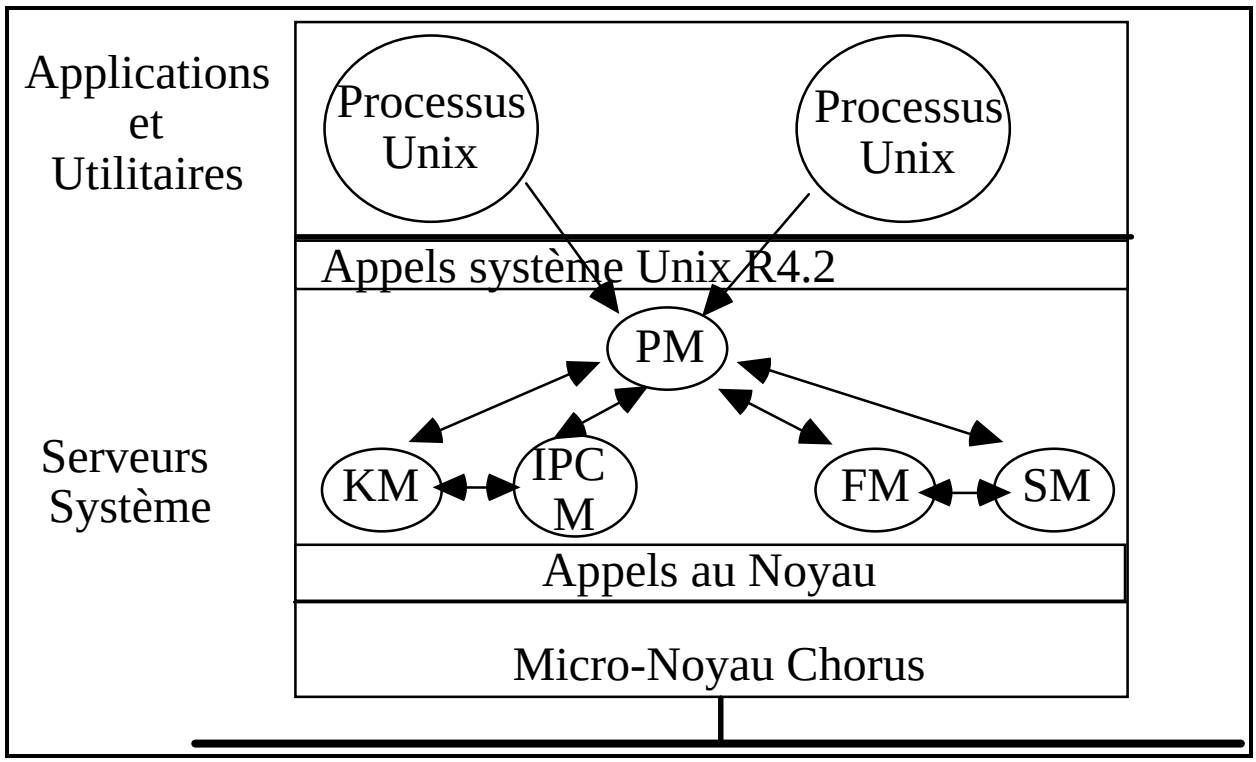
L'utilisateur ne "voit" que le système sur lequel il a l'habitude de travailler.

Image unique du système - "Single System Image"

En général, on lui offre aussi l'accès aux appels système du micro-noyau.

Exemple Chorus et MIX V4 R2

SUSI : Single Unix System Image



PM = Process Manager

FM = File Manager

SM = Streams Manager

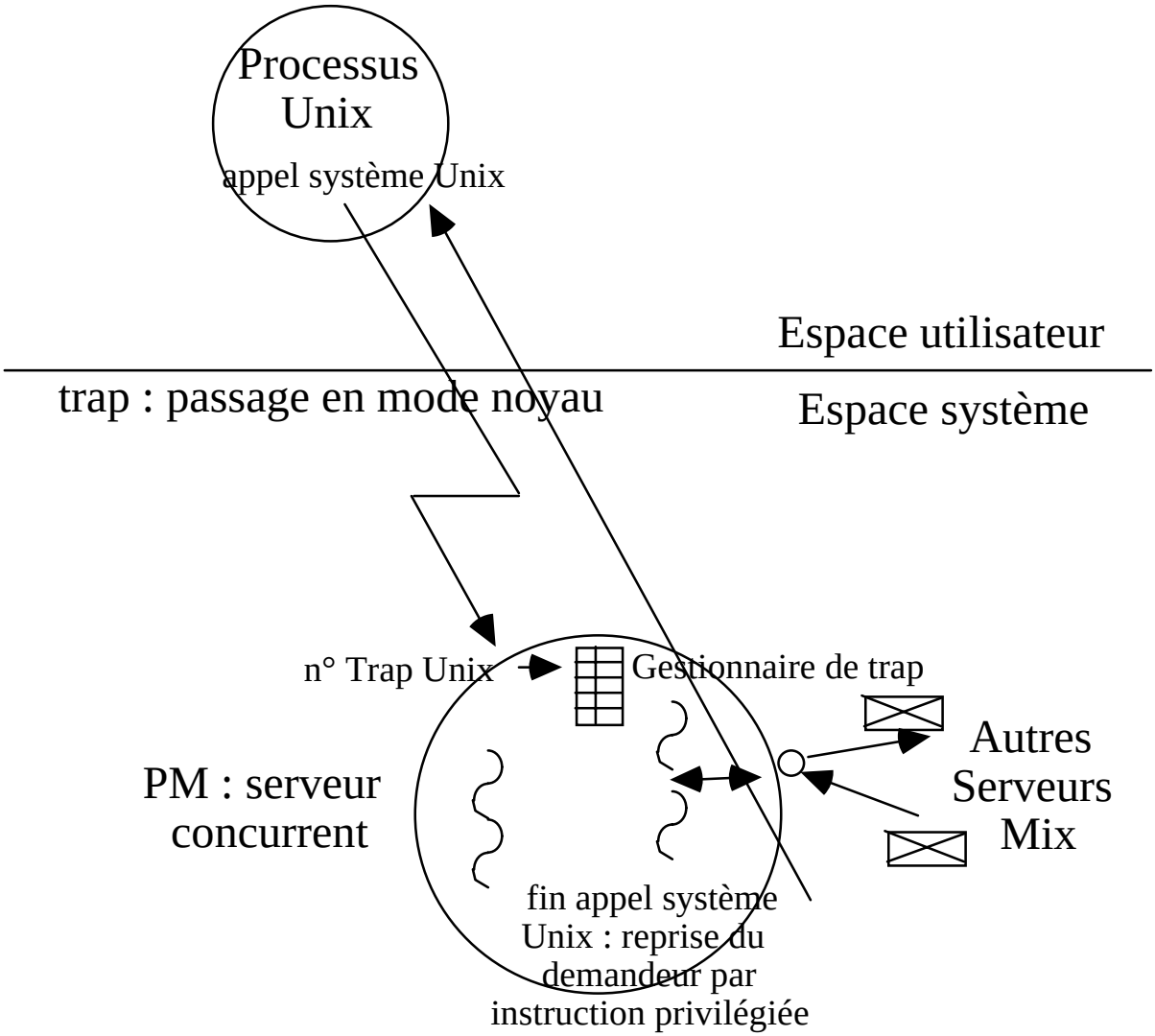
IPCM = System V Inter-Process Communication Manager

KM = System V Key Manager

Il existe un sous-système Mix V.3.2, Mix SCO, ClassiX/C-actors, et Mac OS (INT Evry - C. Bac)

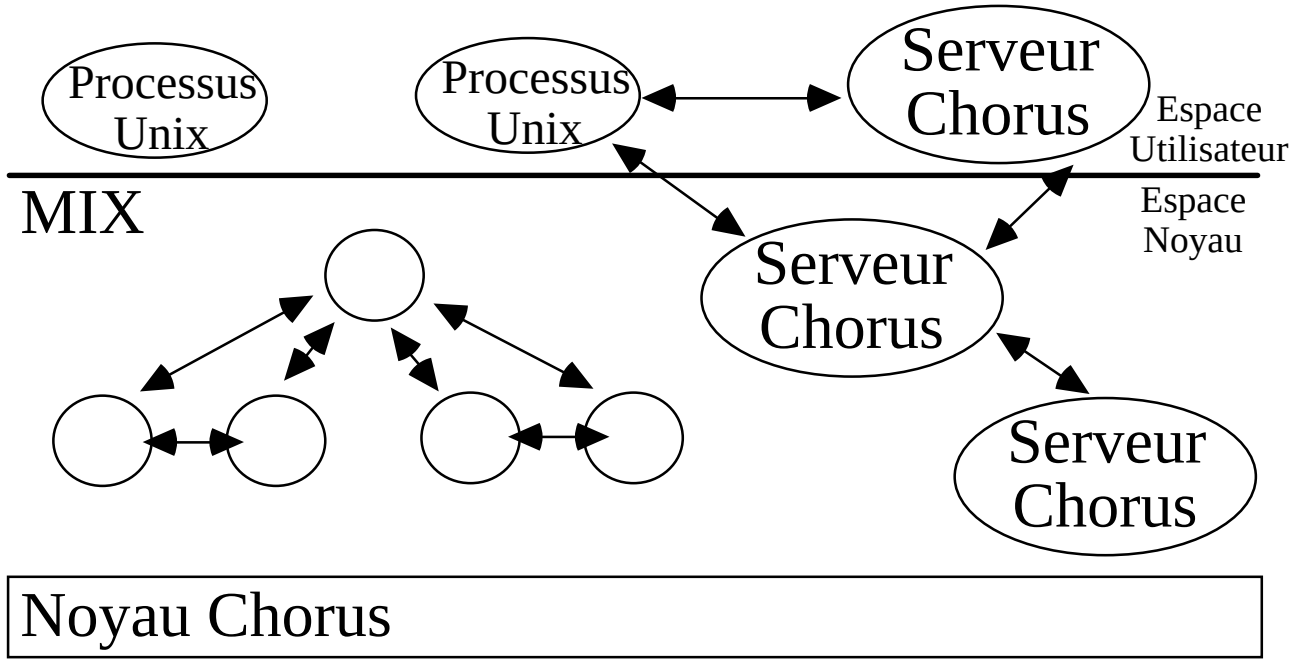
Emulation Unix dans Chorus

Compatibilité Binaire



Avantage de la notion de Sous-système

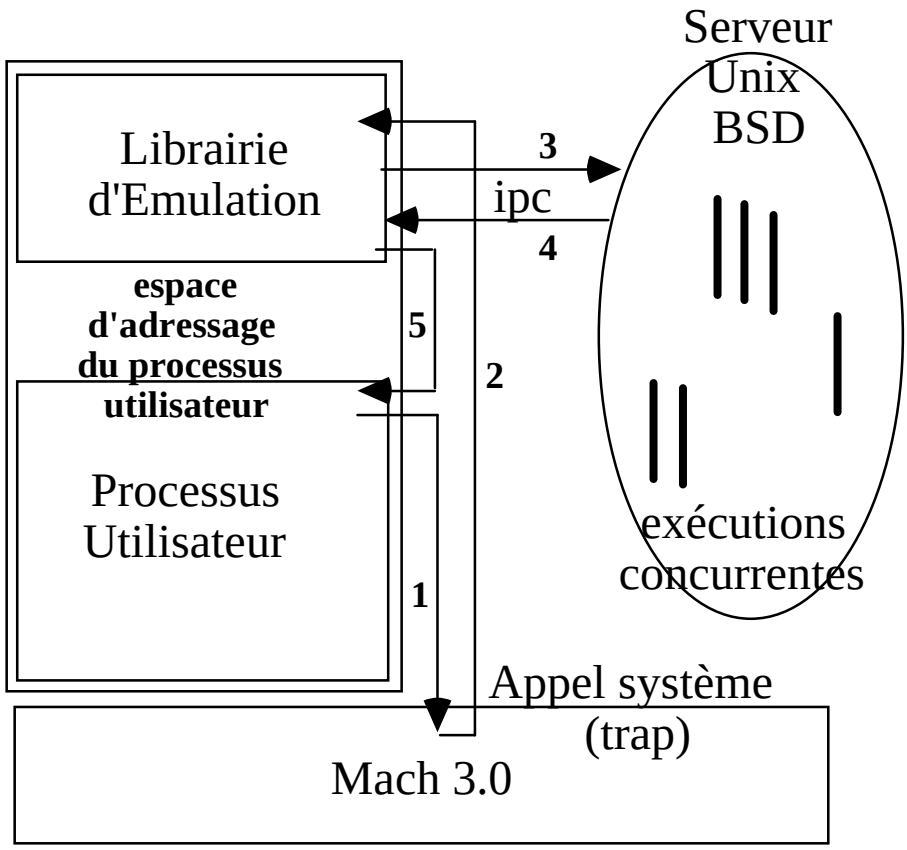
Cohabitation de processus UNIX avec des serveurs Chorus qui coopèrent dans le cadre d'une application temps réel.



Emulation Unix sous Mach

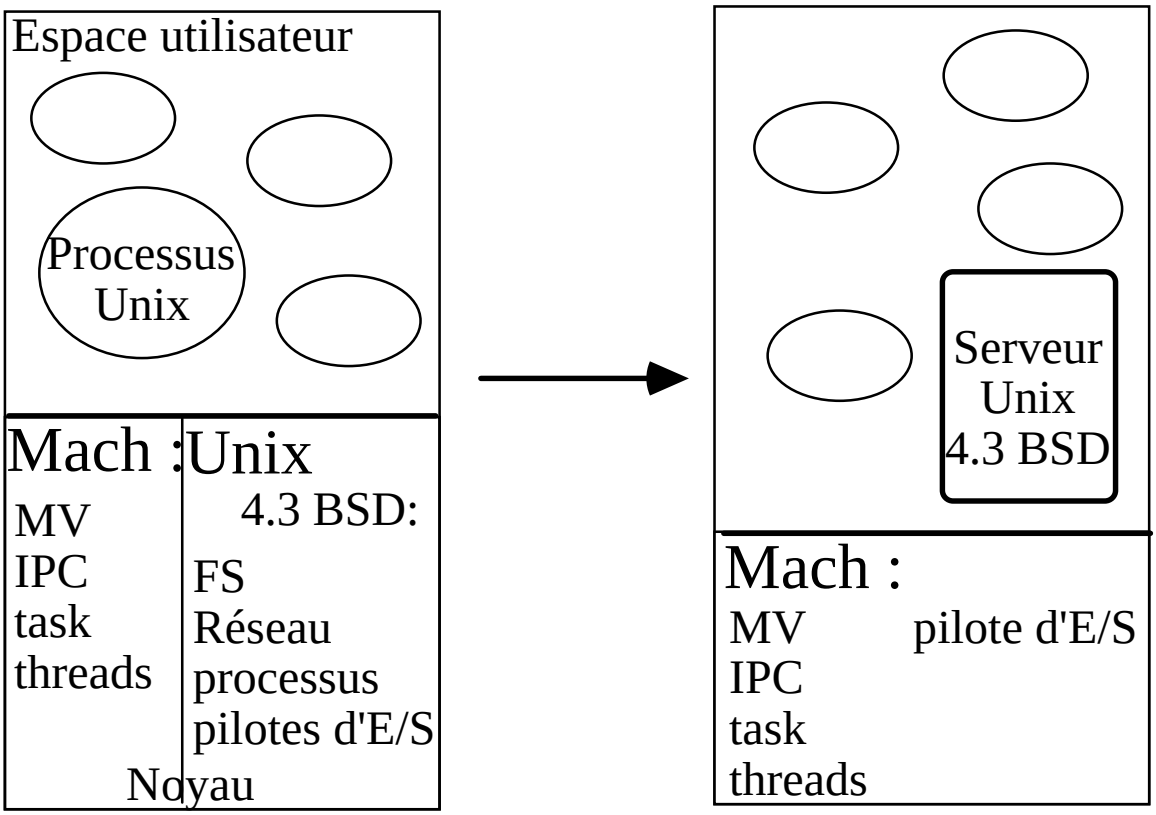
Librairie d'Emulation dans l'espace utilisateur

qui génère des communications (ipc) vers le serveur Unix BSD.



Effet de la modularisation sur les sous-systèmes et le micro-noyau (1)

Génèse de Mach/OSF1 :



Mach 2.5 / OSF1-IK
Integrated Kernel

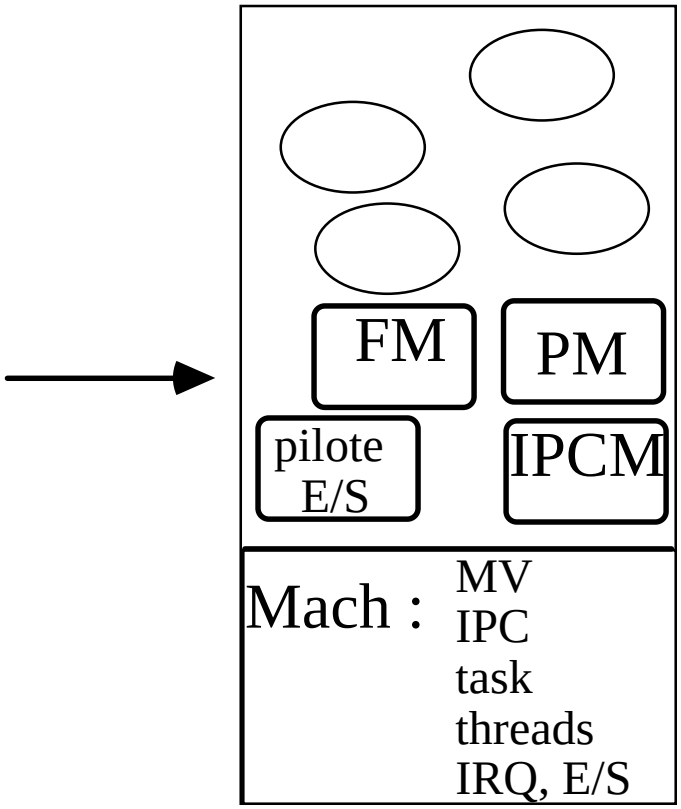
Unix parallèlisé, le support du multiprocesseur

Mach 3.0 / OSF1-MK
Micro-Kernel

Mach possède d'autres sous-systèmes prévus pour cohabiter : MS-DOS, VMS en particulier

Effet de la modularisation sur les sous-systèmes et le micro-noyau (2)

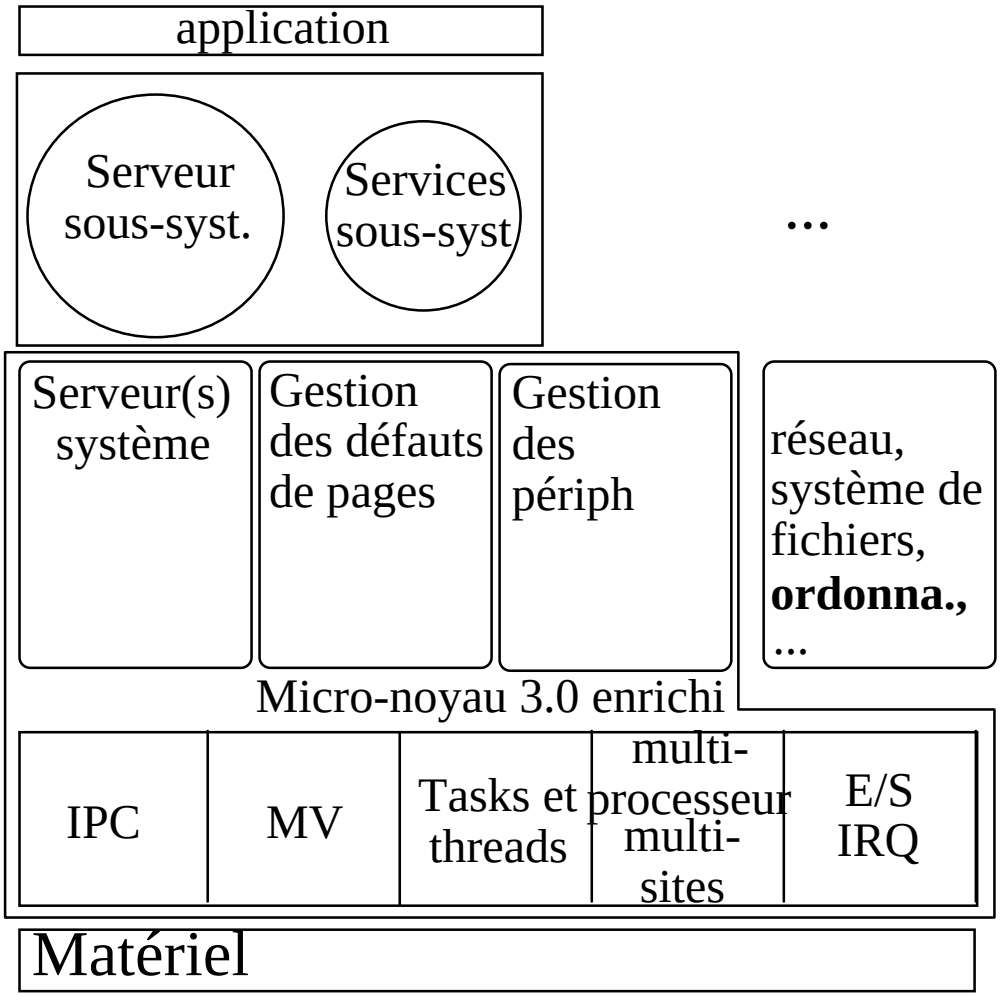
Génèse de Mach/OSF1 (suite) :



Mach / OSF1-AD

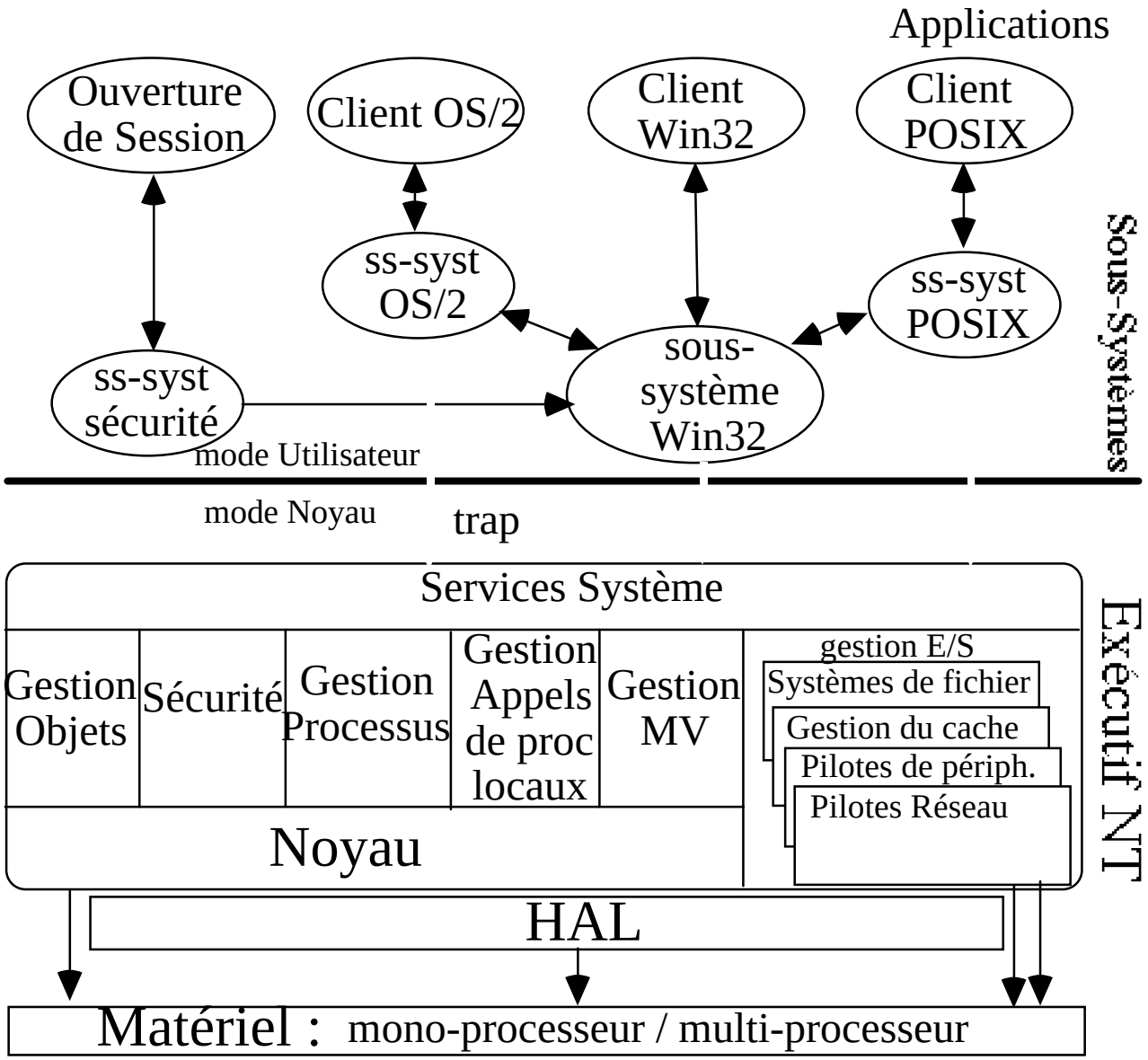
Advanced Development

Exemple : micro-noyau d'IBM pour Workplace OS (abandonné)



prévu pour supporter plusieurs personnalités

Exemple : l'Architecture de Windows NT

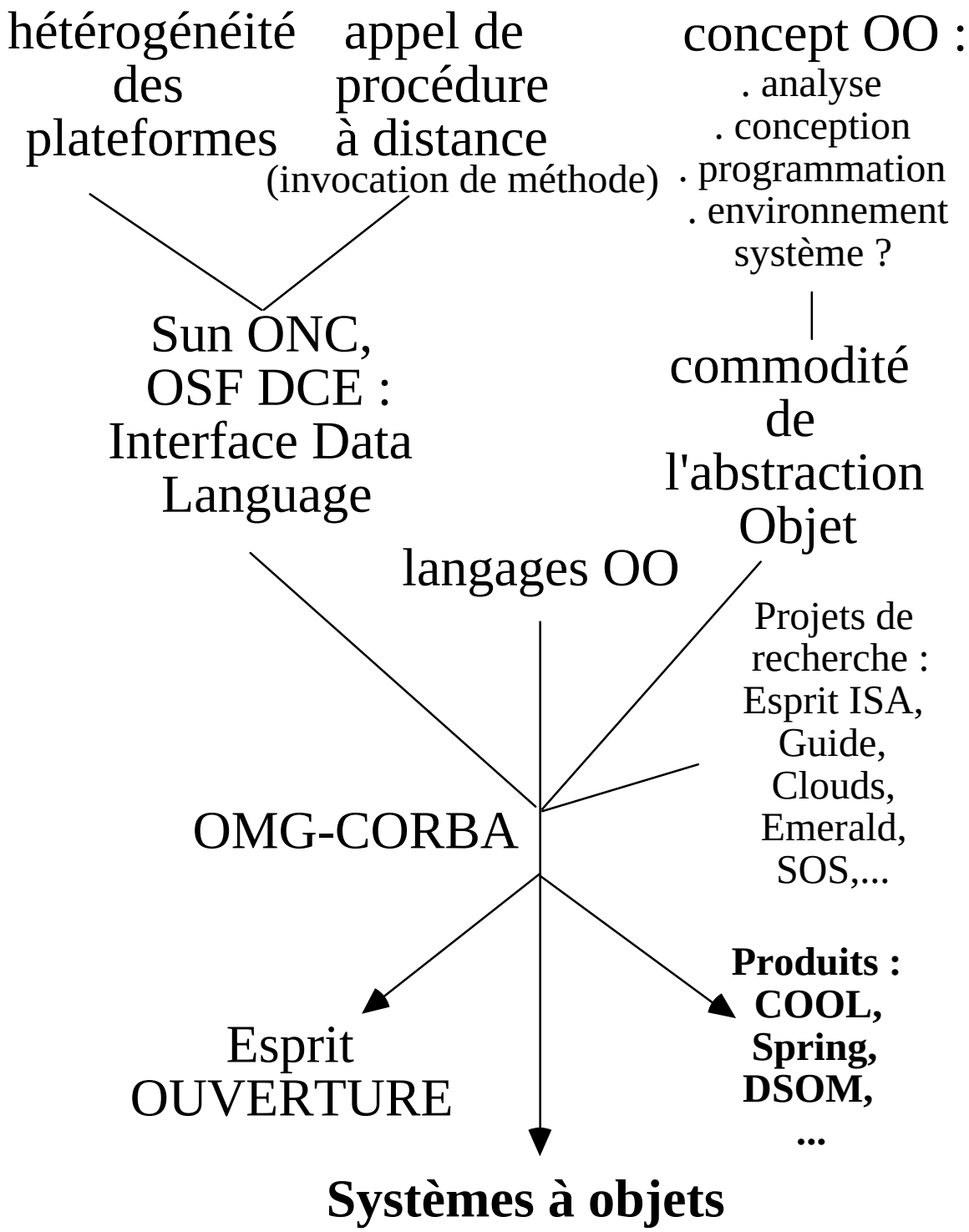


Prévu dès la conception pour fonctionner sur des processeurs différents : Mips, Alpha, Intel.

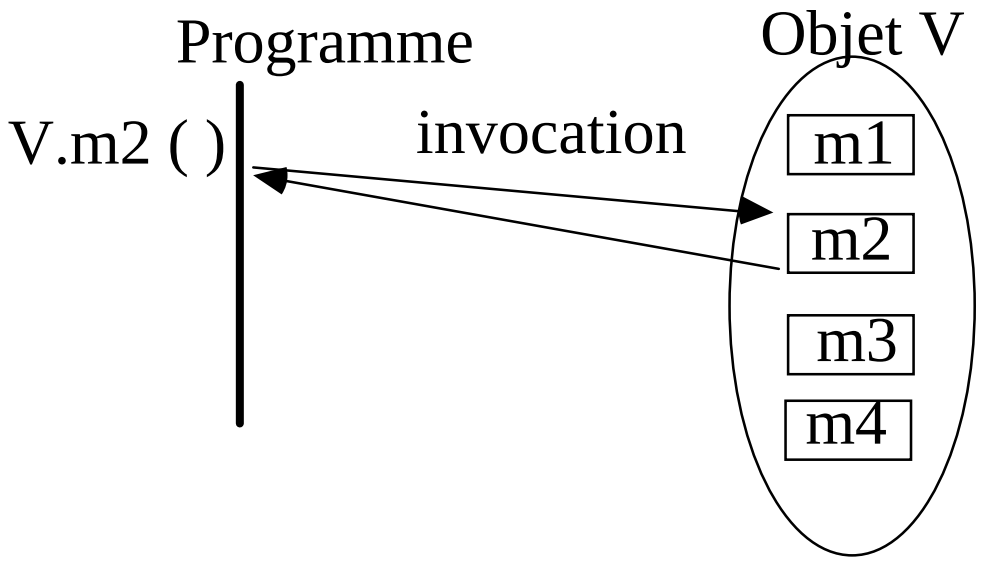
Windows NT bâti sur le modèle micro-noyau

Systèmes à objets

Objets


hétérogénéité des plateformes
appel de procédure à distance
(invocation de méthode)
concept OO :
. analyse
. conception
. programmation
. environnement système ?
Sun ONC, OSF DCE : Interface Data Language
commodité de l'abstraction Objet
langages OO
Projets de recherche : Esprit ISA, Guide, Clouds, Emerald, SOS,...
OMG-CORBA
Esprit OUVERTURE
Produits : COOL, Spring, DSOM, ...
Systèmes à objets

Invocation de méthode

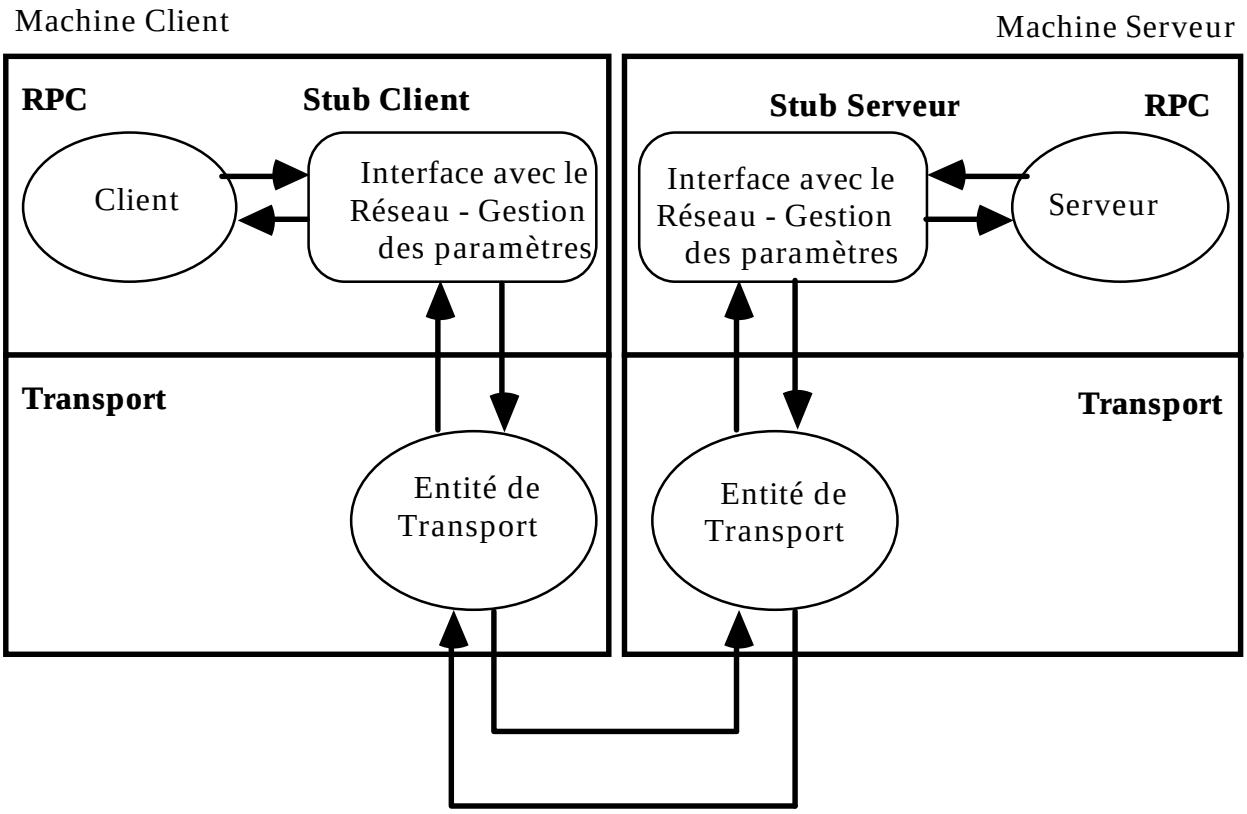


Objet actif : la méthode est exécutée par un programme gestionnaire dans le contexte d'un serveur, modèle d'interaction de type client-serveur

Objet passif : la méthode est exécutée dans le contexte de l'appelant

Appel de Procédure à Distance - RPC

En univers réparti, l'invocation se fait à distance
-> appel de procédure à distance



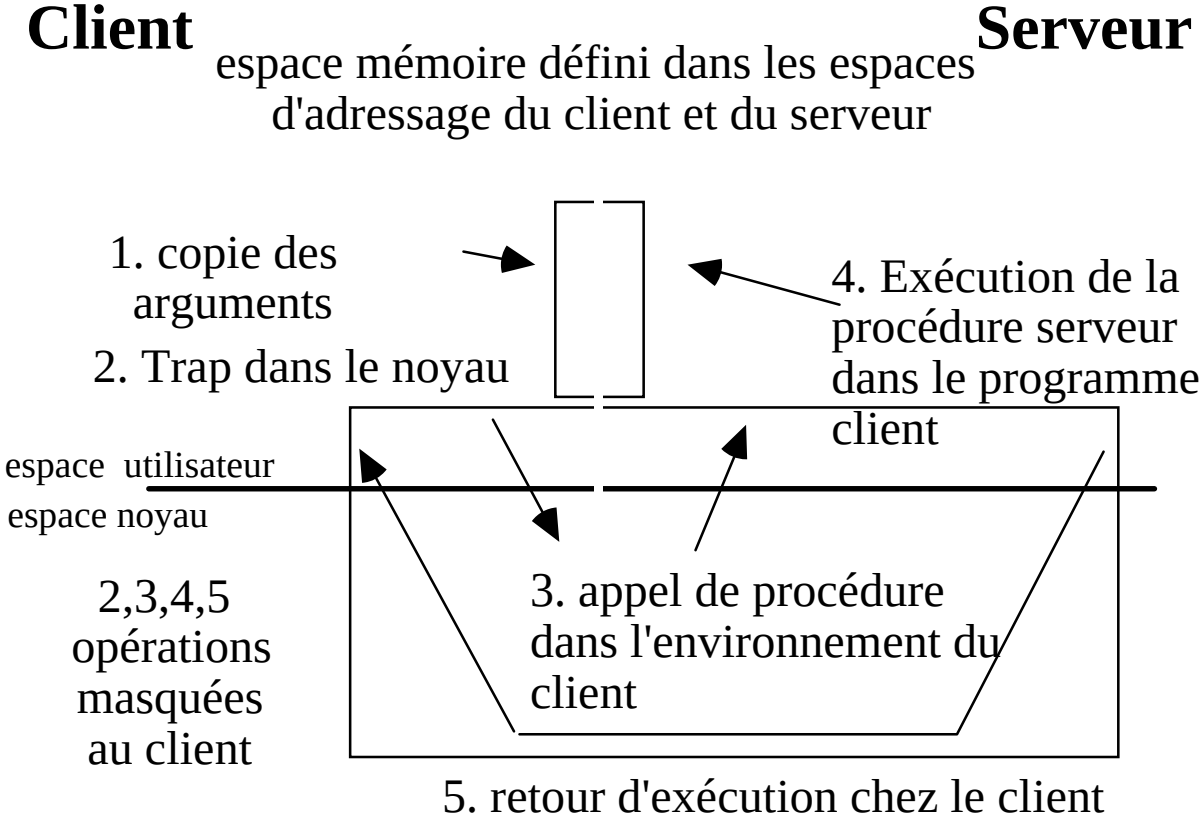
"**Full Weight RPC**"

appel de procédure à distance original, identique à celui définit par Birrell et Nelson

Appel de Procédure Local ("Light Weight RPC")

Certains appels ne partent pas sur le réseau :

il est possible de faire l'économie des phases d'encapsulation des données, le nombre de copies de buffers et de diminuer le nombre de changements de contexte lors de l'ordonnancement[4].



Enregistrement des procédures du serveur auprès d'une entité système "clerc"

[4] Le temps d'exécution serait diminué par 3 quand on compare les deux méthodes en local d'après Bershal (travail sur RPC Firefly).

Problèmes à résoudre dans les systèmes répartis à objets

Plus généralement il faut résoudre :

. désignation/référence

. contrôle d'accès

. granularité des containers d'objets : grain fin, grain moyen, gros grain

. persistance

. migration

. fragmentation

. partage des objets

. **récupération des objets qui ne sont plus référencés (glanage / ramasse-miettes / GC)**

Tendance pour les systèmes à objets répartis

en univers réparti se pose le problème de l'**interopérabilité**

-> prise en compte de l'**hétérogénéité** des produits/architectures qui permettent aux applications de coopérer

standards : OMG -CORBA

OMG Object Management Group
CORBA Common Object Request Broker Architecture

exemples de produits : DSOM[5] (IBM), COOL[6] (Chorus)

[5] Distributed System Object Model
[6] Chorus Object Oriented Layer

OMG - CORBA

CORBA :

. IDL : Interface Data Language pour décrire les données impliquées dans les appels de procédure distant qui est compatible avec les langages de programmation, le C++ en particulier

. Un générateur de souche/stub

. Une architecture qui permet l'édition de liens dynamique entre clients et objets dont la méthode est invoquée
qui permet le mariage des concepts orientés objets et de l'hétérogénéité

Systèmes à Objets et Service de Mémoire Répartie Partagée

L'utilisation d'une mémoire répartie partagée permet de revisiter l'architecture des Systèmes à Objets Répartis :

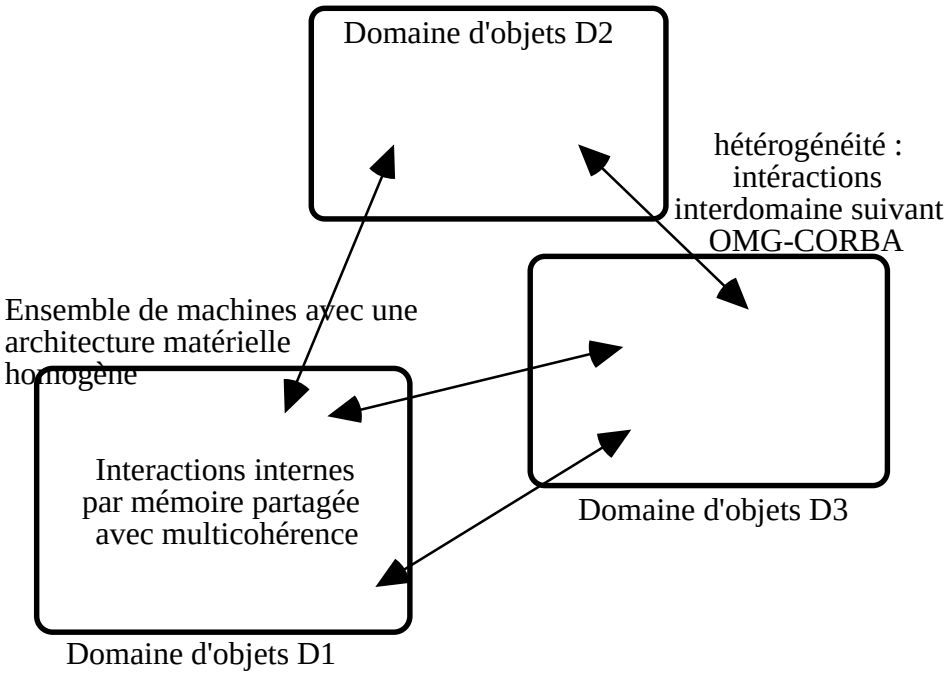
. Projet Larchant : INRIA/SOR

. Projet Arias : IMAG

. Projet Isatis : IRISA

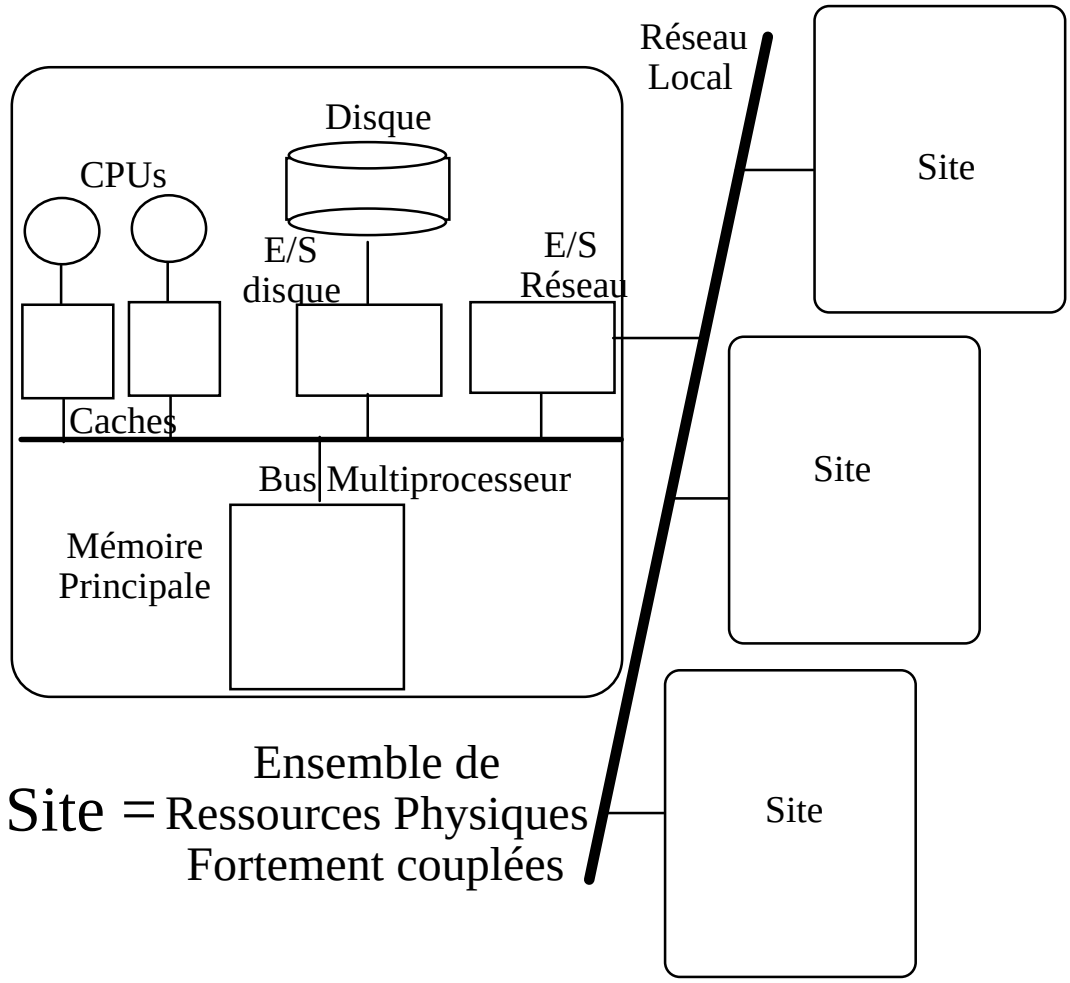
. Projet Saffres : CEDRIC

Plus difficile de gérer l'hétérogénéité :



ABSTRACTIONS DE BASE

Environnement



Ensemble de sites faiblement couplés

Objets Elémentaires

. Noms

. Acteur (Chorus) / Tâche (OSF1) /Processus (Amoeba)

. Activité / Thread / Processus Léger

. Message

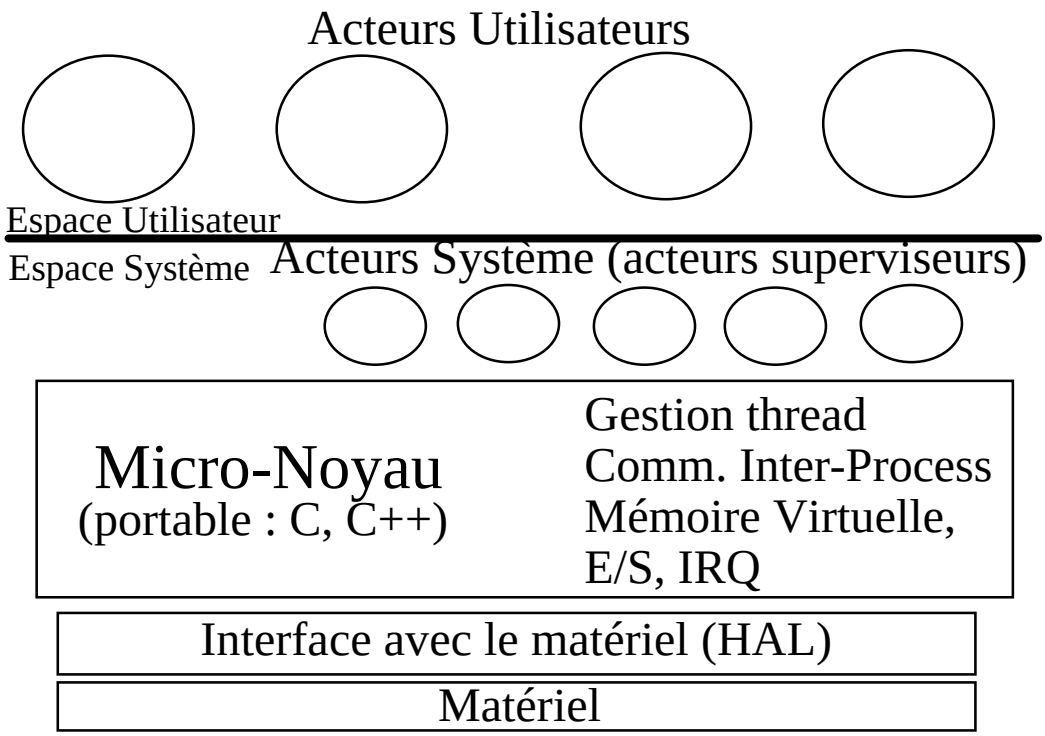
. Porte

. Segment

. Région

Ces abstractions sont combinées et enrichies par les sous-systèmes.

Architecture d'un Système Réparti



. Micro-noyau (minimal) sur toutes les machines

L'ordonnancement (stratégies) doit-il être dans le micro-noyau ? Nouvelle version du micro-noyau CHORUS (v3.6) sort une partie de l'ordonnancement !

. Acteurs Systèmes sur certaines machines en fonction des ressouces présentes et des services à rendre.

Noms (1)

Noms Internes

Noms Uniques et Globaux : **UID**

- Unicité dans le temps : date + nombre aléatoire

- Unicité dans l'espace : contient le nom du site de création de l'objet, parfois son type

permet de référencer les objets élémentaires **indépendamment de leur localisation**, (certains objets peuvent migrer)

Noms (2)

Capacité

Noms intermédiaires :

- connus hors du noyau

- gérés par un serveur

- n'ont de sens que dans le contexte du serveur associé

- liés aux droits d'accès à un objet

Composés :

- nom interne d'un serveur
- clef qui permet d'accèder à la ressource chez le serveur
- protection (droits d'accès)

concept principalement issu d'AMOEBA

Noms (3)

Nom contextuel ou Identificateur Local

Nom qui n'a de sens que par rapport à une entité système.

Equivalent au n° de fichier ouvert pour un processus Unix.

Exemple :

N° de Thread au sein d'un acteur CHORUS

Acteurs/Tâches/Processus (1)

Unité de Structuration =
Acteur/Tâche/Processus

Espace d'allocation de ressources :

. Mémoire : Espace d'adressage (protégé)

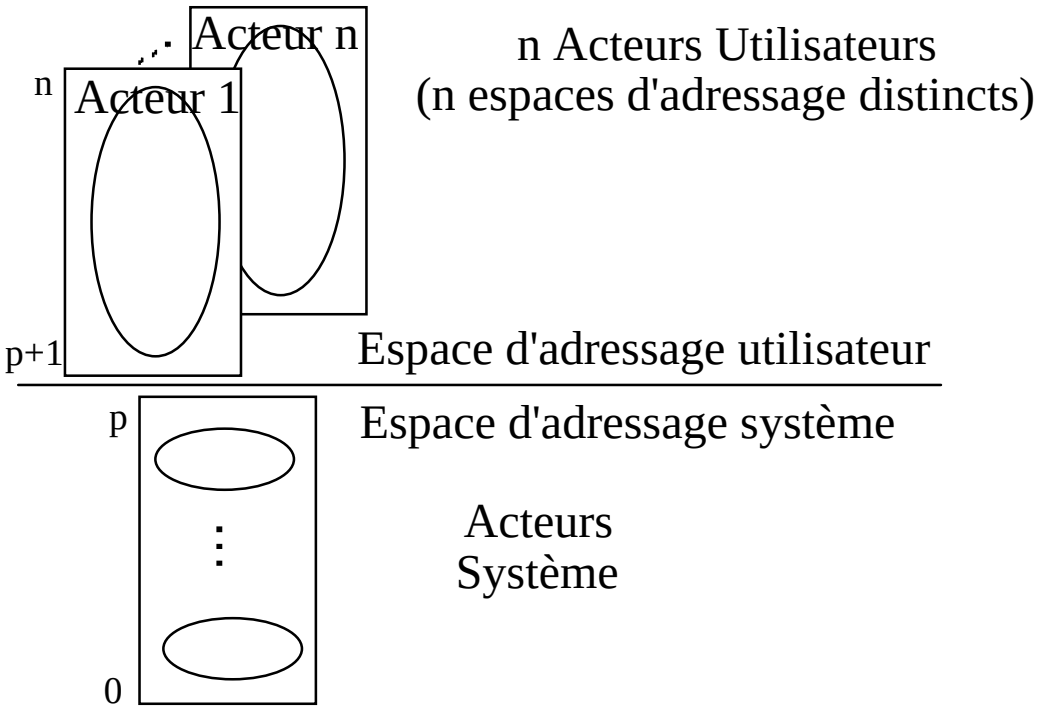
. Ressources système : ports, sémaphores, canaux d'E/S, ...

. Activités

On retrouve le modèle UNIX du processus mais affiné

Pour Chorus, un acteur est une machine virtuelle.

Acteurs/Tâches/Processus (2)



Thread

Unité d'exécution =
Thread/Processus Léger/Activité

. Son exécution a un caractère **Séquentiel**

. Contexte d'exécution :
- état
- compteur ordinal
- registres
- pile d'exécution
- descripteur

. Lié à un acteur

. Partage l'espace d'adressage et les ressources d'un acteur avec d'autres threads

Threads

Ordonnancement

Thread = Unité d'ordonnancement indépendante pour le noyau

Parallèlisme

pseudo-parallèlisme
ou
parallèlisme réel sur multiprocesseur[7]

Que se passe-t-il de différent entre les deux situations suivantes?

. commutation de contexte entre deux threads d'un même acteur

. commutation de contexte entre deux threads d'un acteur différent

[7] **Symétrique** : les processeurs ne sont pas distingués, ils ont tous les mêmes capacités d'exécution,
Asymétrique : les processeurs sont distingués, certains ont des capacités différentes, un co-processeur arithmétique par exemple, et certaines threads doivent s'exécuter exclusivement sur ces processeurs

Acteurs Chorus (1)

Création *actorCreate* **:** Comme lors d'un fork Unix, l'acteur créé (fils) peut hériter d'un certain nombre d'attributs de l'acteur créateur (père).

Acteur est créé avec un minimum de ressources:
- une porte
- pas d'activité
- possibilité d'hériter d'une partie de l'espace mémoire de l'acteur père (régions)

Désignation d'un acteur : Un acteur est désigné par une capacité. *actorSelf()* fournit la capacité de l'acteur courant.

Modification: Un acteur qui reçoit l'identification d'un autre peut intervenir sur celui-ci, et peut modifier sa structure :
- création/destruction d'activités,
- modification de l'espace d'adressage.

Un acteur ne migre pas d'un site à un autre.

Acteurs Chorus (2)

Deux types d'acteurs :

- Utilisateurs :

. Utilisateur sans aucun privilège

. Serveur Système accède à certaines primitives du système (équivalent au statut d'un "daemon" Unix)

- Superviseurs : leurs activités s'exécutent en mode système dans l'espace d'adresse du noyau

=> pb graves en cas d'erreur de programmation

Threads Chorus (1)

Thread :

. Liée à un et un seul acteur

. Une thread est créée par une thread éventuellement dans un acteur différent

. Sur un multiprocesseur, les threads d'un même acteur peuvent s'exécuter en même temps de façon concurrente

. Unité d'ordonnancement considérée par le noyau

Threads Chorus (2)

Thread :

. **Classe d'ordonnancement**

. Création : même classe que le créateur

. Possibilité de changer de classe d'ordonnancement par *threadScheduler()*

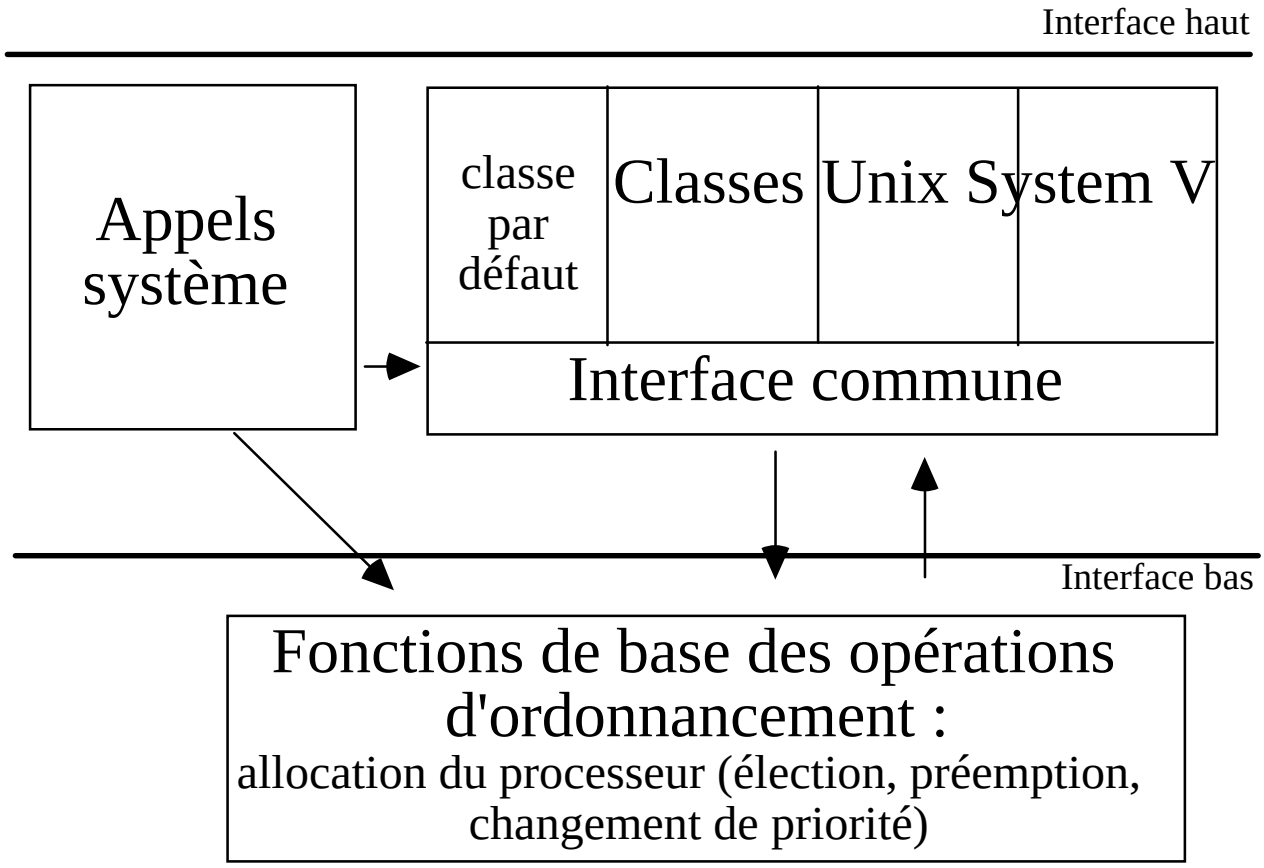
. Priorité : la priorité d'une thread n'a de sens qu'au sein d'une classe d'ordonnancement

. Une partie du contexte du thread est fournie à la création :

- compteur ordinal
- le pointeur de pile (mais pile elle même non allouée)
- le mode d'exécution (système ou utilisateur)

Ordonnancement Chorus (1)

Structure de l'ordonnanceur :

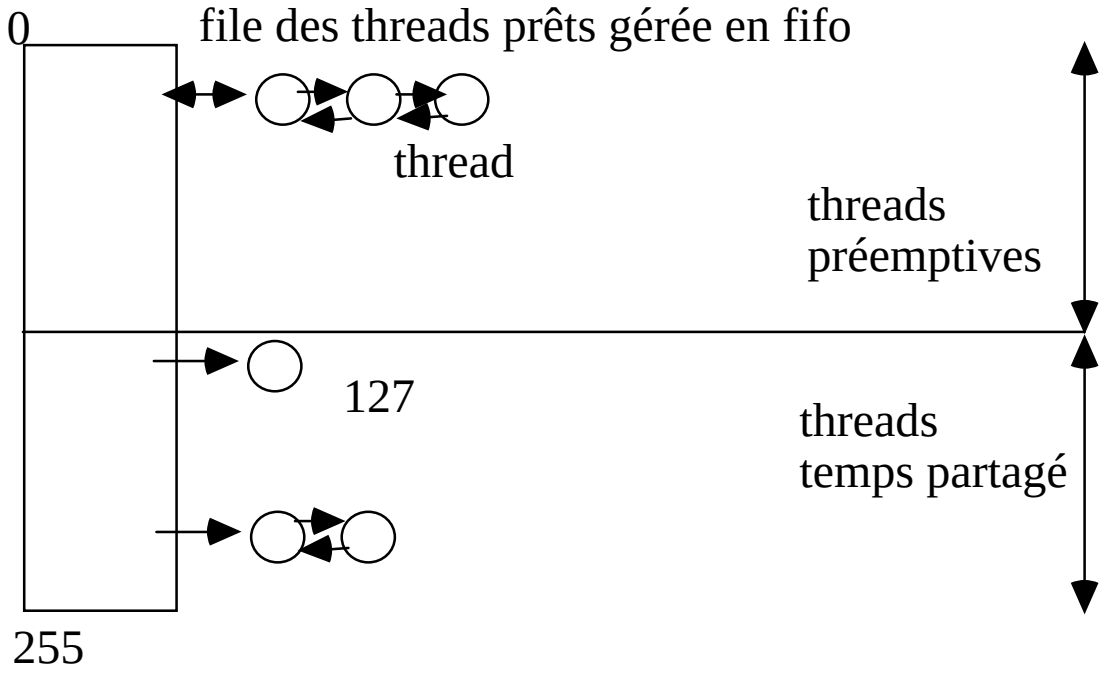


Ordonnancement Chorus (2)

Le noyau ordonnance les threads suivant une priorité à valeur numérique, ***priorité noyau*** :

classe par défaut :

0 plus forte priorité
255 plus faible priorité



Sur un multiprocesseur symétrique à mémoire partagée de N processeurs, le noyau garantit que les N threads élues sont celles qui sont prêtes et de plus forte priorité.

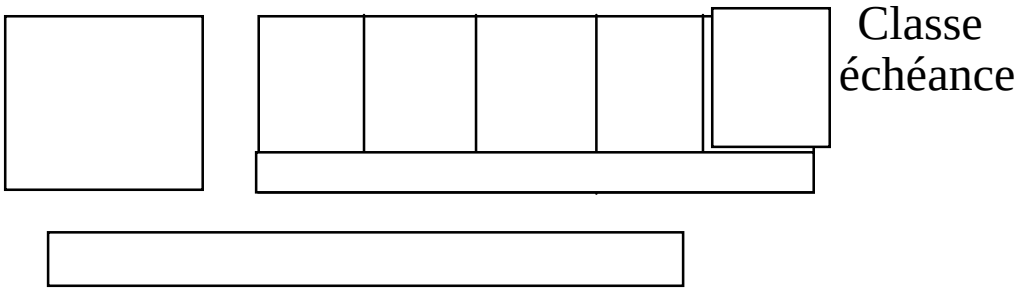
Le noyau est préemptif => **temps réel**

Classes d'ordonnancement

Modularité :

Pouvoir implanter simultanément plusieurs politiques d'ordonnancement

-> une classe par défaut
-> une classe pour le sous-système Unix
-> autre : une classe à échéance par exemple :



travail O. Gaultier et O. Métais

Classe par défaut :

. priorité relative : thread relative à l'acteur
. priorité absolue : thread + acteur

=> priorité noyau

Coopération entre Threads

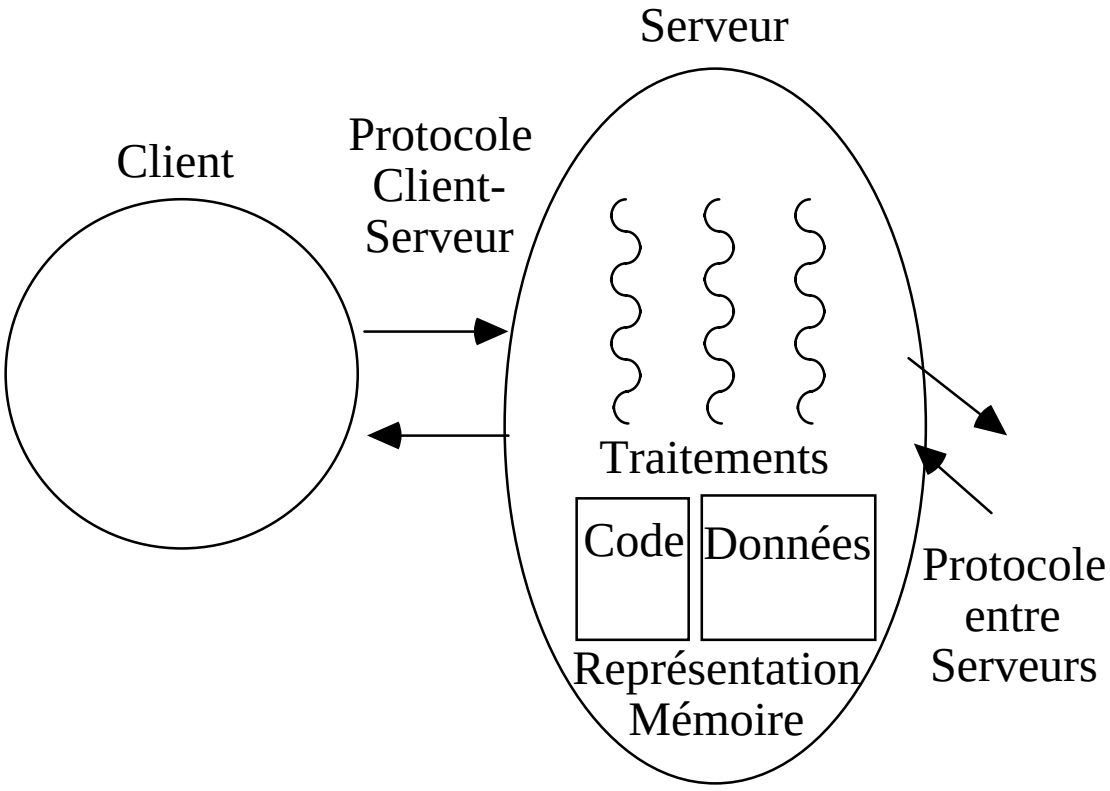
Entre threads d'un même acteur

=> par partage de mémoire

Entre acteurs
=> par messages, en utilisant des portes

Communication Client-Serveur en local

Threads => possibilité de traiter plusieurs requêtes simultanément chez un serveur



Messages

Unité d'échange entre acteurs = Message

Suite d'octets

Non typé dans Chorus, Amoeba, Windows NT

Typé dans Mach en fonction de la nature des données qu'il contient
(plusieurs données dans un message => plusieurs types[8] coexistent dans un message)

[8] Il ne faut pas entendre type au sens du typage des données dans les langages de programmation, d'ASN1 ou de XDR, mais plutôt nature des informations pour la gestion du système.

Portes

Unité d'adressage = Porte

Porte :

- Destination d'un message

- File de messages

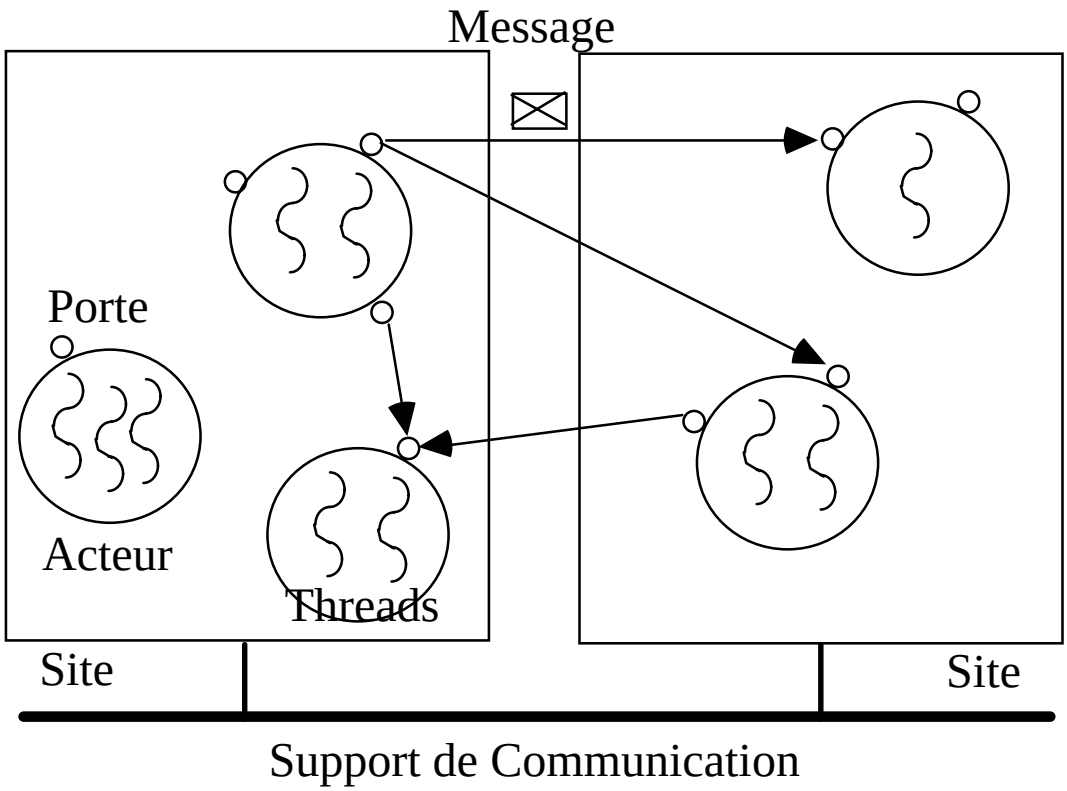
. Droits d'utilisation : [Emission[9]], Réception

. Porte liée à un acteur : seules les activités de cet acteur possèdent le droit de réception
. Droit de réception transmissible (migration de porte vers un autre acteur)

. Droit d'émission transmissible et partageable

[9] Portes unidirectionnelles dans Mach.

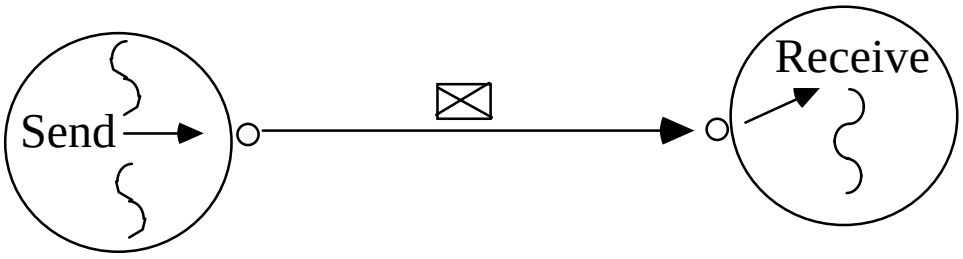
IPC - Communication entre acteurs



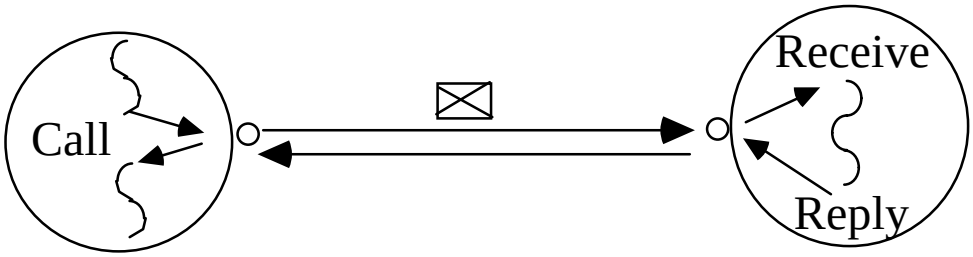
IPC = Inter-Process Communication

IPC

IPC non bloquant/asynchrone :



IPC synchrone :



IPC sans threads dans CHORUS

Programmation par **interruption logicielle** (mécanisme de"message handler") :

On peut associer un traitement à une porte. On connecte un service à une porte.

Dans ce cas, toute réception de message provoque l'exécution du traitement :

- l'ipc est local (LRPC), l'appelant exécute le traitement chez lui

- l'ipc est distant vers un serveur (FRPC), le noyau du site distant dispose d'un "pool de threads", une de ces threads est utilisée quand le serveur est sollicité pour traiter une demande.

Ce mode de traitement est restreint aux acteurs superviseurs et est incompatible avec le mode normal.

Diffusion sur groupe dans Chorus

Groupe de portes

Un groupe de portes est désigné par une capacité

2 modes d'adressage pour l'émission des messages :

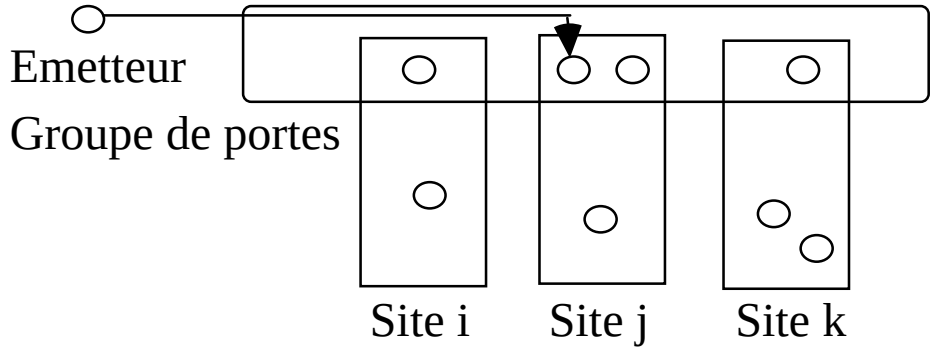
- Diffusion : toutes les portes d'un groupe reçoivent le message

- Fonctionnel : une seule porte reçoit le message

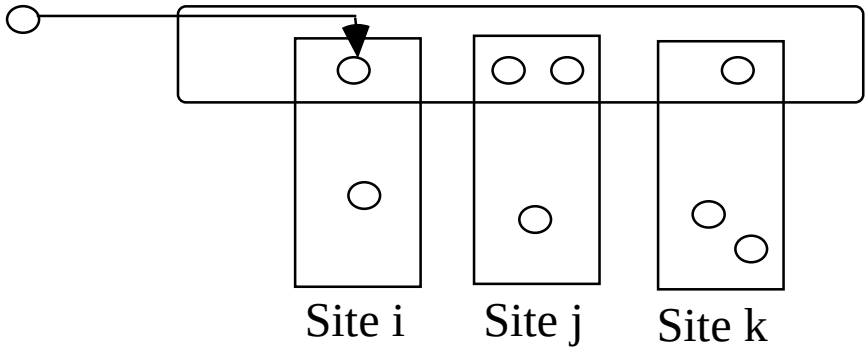
L'adressage fonctionnel peut être avec ou sans sélection du destinataire. Cette sélection est effectuée par l'émetteur sous la forme

Adressage Fonctionnel dans Chorus

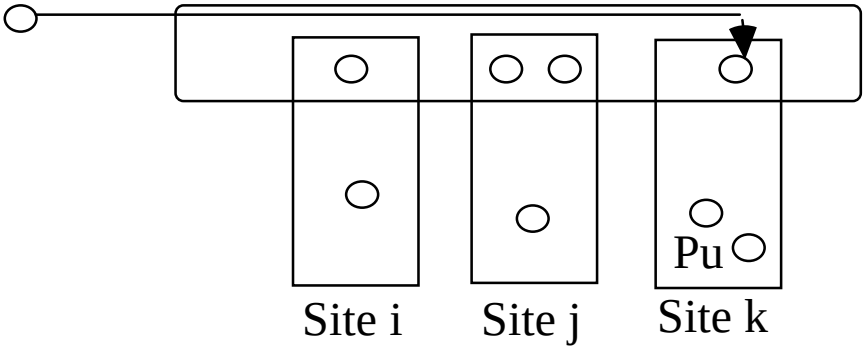
- Adressage Fonctionnel sans sélection : une dans le groupe (idem pour Mach)



- Adressage Fonctionnel avec sélection sur site i



- Adressage Fonctionnel avec sélection sur même site que porte Pu (Pu peut ne pas appartenir au groupe)



Protocoles

Communications locales

=> éviter les recopies (RPC Léger)

Communication à l'intérieur d'un domaine

- une première couche au dessus du réseau avec un protocole de type IP (mode datagramme)

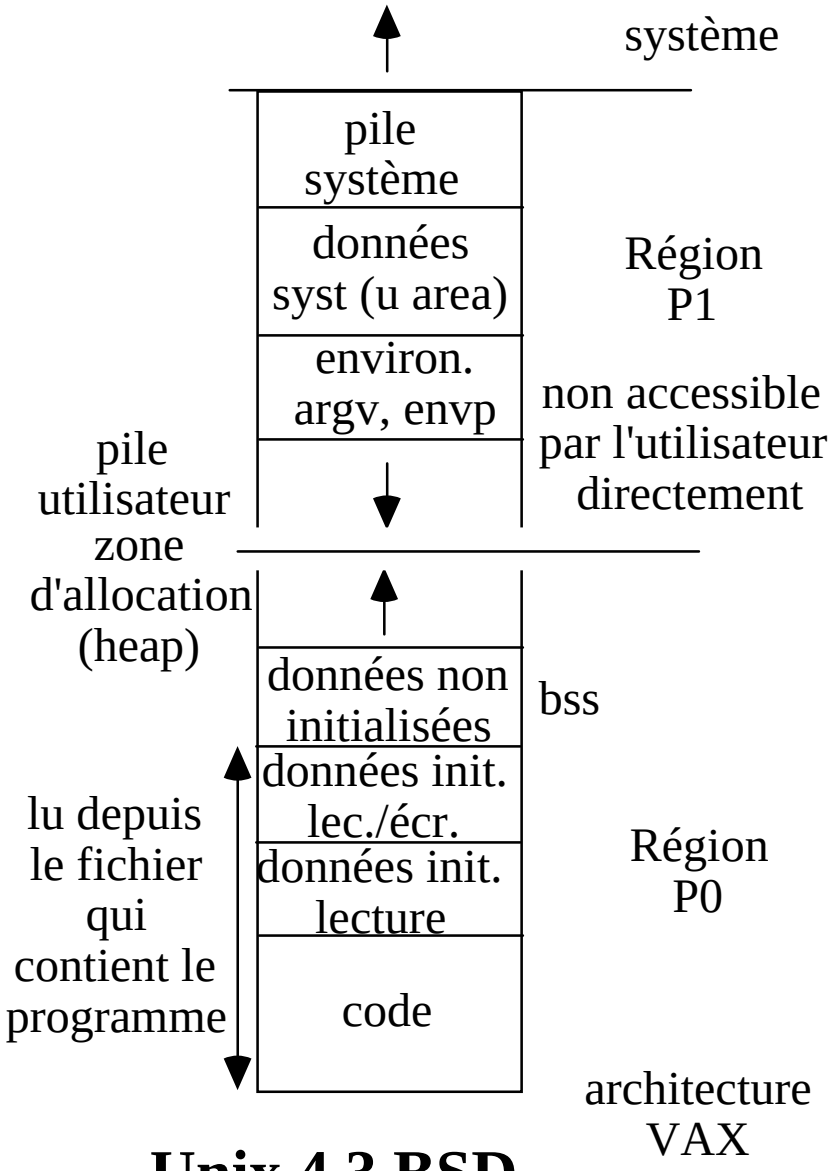
- une seconde couche qui joue le rôle d'une couche transport ou session

Communications extérieures

=> serveurs spécialisés qui sont équipés de piles de communication plus complexes : TCP-IP, OSI, ...

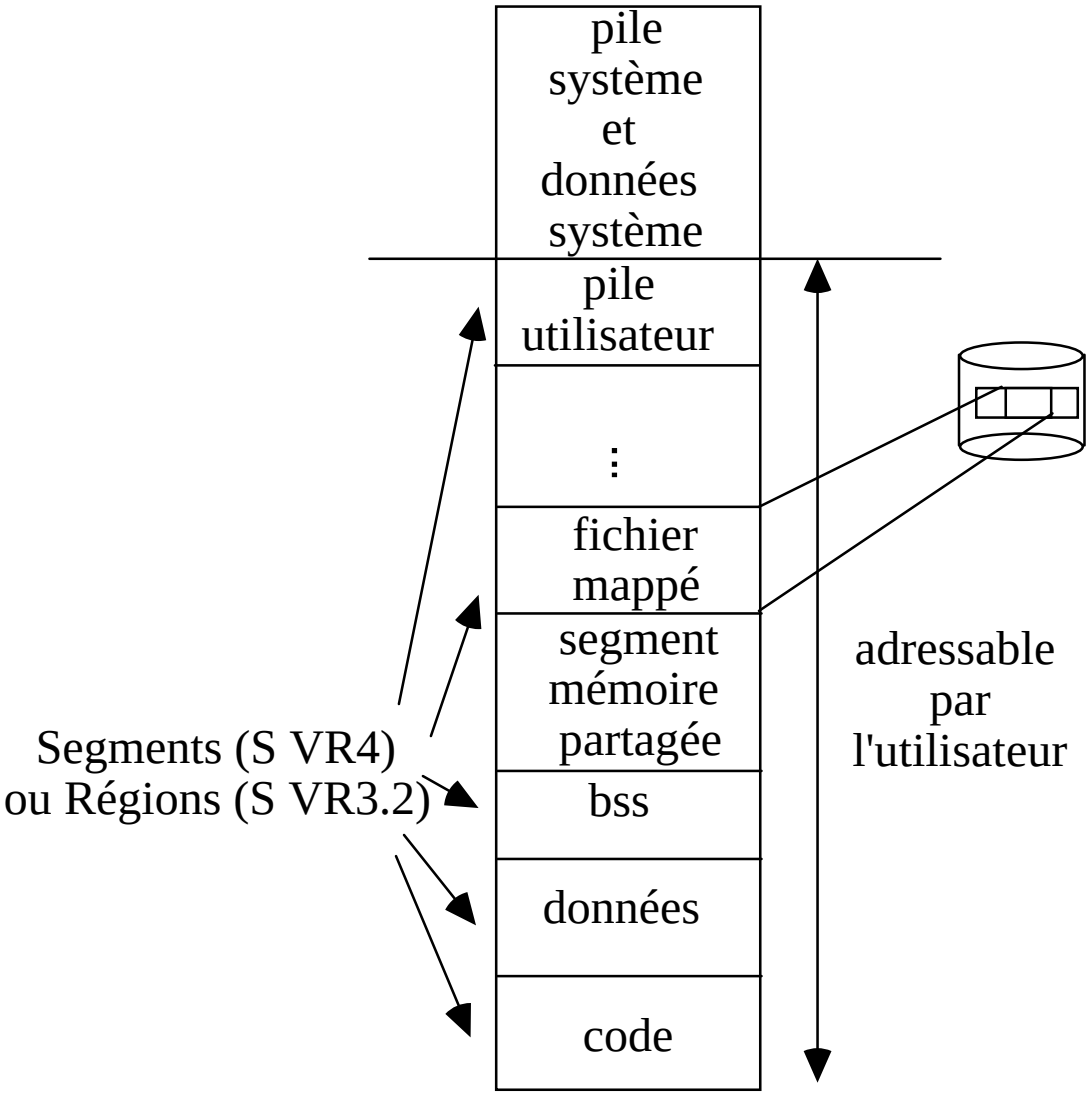
Evolution du Modèle d'Espace d'adressage (1)

4 Go d'espace mémoire : adressage 32 bits (dont 1 gaché puisque non utilisé)



Unix 4.3 BSD

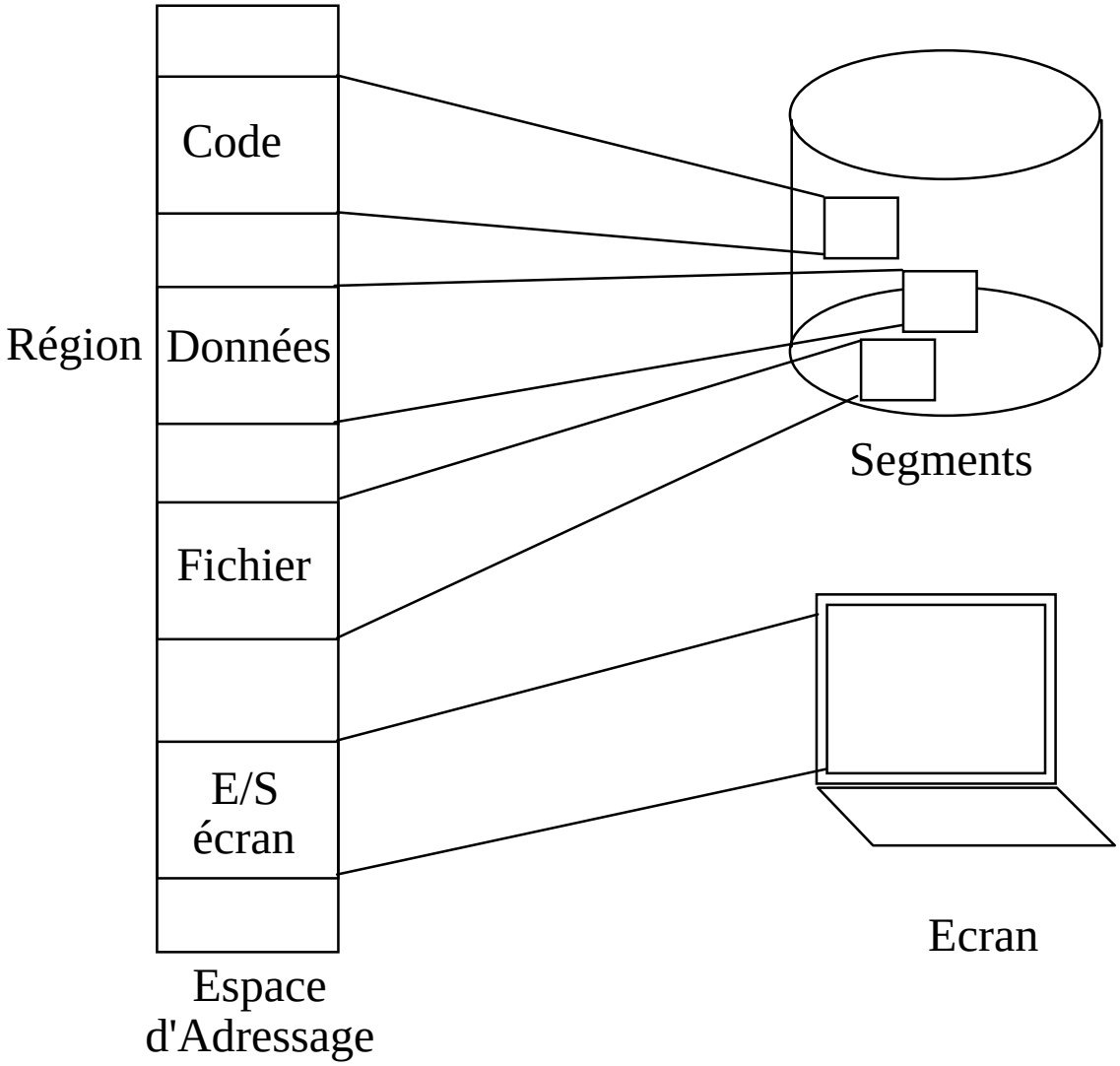
Evolution du Modèle d'Espace d'adressage (2)



Unix System V R4

Objets mémoire

Unité de représentation des Données = **Segment**
Unité d'accès aux données en mémoire = **Région**



Segments et Régions

Segment :

- Données permanentes (fichier) ou temporaires (espace swap)

- Géré par serveur externe au noyau (mappeur)

- Identifié par une capacité

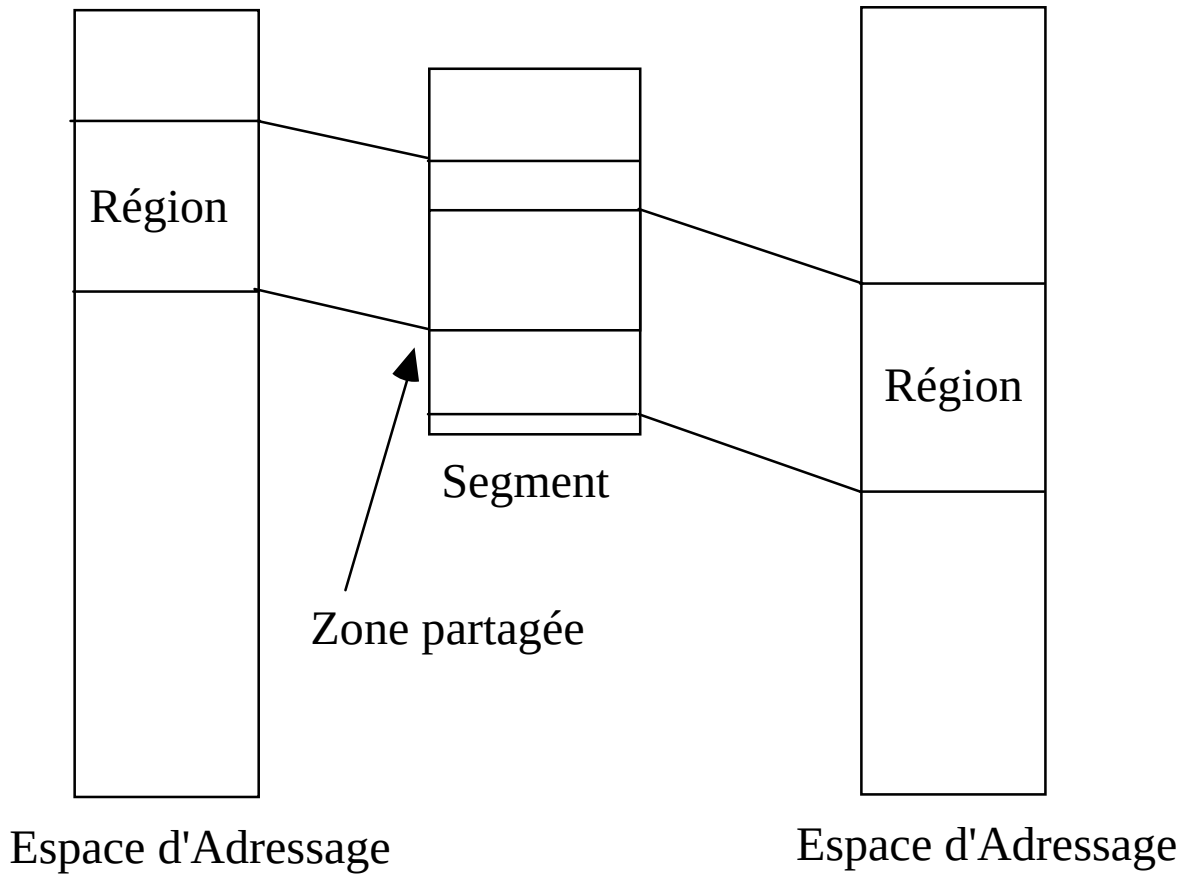
Région :

- L'espace d'adressage d'un acteur est divisé en plusieurs régions

- Partie visible en mémoire d'un segment : mise en correspondance d'un segment dans l'espace d'adressage d'un acteur

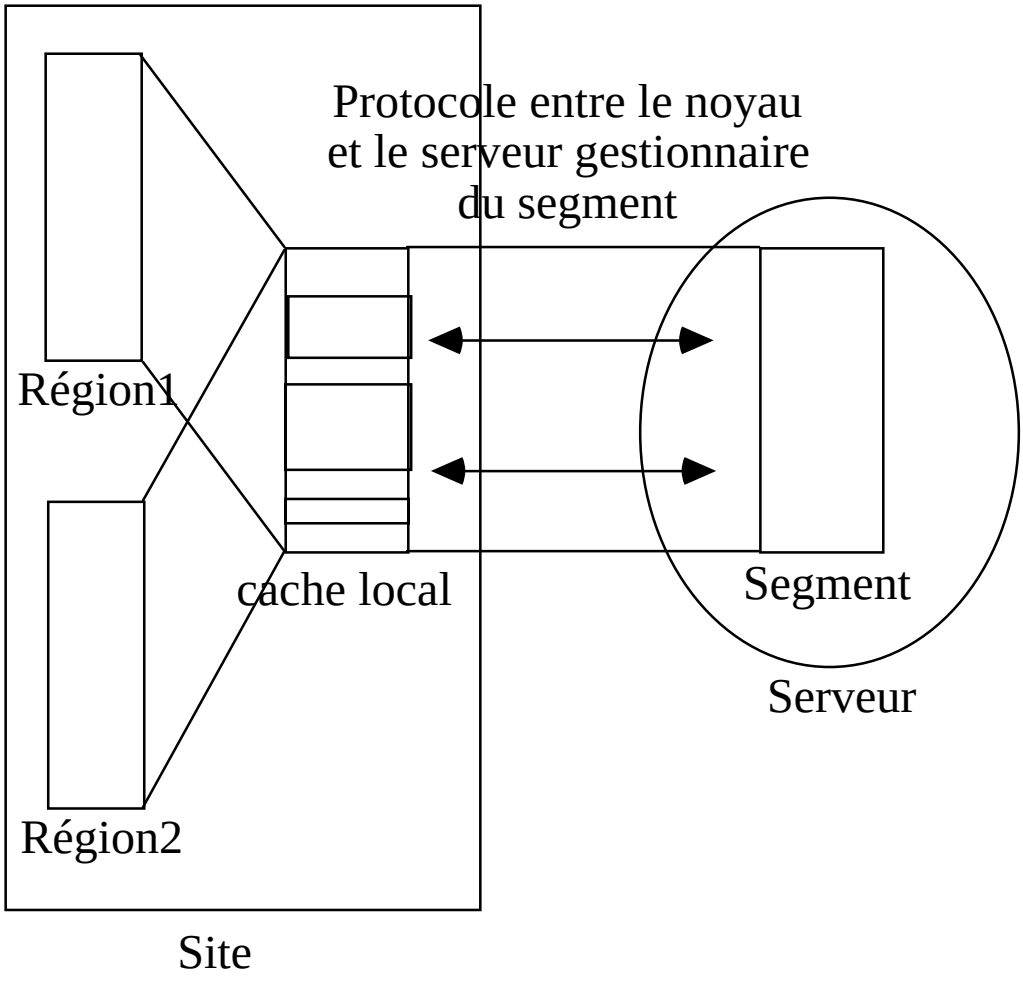
Partage

Partage d'un segment entre acteurs par région



Cache des segments

Le noyau peut gérer un cache des pages virtuelles associées au segment.



-> Pas plus d'un cache par segment et par site
-> Gérer la cohérence du segment entre sites

CHORUS

. Projet INRIA 1981 - 1986 (V1, V2)

. Chorus Systèmes 1987 - (V3)

. Micro-Noyau

. Unix Sous-système : System V R4, 4.3BSD, SCO

. Sous-système orienté objet : COOL

. Architectures faiblement couplées

. Architectures fortement couplées à processeur symétrique

Références Bibliographiques

[1] Construction des systèmes d'exploitation répartis. Editeurs : R. Balter, J-P Banâtre, S. Krakowiak. Inria. Collection Didactique. 1991

[2] Supports de cours : Technologie des micro-noyaux.. Marc Rozier. Ecole d'Eté d'Autran. Septembre 1993.

[3] Distributed Operating Systems. Andrew S. Tannenbaum. Prentice Hall. 1994.

[4] Supporting an Object-oriented distributed system : experience with Unix, Mach and Chorus. Boyer, J. Cayuela, P-Y. Chevalier, A. Freyssinet, D. Hagimont. Rapport Technique Bull-Imag. N 7-90. Décembre 1990.

[5] Inside Windows NT. Helen Custer. Microsoft Press. 1992.

[6] Mach 3 Kernel Principles. Keith Loepere. Open software Foundation and Carnegie Mellon University. NORMA-MK12: July 15, 1992.

[7] Chorus/Mix V.4 : Programmers Guide & Implementation Guide. Chorus Système. 1992.

[8] Chorus Kernel V3 : Programmers Guide & Implementation Guide. Chorus Système. 1993.

[9] Distributed Systems : concepts and Design. George Coulouris, Jean Dollimore, Tim Kindberg. Addison Wesley 1994.

[10] Chorus Jam Session 1994. Proceedings. Septembre 1994.

[11] Advanced Operating Systems. Dossier Spécial. V19 N1. Byte. January 1994.

[12] Projet MaX : Construction d'un système d'exploitation basé sur le micro-noyau Mach. Daniel Azuelos. Mémoire d'ingénieur Cnam. Juin 1994.

[13] Using CHORUS to Develop Real Time Applications. Chorus System. Chorus Real Time Tutorial. Juillet 1994.

[14] Mise en place d'une plate-forme Chorus, Conception et Implantation d'un Ordonnanceur à échéance au sein du noyau Chorus. O. Gaultier, O. Métais. Mémoires d'ingénieur Cnam. Mars 1994.

[15] Distributed Systems : concepts and Design. George Coulouris, Jean Dollimore, Tim Kindberg. Addison Wesley 1994.

Le développement de protocoles de communication en mode message asynchrone

G. Florin

Le contrôle réparti

Ensemble des mécanismes offerts à un programmeur pour développer des applications réparties

Utilisation de l'interface d'un système réparti.

IPA "Interface de programmation d'applications"

API "Application Programming Interface"

Problèmes posés:

-La concurrence, la synchronisation des activités locales

-Les communications entre sites

(les interactions)

Différents styles d'interactions

Schéma de complexité croissante:

Le mode message asynchrone

Le mode rendez-vous

Le mode appel de procédure distante

Le mode mémoire virtuelle répartie

Les protocoles coopératifs

...

Construction d'API unifiant l'approche objet et les interactions:

Les systèmes d'objets répartis

Première partie

Le mode message asynchrone

Rappel de quelques points essentiels

Généralités

- Mode de base de la communication.

- Offert par le plus grand nombre des IPA réparties

- Le service comprend deux primitives principales pour communiquer et se synchroniser (couches transport).

TYPE COM_MESSAGE_ASYNCHRONE;

METHOD envoyer (id_émetteur, id_récepteur, compléments, message);
METHOD recevoir (id_émetteur, id_récepteur, compléments, message);
METHOD ...;

END COM_MESSAGE_ASYNCHRONE.

identificateurs : ports
compléments: qualités de services (selon des sémantiques multiples)
messages: zones de données.

Exemples d'IPA distribuées en mode message asynchrone

Presque toutes les interfaces des architectures "ouvertes" de réseaux

(pour presque tous les niveaux)

- **Internet, OSI, SNA**

Presque toutes les interfaces des systèmes répartis classiques,

- **Chorus , Mach, Amoeba**

De nombreux langages de programmations

- **Langages de spécifications de protocoles** (Estelle, ...)

- **Langages "acteurs" de l'intelligence artificielle distribuée** (Act1, ...)

Exceptions : les produits orientés RPC, objet, partage de mémoire.

Variantes sémantiques du mode message asynchrone

- **Propriétés de synchronisation**

  La synchronisation locale

  La synchronisation inter-sites.

- **Nature des entités** communicantes

  point à point, diffusion

  mode alternat, bidirectionnel

- **Propriétés d'ordre**.

  local, global, causal

- **Propriétés de tolérance aux pannes**.

  messages, sites

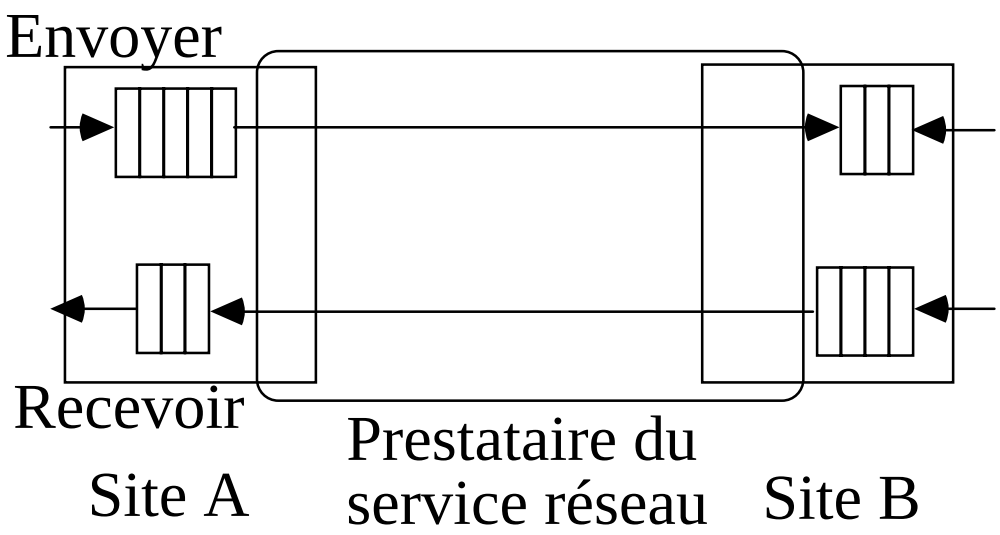
- **Propriétés temporelles**

  débits, délai, variation de délai

"Cours de réseaux habituels"

Les propriétés de synchronisation

Le mode message asynchrone réalise un **"producteur-consommateur" réparti** entre un émetteur et un récepteur.



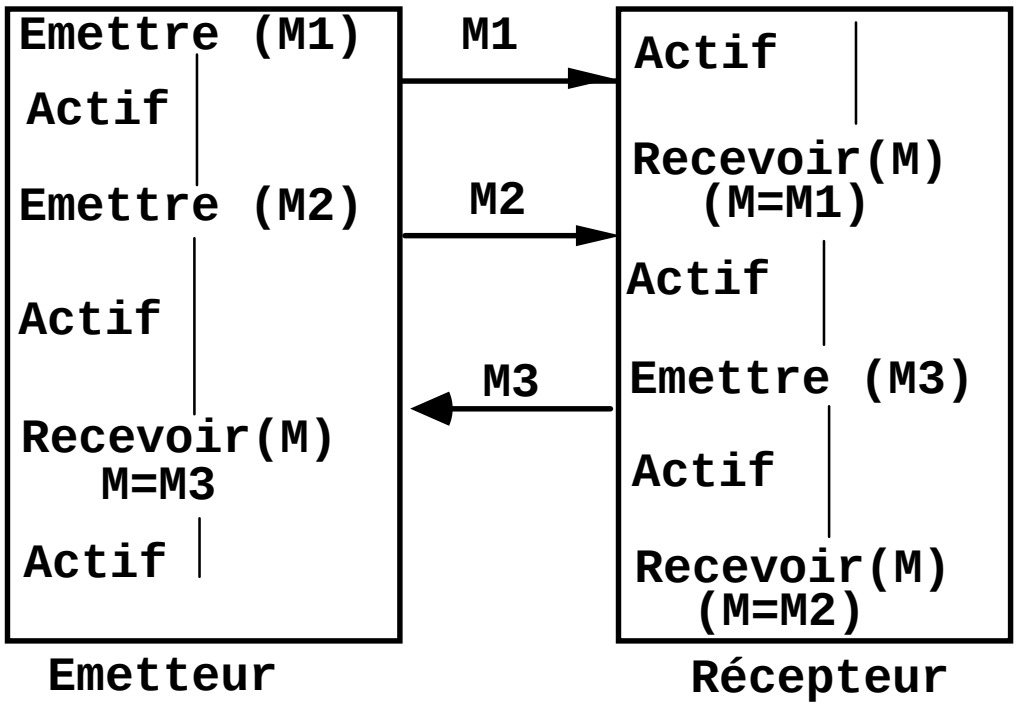
Asynchronisme entre sites distants.

- propagation sur le réseau

Synchronisation locale sur les tampons d'émission et de réception.

A) La synchronisation inter-sites

Aspect principal: l'asynchronisme entre l'émetteur et le récepteur



Détails du comportement

L'émetteur

. Ayant demandé une émission, **reprend la main et continue son exécution *"immédiatement"* après** la prise en compte de sa demande.

. Le message est transmis (au rythme du transport d'informations par le réseau de communication) donc de façon **asynchrone** avec le comportement émetteur**.**

Le récepteur

Ayant décidé de prendre en compte un nouveau message il acquiert un message (le premier) en instance.

-Celui qui se présente après le recevoir

-Un message qui se trouve dans une file d'attente de réception.

B) La synchronisation locale

Synchronisation sur l'interface de programmation d'application.

Primitives envoyer "bloquantes" "non bloquantes"

Le processus émetteur a fourni l'adresse d'un tampon (contenant un message) partagé avec le prestataire.

. Cas 1 :

L'émetteur reste bloqué tant que le message n'a pas été envoyé => le tampon est redevenu libre.

. Cas 2 :

L'émetteur reste bloqué tant que le message n'a pas été recopié dans une file d'attente du prestataire.

. Cas 3 :

L'émetteur reprend la main alors que le prestataire utilise le tampon. Il est averti par un signal de la fin d'utilisation => il peut le réutiliser.

Primitives recevoir "bloquantes" "non bloquantes"

Le processus récepteur a fourni l'adresse d'un tampon au prestataire

. Cas 1 : Recevoir bloquant

Le récepteur reste bloqué tant qu'un message reçu n'a pas été écrit.

. Cas 2 : Recevoir non bloquant + attente

Le récepteur continue son exécution et se bloque quand il ne peut plus ne pas disposer d'un message reçu.

. Cas 3 : Recevoir non bloquant + primitive de test de message

Le récepteur continue mais il peut savoir si un message reçu existe.

. Cas 4 : Réception conditionnelle

Le récepteur reprend toujours la main avec un message reçu ou avec un diagnostic.

L'utilisation du mode message asynchrone

En général deux aspects simultanément réalisés:

Invocation d'action distante

La nature de l'action est définie par le typage du message.

La **nature de l'action déclenchée** dépend du contexte dans lequel le message est pris en compte.

(Idée de contexte ou d'état)

("Acte de langage")

Transport d'informations dans la partie données du message

Action "d'information"

Paramétrage d'exécution distante.

Sémantique de l'activation

L'échange en mode message asynchrone permet de définir des structures de contrôle classiques en univers réparti

a) **Activation en parallèle ("fork")**

D'une activité à distance puisque en mode normal l'émetteur et le récepteur continuent leur exécution après l'interaction envoyer recevoir.

b) **Branchement inconditionnel ("goto")**

Une activité à distance s'exécute après l'activité locale si l'émetteur se suspend définitivement (wait) après l'émission.

Sémantique d'échange de données

- Le mode message permet une opération d'affectation à distance d'une variable M (de type article):

émettre (M) = écrire (M) (vers dest)

recevoir (M) = lire (M) (de emet)

. Avec possibilité de **vérification de cohérence des types** des variables (si type(M) acheminé).

. Avec **une sémantique de consistance des données entre l'émetteur et le récepteur très faible.**

Reposant sur les propriétés d'ordre, temporelles, de tolérance aux pannes spécifiées par la qualité de service.

Propriété minimale de cohérence des échanges de données par message asynchrone: la causalité

émettre/écrire (M)->recevoir/lire(M)

∃ t1 : émetteur.écrire (M)

=> ∃ t2>t1 : recepteur.lire (M)

Les messages ne remontent pas le temps

Propriétés supplémentaires de cohérence (propriétés temporelles et propriétés d'ordre).

mode rendez-vous,

mode appel de procédure

mode mémoire répartie.

Seconde partie

Spécification des applications utilisant le mode message asynchrone

Objectifs

Définir des méthodes formelles de spécification des applications réseaux en mode message asynchrone.

(FDT "Formal Description Techniques")

=> Processus **séquentiels**

=> Communicants par messages

Comportement séquentiel des processus

=> Les méthodes de spécification utilisent des **automates d'état fini.**

=> Chaque entité communicante est représentée par son automate d'état.

Interactions communication par messages

=> Les automates sont synchronisés pour la représentation des échanges de messages.

Représentation des applications: automates

Une modélisation état/transition.

États

Chaque état doit être **aisément interprétable** en terme d'évolution locale de l'application répartie.

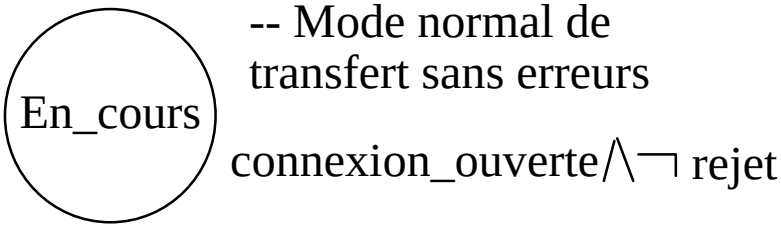
- Il représente un point d'avancement du contrôle.

- Il est défini par **un ensemble significatif de variables locales** de chaque processus. Il peut recevoir:

- un identifiant bien choisi

- un commentaire

- le prédicat sur les variables



Identification des états

Étape préliminaire importante de la spécification:

. Pas un niveau de détail trop fin (illisible, trop grand nombre d'états).

. Pas un niveau de détail trop gros
Tout est modélisé dans les transitions.

Représentation graphique de l'évolution au niveau permettant l'observation de la synchronisation.

Transitions.

Elles comportent deux mentions:

la condition déclenchante ,
l'action à réaliser.

Elles sont représentées par les arcs du graphe associé à l'automate conduisant d'un état initial à un état final.

Conditions ou gardes

Les transitions sont franchissables lorsqu'une condition booléenne ou garde est satisfaite:

. condition booléenne portant sur **les variables locales**.

. condition booléenne portant sur les **messages entrants**.

. condition booléenne portant sur des d'événements internes (**horloges**)

Action

- C'est l'opération réalisée lors du franchissement de la transition.

les traitements à réaliser lors du passage à l'état suivant

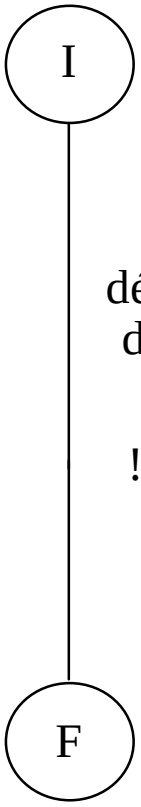
affectation de variables,

envoi de messages.

- Le détail des actions ne doit pas être trop important sinon le modèle est illisible

Sinon renvoyer à des textes annexés à la spécification.

Représentation des transitions.



Prédicat définissant l'état

Etat de contrôle antécédent
Etat initial

?condition: prédicat définissant les conditions de déclenchement de la transition

Partie condition ou garde

! action:comportement associé à la transition

Partie action de la transition

Prédicat définisant les connaissances après la transition

Etat de contrôle conséquent
Progression de l'algorithme

Les commandes d'échanges

- Notation des opérations envoyer, recevoir

<émission> ::= <dest> **!** <message>

<réception> ::= <source> **?** <message>

Désignation

La commande d'émission doit désigner un destinataire (P2).

Ex : Dans le processus P1 :P2!M1

La commande de réception (exécutée par un processus P2) doit évoquer une source P1.

Ex : Dans le processus P2 :P1?M1

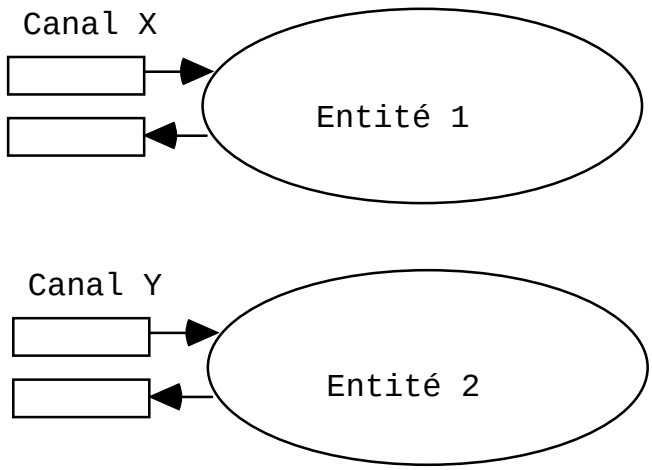
Remarque: Dans le cas d'une communication point à point absence d'ambiguïté => pas de désignation.

Utilisation de canaux

- Très souvent la désignation se fait au moyen de canaux (terminologie des langages de description de protocoles) qui sont ensuite connectés pour réaliser une architecture.

Ex : Émission sur canalX : canalX!M

Réception sur canalY : canalY?M



La définition d'architecture consiste à spécifier ensuite une liaison entre canalX et canalY.

Typage

Il doit y avoir correspondance de type entre le message reçu (la variable de réception) et le message émis (la valeur émise).

Données d'un message =

variable de type article.

Type[message_émis]=

Type [message_reçu]

Si le type d'un message reçu n'est pas prévu à la réception:

Erreur: "réception non spécifiée".

Sémantique de l'alternative constituée par l'existence de plusieurs transitions en un état

- Évaluation des gardes

. Une garde est **franchissable** si elle est constituée par **une expression booléenne** portant sur des variables locales à **valeur vrai**.

. Une garde est **franchissable** s'il s'agit d'une **commande d'échange** satisfaite : en fait une réception dont le processus source à fait parvenir un message du type attendu qui se trouve en tête de la file d'attente.

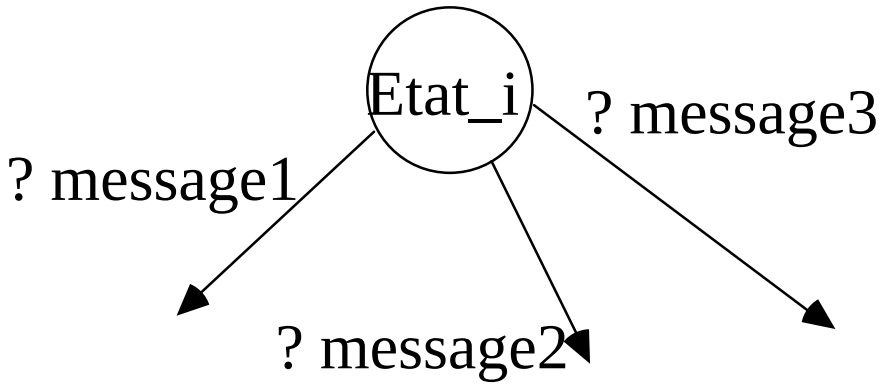
. Une garde **combinaison des deux cas précédents.**

Alternative en un état

Indéterminisme de comportement

Indéterminisme aspect fondamental du réseau.

- en un même état plusieurs messages (ou événements) peuvent arriver.
- l'un des messages est placé en tête de la file de réception.
- ce sont les aléas de fonctionnement des matériels et des logiciels, les choix effectués qui conduisent souvent l'interclassement dans la file d'entrée.



Alternative en un état

Aspect séquentiel des automates

- On choisit **l'une des transitions** ayant sa condition booléenne satisfaite (sa garde ouverte).

Indéterminisme de l'évaluation

- Si plusieurs conditions sont simultanément vérifiées l'une **quelconque des** transitions peut-être sélectionnée

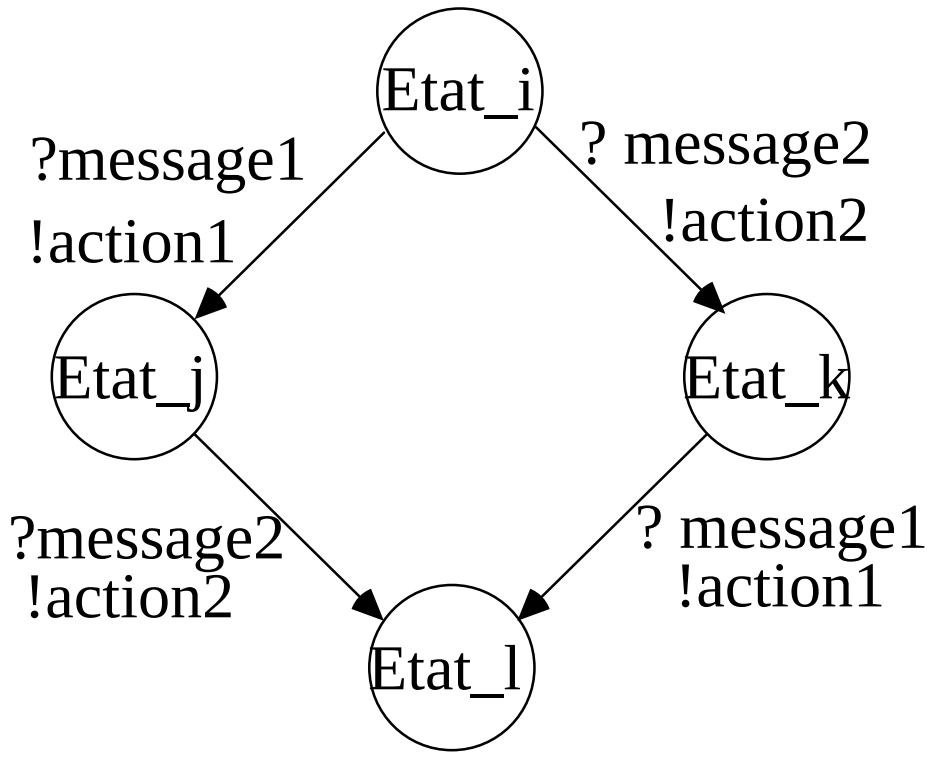
- **La liste des commandes** associée à la garde est **exécutée**.

La modélisation doit tenir compte de cette caractéristique majeure.

Schémas persistants

Si deux messages peuvent être reçu en un état et que leur réception ne modifie pas les conditions de fonctionnement ultérieures

=> leur réception doit continuer à être prévue.



Alternative en un état

Sémantique en cas d'attente

- Aucune garde n'est franchissable mais il existe des gardes avec condition de réception pouvant être satisfaite par une émission d'un site distant.

=> **Attente que l'une d'elles soit satisfaite.**

- Aucune garde n'est franchissable et elles sont toutes booléennes.

=> Fin du programme
Probable **erreur d'exécution.**

- Aucune garde ne pourra plus jamais être franchie (certaines sont des attentes de message).

=> Fin du programme
Erreur: interblocage de message.

Différents automates

Le service

Définit les échanges entre un prestataire (fournisseur) et un utilisateur d'un service.

. Établissement de **la liste des unités de service** (primitives, SDU) échangés entre le prestataire et l'utilisateur (niveau n et niveau n+1).

. **Définition des enchaînements autorisés** de primitives sous la forme d'un automate d'état.

=> Toutes les successions légales qu'un observateur de l'interface n n+1 peut observer.

=> Deux points de vue duaux:

- le prestataire du service
- l'utilisateur du service

Exemple de comportements autorisés dans une entité de liaison: éléments de connexion.

Éléments du service de connexion avec accord confirmé

Connect.request

Connect.indication

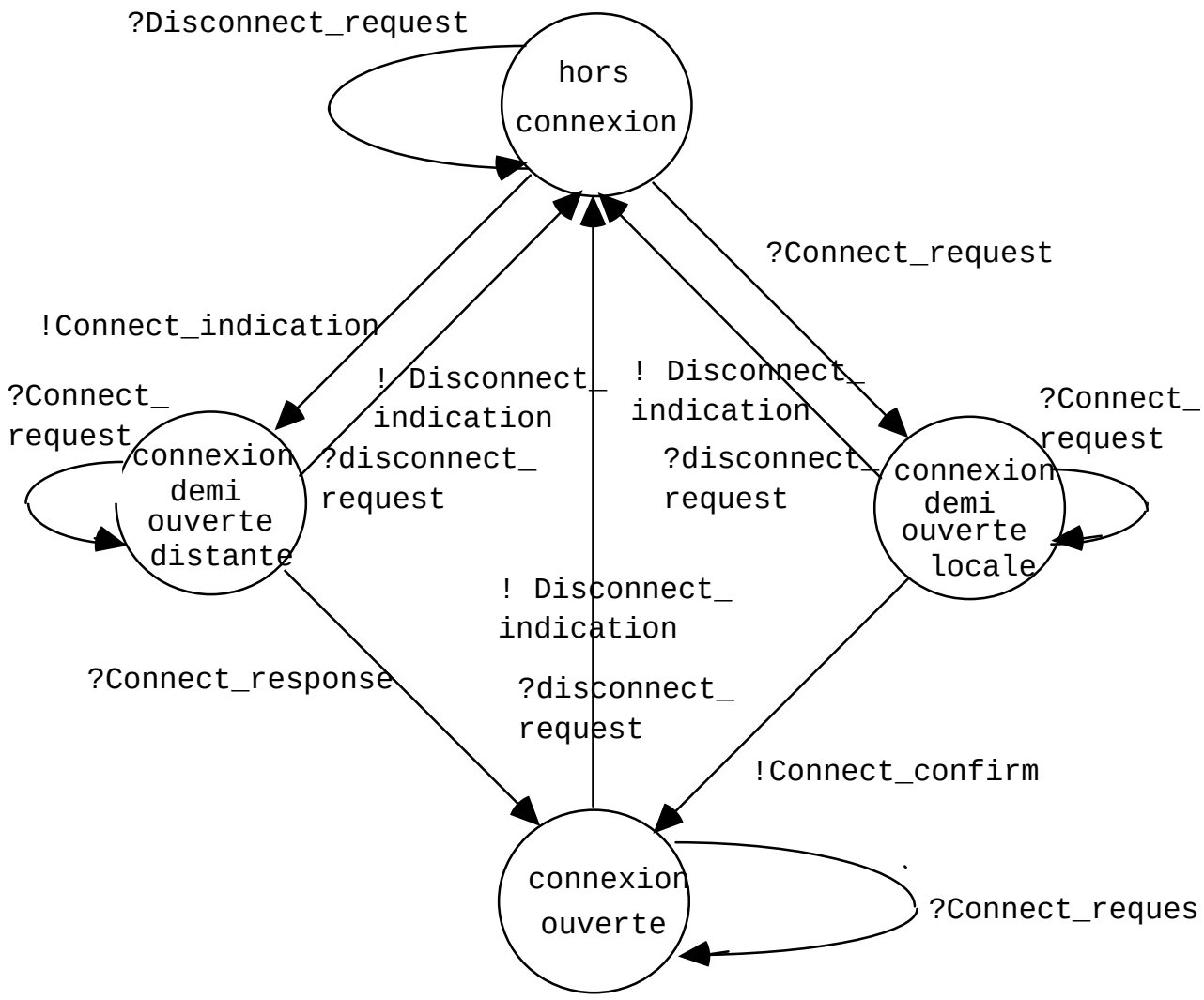
Connect.response (cas d'accord)

ou Disconnect.request (cas de rejet)

Connect.confirmation (accord)

ou Disconnect.indication (rejet)

Automate du service



Le protocole

Définit les échanges entre deux prestataires de même niveau.

. Établissement de la liste des unités de protocole (messages, PDU) échangés entre deux niveaux n.

. Définition des suites d'échanges qu'un observateur de la voie de communication entre les deux niveaux n peut observer.

=> Toutes les successions légales d'éléments de protocole observables sur la voie logique de communication n <--> n.

Éléments du protocole de liaison en phase de connexion

Demande d'ouverture

SABM

"Set Asynchronous Balanced Mode"

Demande de fermeture

DISC

"Disconnect"

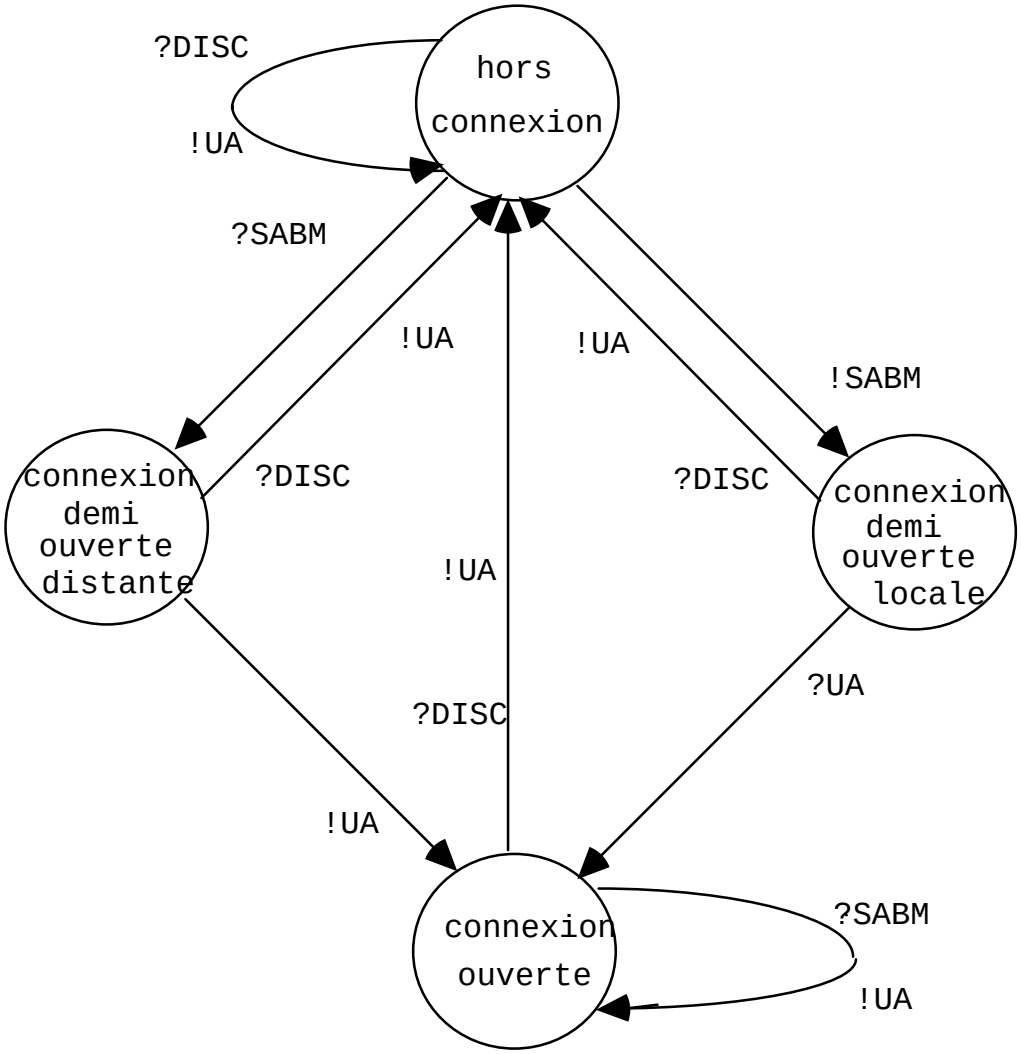
Accusé de réception non numéroté

UA

"Unnumbered Acknowledge"

Automate du protocole de connexion

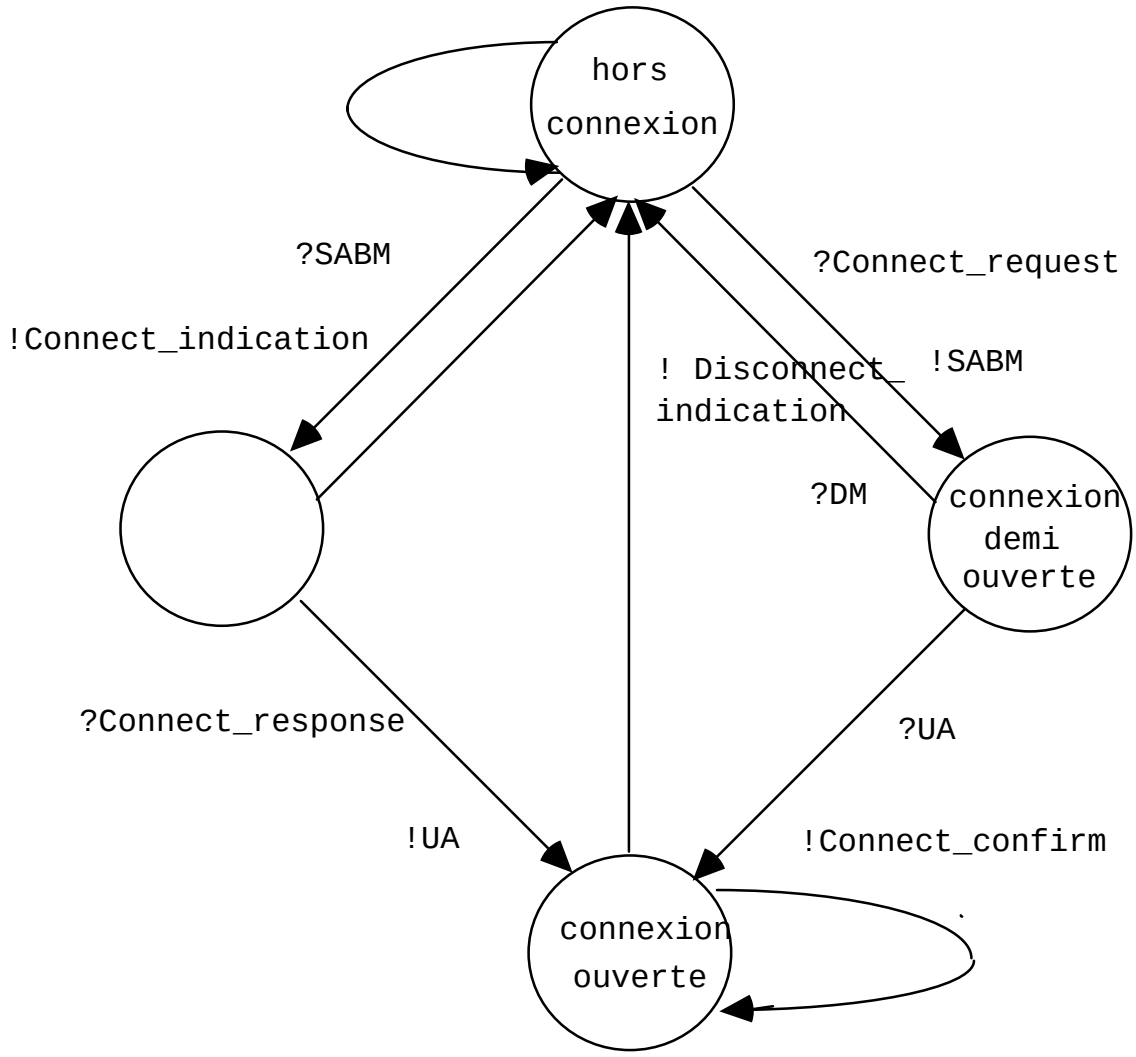
Vision partielle



Automate complet

. Réunion des deux automates de service et de protocole

Vision partielle

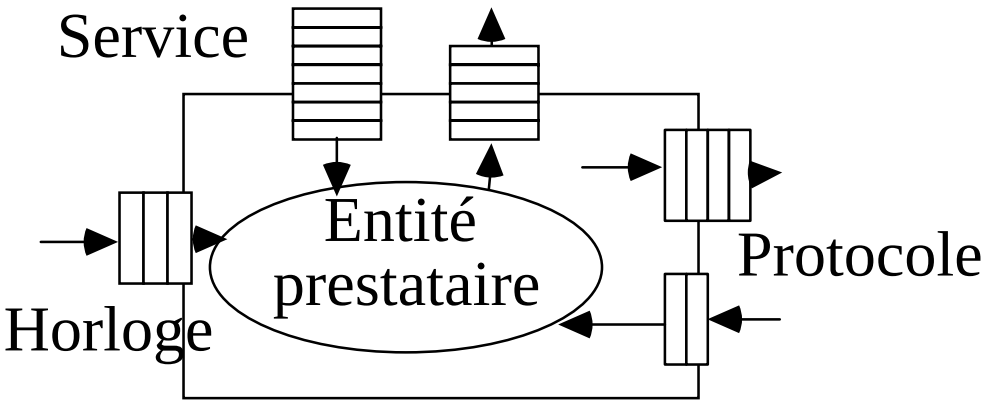


Gestion des files d'interactions

Événements systèmes locaux: retombée signaux d'horloge

Événements distants :
arrivée d'éléments protocolaires

Événements requêtes de service:
arrivée d'éléments de service



Validation des spécifications

Problèmes de constructions

Réceptions non spécifiées

Dans un état un message peut se présenter dont le cas n'a pas été prévu

Interblocages

Dans un état un message (ou une configuration) est attendu qui ne peut jamais se présenter.

Plus généralement

Définition d'assertions de bon fonctionnement sur le modèle à automates communicants.

- assertions portant sur des variables d'état (booléennes)

- assertions portant sur des trajectoires (logique temporelle)

Méthodes de validation systématiques employées

Simulation comportementale

Exécution en **simulation** du comportement décrit par les automates: une partie seulement des comportements sont couverts.

Détection d'erreurs et correction (technique s'apparentant au test du modèle).

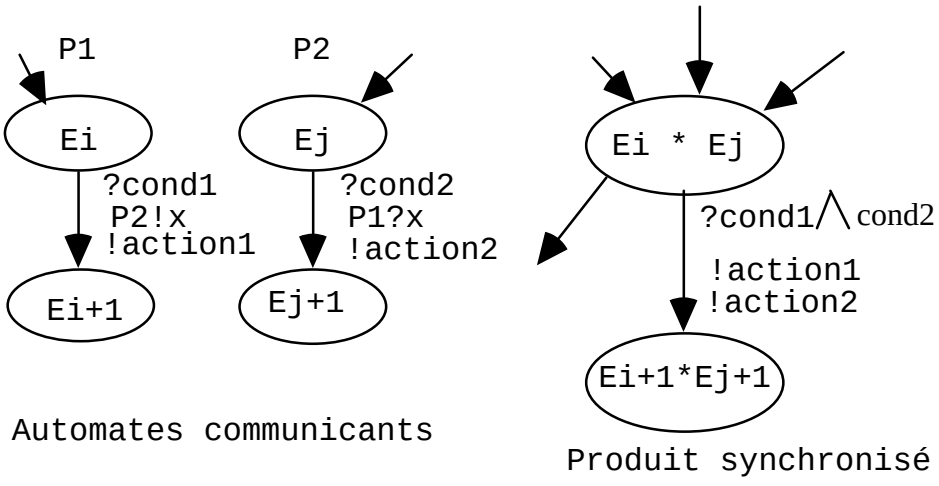
Difficulté majeure : On ne connaît pas la couverture du test (le taux d'erreurs résiduelles après le test).

Exemple d'outil: Estelle- VEDA

Analyse exhaustive.

- L'application est définie par un ensemble d'automates communicants

- Réalisation du produit des automates (produit "synchronisé" tenant compte des communications).



- Obtention du graphe d'accessibilité du système complet.

- Vérification d'assertions sur le graphe complet.

Difficulté majeure

Explosion combinatoire de la taille des espaces d'état.

Conclusion : mode message asynchrone

- C'est le mode le plus basique

Comparable à l'assembleur

Réalisation d'instructions

d'affectation, de branchement.

- Le moins contraignant il permet aux utilisateurs par des échanges successifs, la construction de tout type de protocole.

- L'utilisateur n'a pas en général envie d'être obligé de construire ses propres outils d'où:

Le besoin d'autres schémas prédéfinis plus complexes.

L'enrichissement du mode message en termes de qualité de service par des couches successives.

- Ce mode est encore **le mode privilégié des interfaces réseaux.**

Le Développement De Protocoles En Appel de Procédure Distante

G FLORIN

INTRODUCTION

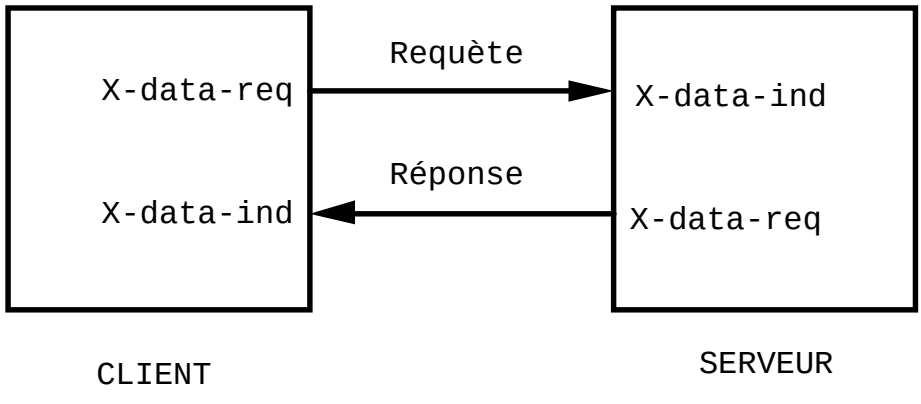
L’approche client-serveur

Mode de fonctionnement système réparti dissymétrique ou:

- un ensemble de clients transmettent des requêtes

- exécutées par des serveurs qui en retournent les résultats aux clients.

On peut donc utiliser l'échange de 2 messages en mode asynchrone (offert par exemple par les couche transport).



Exemple :MMS contrôle de procédé

Le premier message est une requête de lecture d'un capteur. Dans un second message de réponse la valeur lue est retournée .

L’approche "appel de procédure distante"

- C'est une réalisation du mode client-serveur.

L'opération à réaliser est présentée sous la forme d'une **procédure** que **le client** peut faire exécuter à distance par **un autre site** : **le serveur.**

Exemple de service APD:

TYPE COM_APPEL_PROCEDURE_DISTANTE;

/* Transforme un appel local en appel distant */

METHOD genere_appel (id_emet, id_recept nom_procedure, paramètres);

/* Recoit un appel distant et l'exécute localement */

METHOD traite_appel (id_emet, id_recept nom_procedure, paramètres);

METHOD ...;

END COM_MESSAGE_ASYNCHRONE.

Exemple de service intégré objet:

TYPE OBJET_QUELCONQUE

METHOD nom_procedure (paramètres);

END OBJET_QUELCONQUE;

PROBLEMES GENERAUX DE CONCEPTION DE PROTOCOLES D'APPEL DE PROCEDURE DISTANTE

MISE EN OEUVRE DE L'A.P.D

Trois techniques ont été proposées.

PAR MIGRATION

Le code et les données de la procédure distante sont amenés sur le site appelant pour y être exécutés en appel normal**.**

Analogue d'une stratégie de

pré-chargement.

Avantages Inconvénients:

- univers de systèmes homogènes.
- faible volume de codes,
- faible volume de données globales
- exclusion mutuelle entre les exécutions
- très efficace pour de nombreux appels.

EN COMMUNICATION PAR MEMOIRE

L'appel distant est réalisé au dessus d'une couche de mémoire virtuelle répartie.

L'appel se fait en mémoire comme si la procédure était locale, provoquant un premier défaut de page sur le début du code procédure.

Le code et les données de la procédure distante sont amenés sur le site appelant si besoin**.**

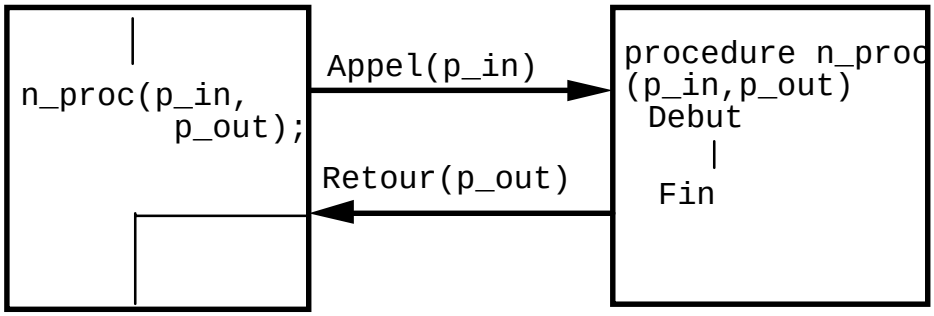
Analogue d'une stratégie de page à la demande.

Avantages Inconvénients:

- univers de systèmes homogènes.
- gros volume de codes peu visité,
- gros volume de données peu visité
- usage important de pointeurs.
- très efficace pour de nombreux appels.

EN COMMUNICATION PAR MESSAGES ASYNCHRONES

=> Deux messages (au moins) échangés:



- Le premier message correspondant à la requête est celui de l'appel de procédure, porteur des paramètres d'appel.

- Le second message correspondant à la réponse est celui du retour de procédure porteur des paramètres résultats.

LES SOUCHES ("Stubs")

"Talons", "Procédures d'interface", "Squelettes"

La souche client ("client stub")

C'est la procédure d'interface du site client:- qui reçoit l'appel en mode local

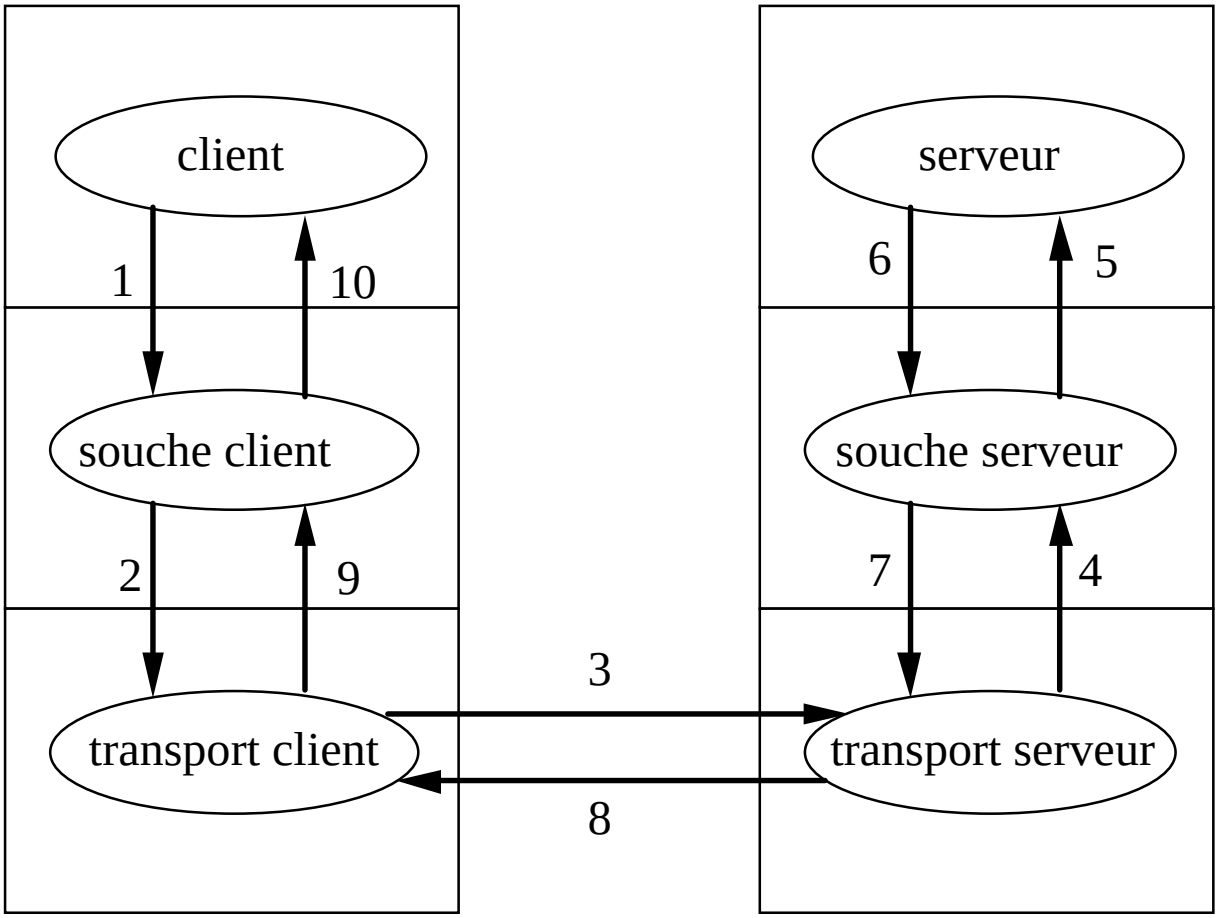
- le transforme en appel distant
- en envoyant un message.
- reçoit les résultats après l'exécution
- retourne les paramètres résultats comme dans un retour de procédure.

La souche serveur ("server stub")

C'est la procédure sur le site serveur qui:

- reçoit l'appel par message,
- fait réaliser l'exécution sur le site serveur par la procédure serveur
- retransmet les résultats par message.

Les dix étapes de traitement



Détail des étapes

Étape 1

- Le client réalise un appel procédural vers la procédure souche client.

- La souche client collecte les paramètres , les assemble dans un message **(“parameter marshalling”).**

Étape 2

- La souche client détermine l’adresse du serveur.

- La souche client demande à une entité de transport locale la transmission du message d'appel.

Étape 3

- Le message est transmis sur un réseau au site serveur.

Étape 4

- Le message est délivré à la souche du serveur.

Étape 5

- La souche serveur désassemble les paramètres, et réalise l'appel effectif de la procédure serveur.

Remarque: Dans l'APD synchrone on a ici réalisé **un rendez-vous d'activation**.

Étape 6

- La procédure serveur ayant terminé son exécution transmet à la souche serveur dans son retour de procédure les paramètres résultats.

La souche serveur collecte les paramètres retour, les assemble dans un message ("parameter marshalling").

Étape 7

- La procédure souche serveur demande à l'entité de transport locale la transmission du message.

Étape 8

- Le message est transmis sur un réseau au site client.

Étape 9

- Le message est délivré à la souche du client. La souche client désassemble les paramètres retour.

Étape 10

- La procédure souche client transmet les résultats au client en effectuant le retour final de procédure.

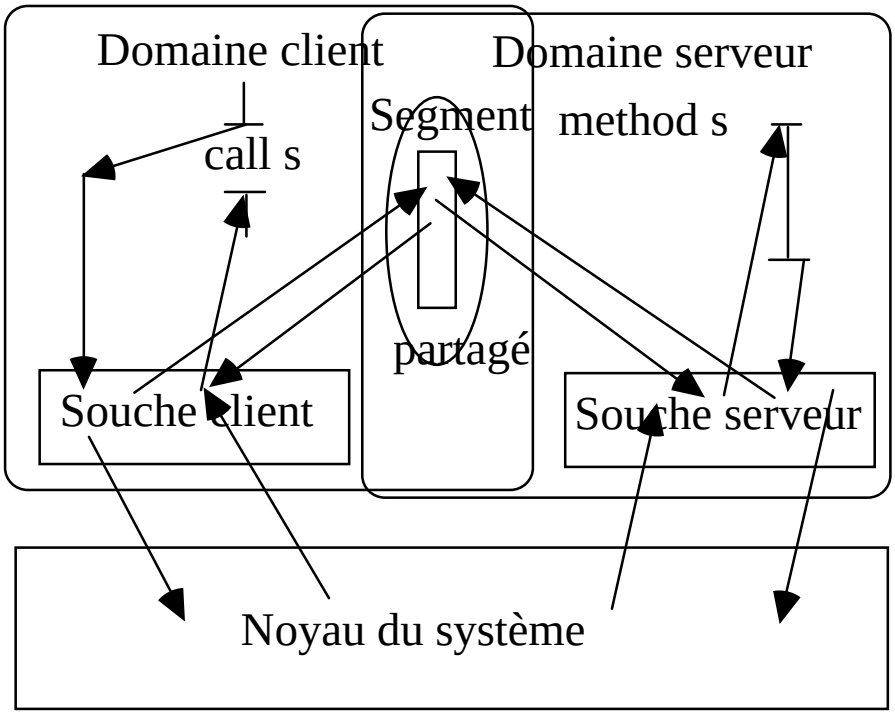
Avantages Inconvénients de l'invocation distante par messages

- Applicable en univers hétérogènes moyennant des conversions.
- volume de code quelconque,
- volume de données quelconque,
- faible usage des pointeurs sur les paramètres.
- problèmes d'entrelacement d'accès sur les données globales analogue au problème transactionnel.
- peu efficace pour de très nombreux appels.

L'appel de procédure distante léger ("Lightweight RPC")

- Quand on invoque un serveur qui se trouve sur la même machine la traversée des couches réseaux est inutile et coûteuse.

- Le serveur se trouve dans un autre processus (autre domaine de protection) sinon pas de problème (appel local).



La communication réseau est réalisée par un segment de mémoire partagée entre le client et le serveur qui contient un tas pour les paramètres d'appel et de réponse.

Conclusion conception APD

L'APD est le plus souvent développé en invocation distante par messages.

- supporte l'hétérogénéité
- finalement le plus simple à réaliser.

Des optimisations peuvent être obtenues par l'usage opportun des autres solutions.

Ex: Chorus Cool a développé les quatre solutions.

Ex: DCOM microsoft RPC par messages + RPC léger (aussi prévu en CORBA).

Sémantique de l'appel de procédure distante

Introduction

Très nombreuses implantations différentes du RPC ayant des sémantiques variées.

L'objectif affiché par la plupart des implantations serait pour le programmeur:

- de retrouver la sémantique habituelle de l'appel de procédure en mode centralisé

- sans se préoccuper de la localisation de la procédure exécutée (sauf s'il le souhaite explicitement).

Objectif difficile à atteindre

De nombreuses réalisations sont au contraire très peu conviviales avec une sémantique différente de l'appel de procédure.

Aspects de communication et de synchronisation

Mode avec ou sans données rémanentes (persistantes)

1 Sans données persistantes

Situation idéale du cas ou l'appel de procédure s'exécute en fonction uniquement des paramètres d'entrée: en produisant uniquement des paramètres résultats.

Il n'y a **pas de modification de données rémanentes** sur le site serveur.

Situation très favorable

- pour la **tolérance aux pannes.**

-pour le **contrôle de concurrence**

Exemple: calcul d'une fonction scientifique simple (sin(x))

2 Avec données persistantes

Les **exécutions successives** manipulent **des données persistantes** sur le site serveur.

Une telle exécution modifie le contexte d'exécution sur le site distant.

=> Problème de contrôle de concurrence.

=> Difficultés en cas de panne en cours d'exécution => validation à deux phases.

Exemple 1 : Un serveur de fichier réparti.

Opérations d'écriture, de pose de verrou.

Exemple 2 :Données persistantes, la structure de donnée manipulée par les méthodes d'un objet déclaré persistant.

Une solution: le couplage d'un moniteur transactionnel et d'un système d'objets répartis.

Mode avec ou sans état

Autre aspect de la rémanence très voisin du cas précédent

Mode sans état

Les appels successifs de procédure s'exécutent sans lien entr'eux.

Il peut y avoir modification de données globales rémanentes sur le site serveur mais l'opération s'effectue sans référence au passé (indépendamment de toutes celles qui ont précédé).

Exemple:

Écriture du nième article d'un fichier dont toutes les caractéristiques utiles (nom, droit d'accès) sont passées au moment de l'appel.

NFS (Network File System de SUN système de fichier réparti sans état).

Mode avec état

Les appels successifs s'exécutent en fonction d’un état laissé par les appels antérieurs.

La gestion de l'ordre des requêtes est indispensable.

Exemple: Lecture d’article de fichier sur le pointeur courant.

Remarque:

La terminologie avec ou sans état porte donc plutôt sur l'existence de données rémanentes au niveau du protocole de RPC (voir aussi l'idée de mode en connexion ou sans connexion).

Appel de procédure distante avec ou sans connexion

Appel de procédure distante sans connexion

- Dans ce mode le client peut envoyer des appels au serveur à n'importe quel moment.

- Le mode sans connexion est donc un mode assez léger orienté:

Vers un **traitement non ordonné** des appels

Vers **l'absence de mémoire** entre appels successifs (**serveur sans données rémanentes, sans état**).

Exemple :

Le client demande l'heure au serveur.

Le client fait calculer une fonction numérique qui ne dépend que des paramètres d'appel.

Exception : système de fichier réparti NFS

Appel de procédure distante avec connexion

- Dans ce mode le client doit ouvrir une connexion avec le serveur avant de pouvoir lui adresser des appels.

- Lorsqu'une suite d'opérations est terminée la connexion est fermée.

- Il y a donc pour une connexion:

. délimitation temporelle des échanges

. maintien de l'état de connexion pour la gestion de paramètres de qualité de service (traitement des pannes, débit, propriétés d'ordre, ...)

- Le mode connexion est donc orienté:
Vers le traitement **ordonné** d'une suite d'appels émanant du même client.
Vers la gestion de **données persistantes** et de protocoles **avec état**.

- Données de connexion du client

Le descriptif de la connexion en cours

Les ressources manipulées pour le client (descriptifs de ressource ouverte).

- Données de connexion du serveur

L'état du serveur est utilisé lors des appels successifs pour:

Vérification de la légalité des accès.

Service dans l'ordre souhaité.

Ex : Serveur de fichier en connexion

Un client manipule un fichier séquentiel au moyen d'un serveur de fichiers distant.

Les requêtes ne doivent pas être déséquencées.

Le serveur maintient un descriptif du fichier comprenant entre autres données rémanentes le pointeur courant d'accès.

Comparaison des modes

- Le mode sans connexion est plus léger que le mode en connexion (pas d'ordre, pas d'état, traitement de pannes simplifié).

- Le mode avec connexion peut permettre de mieux gérer les qualités de service et les pannes.

Les propriétés d'ordre

L'appel de procédure distante peut avoir une qualité de service qui comporte des **spécifications de propriétes d'ordre**.

Le respect d'une propriété d'ordre peut porter: . sur **le l**

. sur **la totalité de son exécution**.

- Ordre local

Le service des requêtes d'un client est réalisé dans l'ordre d'émission.

- Ordre global

Le service des requêtes d'un client est réalisé dans le même ordre sur tous les destinataires (cas des communications de groupe).

- Ordre causal

Le service des différentes requêtes est effectué en respectant la relation de causalité qui existe entre les requêtes.

Gestion du parallélisme

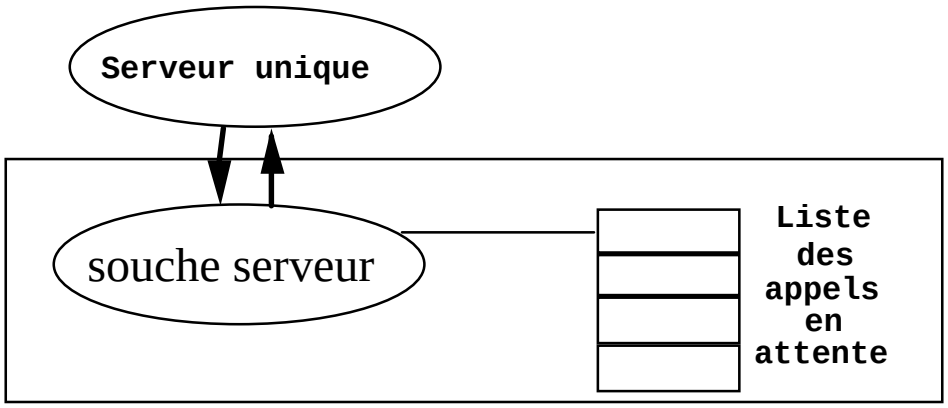
A) Parallélisme dans la réalisation des services.

1 Exécution séquentielle des appels

- Les requêtes d'exécution sont traitées l'une après l'autre dans l'ordre d'arrivée.

- Comme la couche transport utilisée assure en général la livraison en séquence et que le client attend la fin de l'appel on a un traitement **ordonné** des suites d'appels de chaque client en exclusion mutuelle.

Exemple : RPC SUN : requêtes non ordonnées, traitement séquentiel.

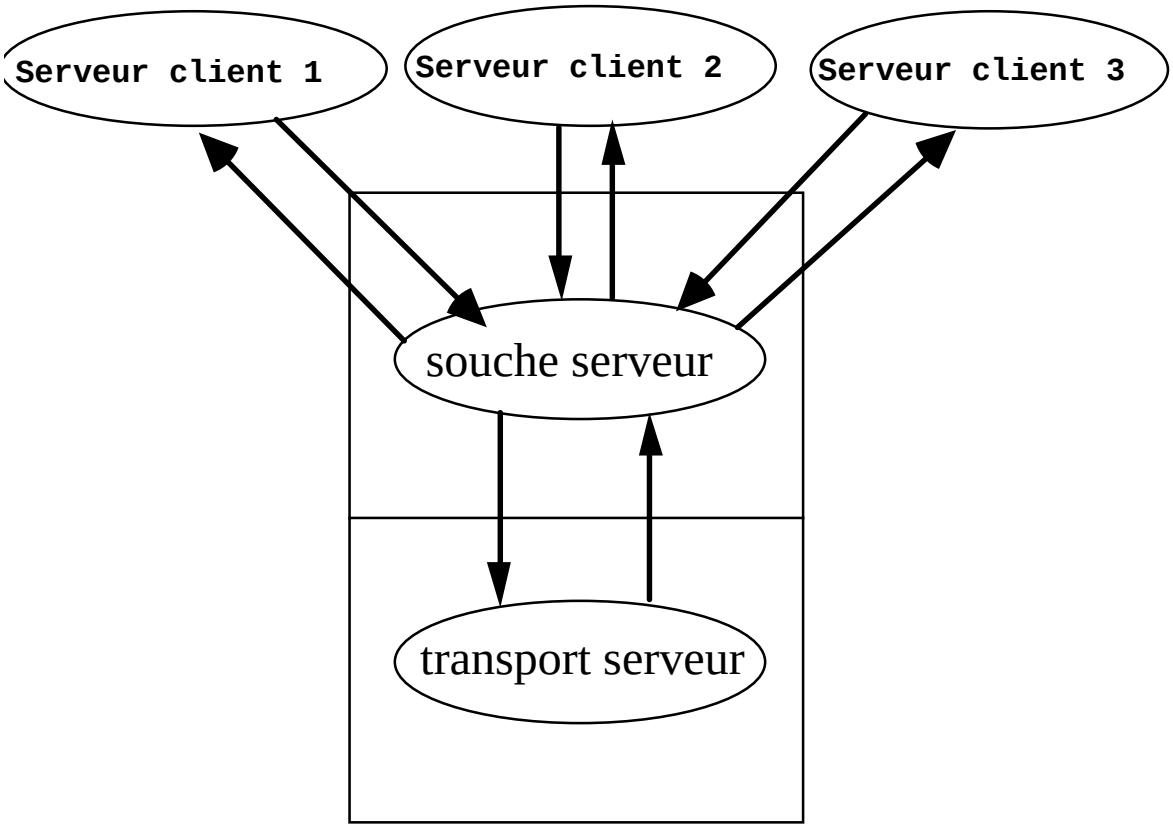


2 Exécution en parallèle des appels

- Dans ce mode le serveur créé une activité (un processus léger ou “thread”) par appel. - Les appels sont exécutés concurremment.

- Si les exécutions parallèles manipulent des données globales rémanentes sur le site serveur, le contrôle de concurrence doit être géré par l’utilisateur.

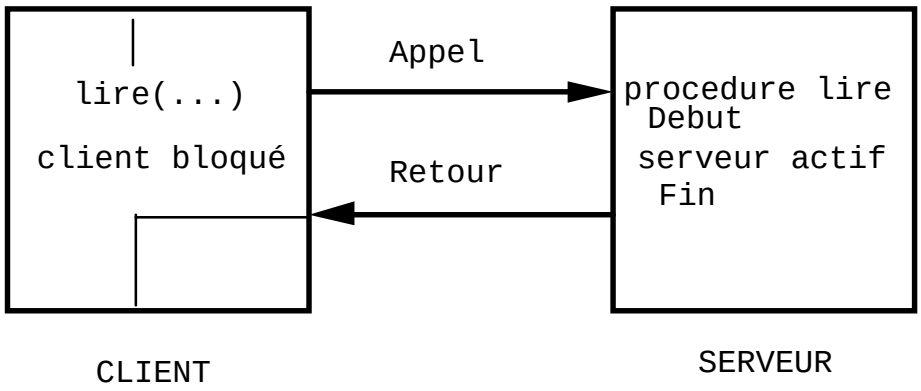
Exemple : Système d'objet réparti GUIDE



B) Le parallélisme chez le client

1 - A.P.D synchrone.

L'exécution du client est suspendue tant que la réponse du serveur n'est pas revenue ou qu'une condition d'exception n'a pas entraîné un traitement spécifique.



Avantage: la sémantique est la même que dans l'appel en mode centralisé.

Critique (habituelle): le client qui pourrait faire quelque chose pendant l'exécution du serveur reste inactif.

L'appel de procédure distante et les activités

Solution au problème précédent: création de deux activités sur le site demandeur:

- **L'une gère l'A.P.D** en restant bloquée.

Le fonctionnement est exactement celui d'un appel habituel et par suite sans difficultés de conception.

- **L'autre activité occupe le site appelant** par un travail à faire.

Exemple : Utilisation d'un constructeur structuré parbegin, parend.

parbegin

activite1: proc_distante (parametres)**;**

activite2: travail local**;**

parend;

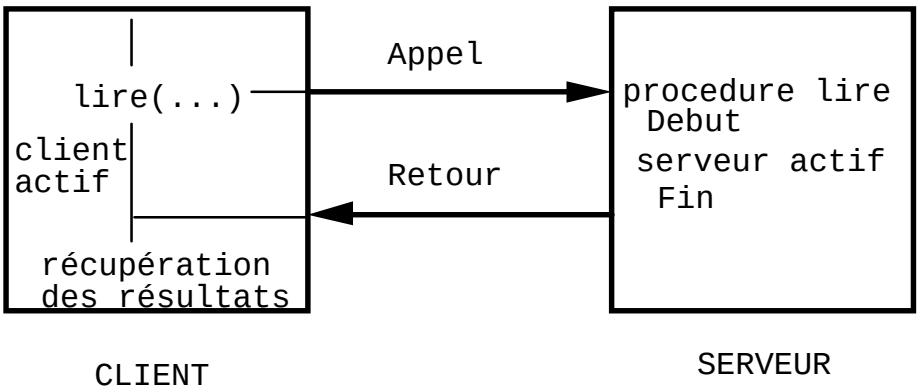
La conception rationnelle de solutions réparties en appel de procédure distante synchrone implique la disponibilité des fonctionnalités des activités.

2 - A.P.D asynchrone

- Le client poursuit son exécution immédiatement après l'émission du message porteur de l'appel.

- La procédure distante s'exécute en parallèle avec la poursuite du client et retourne les paramètres résultats en fin de son exécution.

- Le client récupère les résultats en retour quand il en a besoin (primitive spéciale).



Avantage : Parallélisme plus important en mode standard (sans utilisation d'activités)

Critique : Le client ne retrouve pas la sémantique de l'appel de procédure et doit contrôler la récupération des résultats => problèmes de synchronisation et donc des risques d'erreur.

Invocation asynchrone à sens unique sans réponse ("peut-être", "oneway" , "maybe")

En fait mode message asynchrone utilisé pour déclencher une procédure.

- La procédure ne doit pas avoir à retourner des résultats.

- La récupération d'une réponse si nécessaire doit se faire au moyen d'une invocation d'actions du client par le serveur.

- Le client doit avoir prévu les actions correspondant aux différents types de réponses.

Avantage: Utilisation d'une approche de type appel de procédure mais la communication est de type mode message.

Critique: Uniquement possible en l'absence de retour de résultat, pas d'informations sur la terminaison du travail demandé.

Exemples: CORBA, langage acteur ABCL

Invocation asynchrone avec "futurs"

Notion de **futurs** : en fait des objets particuliers permettant de récupérer des résultats. L'usager ne peut les utiliser immédiatement après l'appel.

Invocation asynchrone à "futur explicite"

Des objets définis par le client ont le statut de paramètre de réponse.

Exemple: En ACT++ une boite à lettre définie par le client sert de moyen de communication asynchrone entre le client et le serveur pour les paramètres résultats.

BAL := factorielle.calculfact(n)

Le client poursuit son exécution immédiatement.

"Invocation asynchrone à futur implicite"

Les objets de communication pour les réponses (ex boite à lettre) sont implicitement créés par le prestataire du service d'APD asynchrone.

Récupération des résultats en APD asynchrone

- Approche générale: défaut d'information.

- Tant que le client n'a pas besoin du résultat il peut poursuivre son exécution.

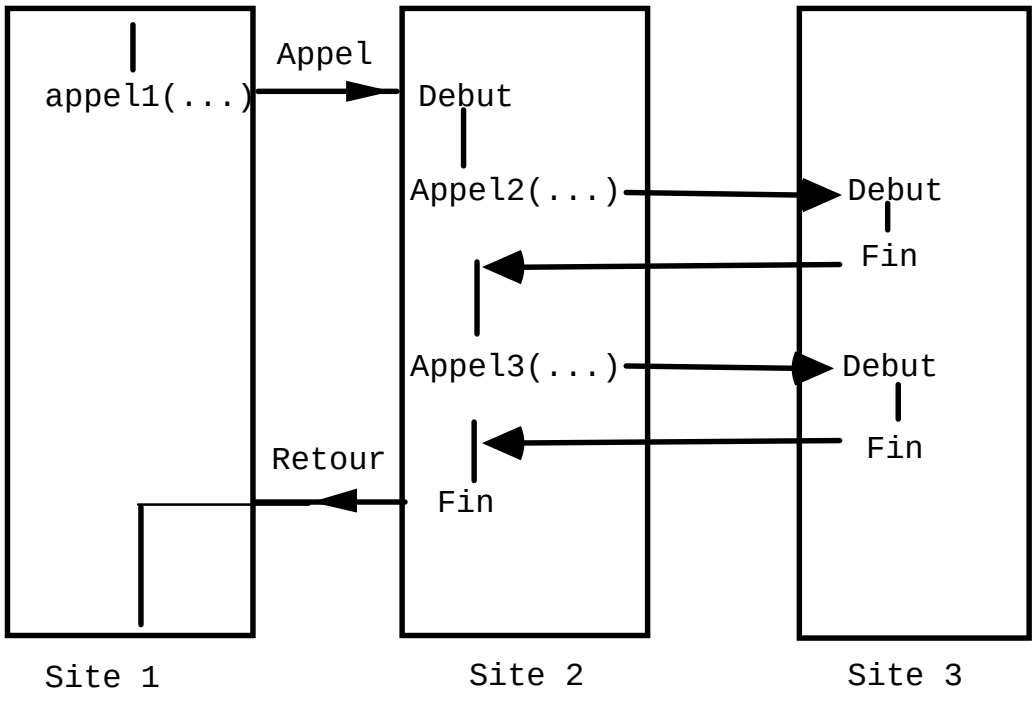
- Il ne se bloque que s'il accède à la réponse et que celle-ci n'est pas parvenue.

Réalisation d'activités réparties par composition de traitements séquentiels

Schémas déduits de l'appel de procédure distante synchrone

- Protocole de demande réponse simple, avec dialogue suivi, avec appel en cascade.

- Application : tous les traitements procéduraux classiques en univers multi-site



Schémas à continuations déduits de l'appel de procédure distante peut-être

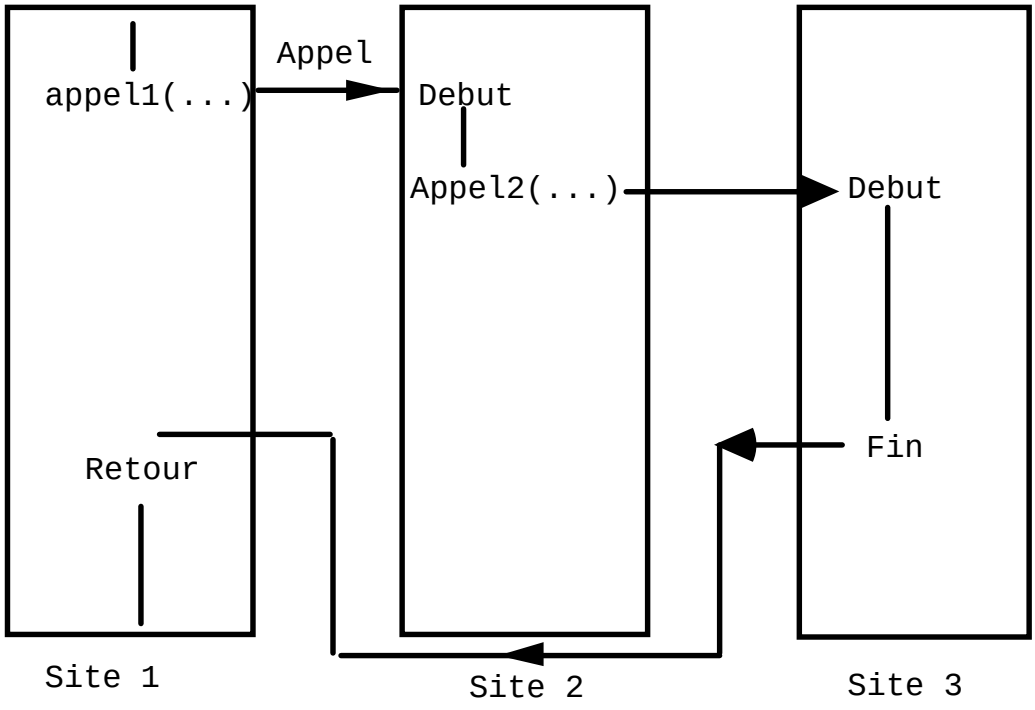
- Sans indication du demandeur initial

- Avec en paramètre le demandeur initial

- Pour l'émetteur un protocole de demande réponse.

- Mais le site serveur sollicité en premier peut déléguer à un autre la charge de réaliser l'opération et de fournir au demandeur la réponse finale sans repasser par lui

- Applications : intelligence artificielle



TRANSMISSION DES ARGUMENTS

Problèmes de présentation : traité dans les réseaux

- Si le site client et le site serveur utilisent des formats internes différents (type caractère, entier, flottant, ...) une conversion est nécessaire.

- Solution placée classiquement dans la couche 6 du modèle OSI Présentation.

- Les réseaux n'envisagent qu'un style de transmission: par valeur.

La solution normalisée:

- Syntaxe abstraite de transfert ASN 1

Problème du passage des paramètres dans l'appel de procédure distante

Existence de sémantiques de transmission variées

Valeur, copie/restauration, adresse, nom.

Passage par copie/restauration

Les valeurs des paramètres sont recopiées de l'appelant à l'appelé (et réciproquement).

Pas de difficultés majeures mais ...

Les solutions déjà définies pour les réseaux sont redéfinies

Optimisation des solutions pour le RPC.

Adaptation au langage C (assez faiblement typé).

Adaptation aux langages objets.

Les IDL "Interface Definition Langage"

Objectif

Générer les souches clients et serveurs à partir d'une définition formelle des données échangées.

. pour réduire les ambiguités => erreurs.

. pour compiler une syntaxe de transfert.

. pour optimiser les communications.

Exemples de directives IDL-DCE

- C ne définit pas les modes de transmission.

. **in,out** mode de passage des paramètres.

Si un paramètres est uniquement in ou out on peut éviter un travail inutile.

- Très souvent en C on ne définit pas correctement le type et la taille des données traitées. ignore permet de ne transmettre qu'une partie significative d'un grand ensemble de données.

Différentes fonctions des IDL

- Etre indépendant des langages utilisant le RPC.

-> Notion de correspondance entre les types retenus dans l'IDL et les types d'un langage ("mappings")

- Permettre aux utilisateurs de définir un identifiant unique pour chaque interface afin de l'utiliser sans ambiguité.

- Pour les langages objets permettre de définir des relations d'héritage entre définitions d'interfaces.

- Structurer les interfaces en modules.

Passage par adresse

Graves difficultés : le passage par adresse utilise une adresse mémoire centrale du site de l'appelant qui n'a aucun sens sur l'appelé.

Solutions indispensables

- a) **Interdiction totale** du passage des paramètres par adresse ou de structures de données contenant des pointeurs.

. Introduit une différence notable dans le développement de procédures locales et celles destinées à un usage distant.

b) **Simulation du passage par adresse** en utilisant une **copie restauration.**

Marche bien dans beaucoup de cas.

Mais violation dans certains cas de la sémantique du passage par adresse.

=> Faire déclarer en IDL les pointeurs et la taille maximale de la structure pointée.

=> Valable pour les pointeurs d'appel/retour

Exemple de problèmes :

commentaire : On incrémente deux entiers;

procédure double_incr (x , y) ;

x , y : **entier** ;

début

x := x + 1 ;

y := y + 1 ;

fin ;

Séquence d'appel: passage par adresse

a := 0 ;

double_incr (a , a) ;

Résultat attendu : a = 2

Utilisation d'une copie restauration

Résultat obtenu : a = 1

- c) Faire réaliser des accès mémoire inter-sites **mémoire virtuelle répartie**

Solution maximale très satisfaisante mais également très coûteuse.

Appel: Défauts d'information serveur->client

Retour: Défauts d'information client->serveur

Nécessité d'un système réparti autorisant la mémoire virtuelle répartie (Chorus, Mach).

Ne se conçoit que dans un système réparti de **calculateurs de même type**.

- d) Simuler des accès mémoire sur le site serveur à chaque utilisation de pointeur

Déclaration des pointeurs

A chaque utilisation d'une variable référencée => **trappe** => recherche sur le site distant. **Très coûteux** dans des usages extensifs ou des pointeurs sont mêlés aux données.

Passage par nom

En approche objets répartis

Utilisation des noms d'objets

(références ou pointeurs d'objets)

Le passage d'un argument se fait en transmettant le nom logique de l'objet.

L'accès aux valeurs s'effectue au moyen des méthodes implantées par l'objet (si c'est un attribut lire et écrire)

Avantages inconvénients

Mécanisme universel

Coûteux pour des accès fréquents

Conclusion : Passage des paramètres

- Solutions normalisées

IDL de DCE => M-IDL (Microsoft)

"Distributed Computing Environnment"

IDL de CORBA

"Common Object Request Broker Architecture"

- Protocoles propriétaires

. SUN - XDR

"eXternal Data Representation"

. XEROX - COURIER,

. NIDL "Network IDL" de HP

. Architectures BD réparties?

- **Nécessité de l'émergence rapide d'une normalisation.**
- **Nécessité de définir des solutions efficaces qui cachent aux utilisateurs les mécanismes de présentation.**

Problèmes de désignation

Liaison statique.

La localisation du serveur (l'adresse système réparti du site serveur et le nom de la procédure) **est connue du client à la compilation**.

Liaison dynamique.

Dans le cas ou les **adresse d'accès ne sont pas connues à la compilation** pour:

- Séparer la connaissance du nom de service de la sélection de la procédure qui va effectivement traiter le problème.

Cacher le problème de **localisation**

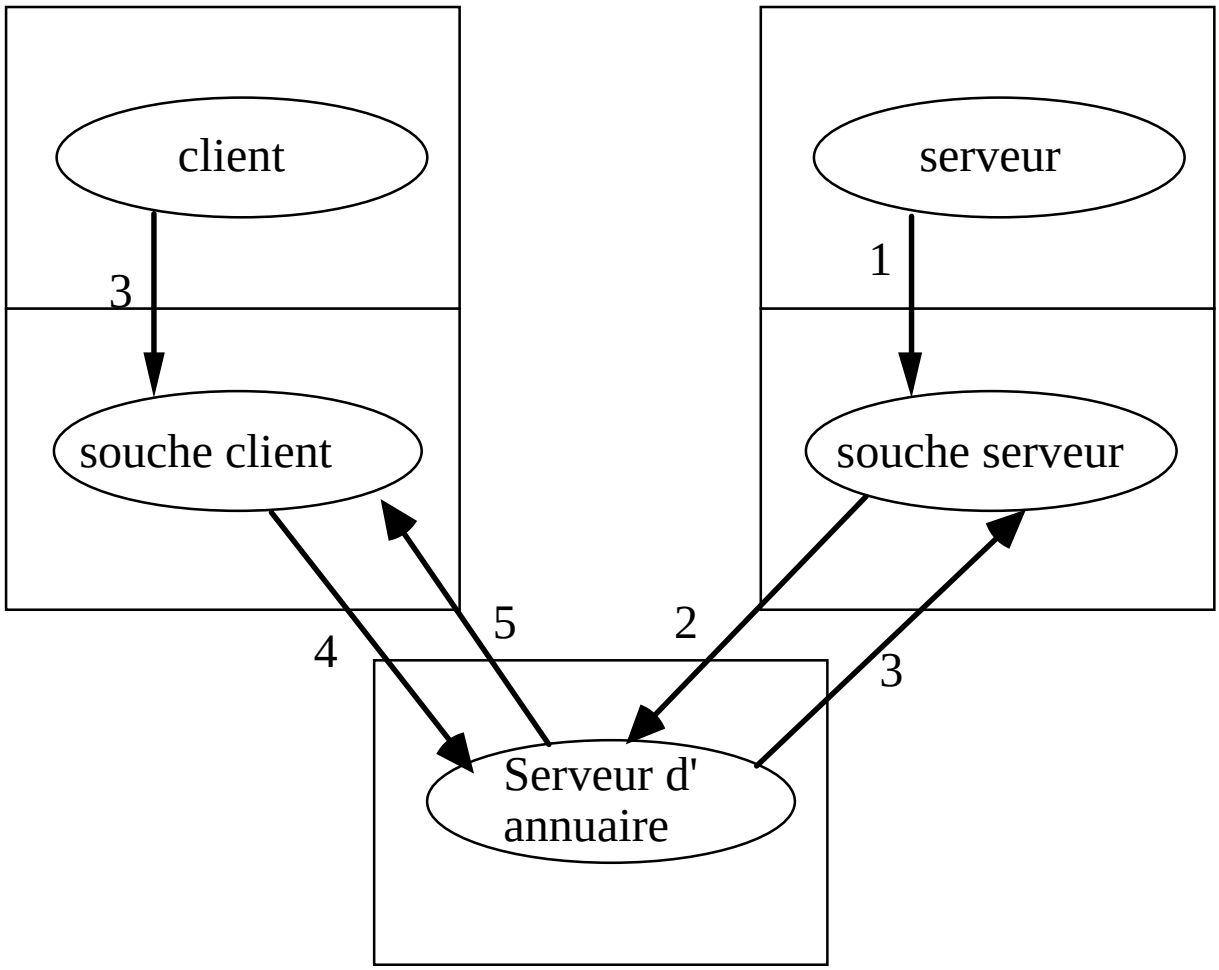
Cacher le problème de la **sélection liée au typage** (surcharge)

Cacher le problème de la **sélection optimale** par le serveur **du traitant** d'une requête (adaptativité, actes de langages)

- Permettre **l'implémentation retardée**

Problème de localisation simple

Solution classique d'un serveur d'annuaire



Exemple: DNS "Domain Name System" Internet ou X500 en DCE.

Étapes 1, 2 et 3 : Enregistrement dans une base de données des noms de serveurs.

- Étape 1 - Le serveur qui réalise un service et souhaite l'offrir à d'éventuels clients demande à la souche serveur d'enregistrer ce service dans la base des noms de services.

- Étape 2 - La souche transmet au serveur d'annuaire

. Le nom du service (caractères)

. L'adresse réseau d'accès du service.

. Tout attribut complémentaire nécessaire

Clef d'accès au serveur: si plusieurs serveurs différents d'un même service

Profil d'appel

- Étape 3 - L'enregistrement est confirmé (pas d'homonymie, ...).

Étapes 4, 5 et 6 : Liaison entre un client et un serveur.

- Étape 4 - Le client demande la réalisation d'une opération définie par son nom logique.

- Étape 5 - La souche client transmet au serveur d'annuaire une demande de localisation du service.

- Étape 6 - Le serveur d'annuaire répond en fournissant la correspondance nom logique adresse réseau.

Conclusion : Liaison

- Localisation dynamique

Le seul problème envisagé au niveau global.

- Autres problèmes de liaison

Rejeté au niveau des serveurs et de la sémantique des langages (typage) ou des applications.

Tolérance aux pannes de l’appel de procédure distante

En appel local

- L'appelant et l'appelé s'exécutent sur la même machine: Même exécutant => mêmes performances et mêmes modes de pannes.

- L'appel et le retour de procédure sont des mécanismes internes considérés comme fiables (sauf systèmes sécuritaires).

- Problème abordé (par certains langages): les erreurs logicielles

=>Problèmes d'exécution de l'appelé qui produisent des exceptions chez l'appelant.

En appel distant

- Le site appelant ou le site appelé peuvent tomber en panne indépendamment.

- Le message d'appel ou celui de retour peuvent être perdus (sauf si l'on emploie un protocole de transport fiable).

- Le temps de réponse peut-être très long en raison de la surcharge du site appelé.

Différents types de pannes

Pannes franches (des calculateurs)

Panne du serveur

1 - Attente indéfinie par le client d'une réponse qui ne viendra jamais.

=> Abandon du programme sur temps limite

2 - Utilisation d'un délai de garde T1 (assez long) par le site appelant

=> Signalement de l'absence de réponse au client qui décide de la stratégie de reprise.

=> Reprise arrière automatique (si possible)

soit tentative d'exécution sur un autre site

soit reprise sur le même site (si relance)

=> **Plusieurs exécutions successives de la même procédure pour une seule demande.**

Panne du client

- Le serveur est dit **orphelin (“orphan”).**

- **Réalisation de travaux inutiles**, utilisation de ressources qui ne sont plus accessibles pour des tâches utiles.

- **Confusion** après relance du client entre les nouvelles réponses attendues et les réponses à d'anciennes requêtes.

=> Nécessité de détruire les tâches serveurs orphelines et de distinguer les requêtes utiles des vieilles requêtes.

Problème :

L’état du client ou du serveur peuvent devenir inconsistants

=> Analyse des modifications de l’état du client et de celui du serveur.

Pannes d’omission (de messages)

- Éventuellement traitées par une couche transport (en mode connexion).

- Il existe des APD implantés pour des raisons d'efficacité au dessus d’une communication non fiable (couche réseau sans connexion exemple Chorus)

=> Mécanisme de traitement des pertes de message à prévoir dans la conception du RPC

Ex : La réponse acquitte la demande

La prochaine requête acquitte la réponse.

=> **On peut encore pour tolérer les pertes de messages lancer plusieurs exécutions de la même requête**.

Pannes temporelles

La réponse ne parvient pas avant une date limite définie par un délai de garde T2 du client (cas des systèmes industriels).

=> Décision à définir en fonction de l'application.

Pannes quelconques

- Traitement encore exceptionnel (systèmes sécuritaires)

- Possibilité d'utiliser une redondance massive (plusieurs exécutions en parallèle de la même requête) avec un algorithme de consensus sur les réponses de type vote byzantin).

Formalisation du problème de pannes
Analyse des transitions entre états

Procédure F appelée à distance:

F

(C_avant, S_avant)---->(C_après, S_après)

État des données État des données

globales avant globales après

- **F dans son traitement modifie l'état du serveur.**

(S_avant)----->(S_après)

- **Les résultats de l'exécution de F en retour modifient aussi l'état du client.**

(C_avant)----->(C_après)

Techniques de reprise sur erreur

En cas de problèmes rééxécution:

$F^{(0)}$, $F^{(1)}$,...., $F^{(n)}$ les tentatives successives

a) Reprise arrière complète

(client et serveur).

- Si les tentatives successives d'exécution du serveur peuvent laisser l'état du serveur indéterminé (panne en cours d'exécution de la procédure).

- Si le client peut être dans un état incertain (panne au moment de la modification en retour des variables persistantes du client).

=> Pour chaque tentative **l'état du client et celui du serveur doivent être restaurés** aux valeurs avant le premier appel:

$F^{(n)}$

(C_avant, S_avant) --> ($C\_après^{(n)}$, $S\_après^{(n)}$)

Validité de la reprise complète

Cas d'une procédure déterministe

- La reprise complète est correcte quelles que soient les pannes

Les états après sont identiques puisque les exécutions répétées du code du serveur sur la même donnée produisent des résultats identiques (la procédure produit un **résultat déterministe**).

Ex : La lecture d'une variable non partagée produit toujours le même résultat.

Cas d'une procédure indéterministe

Les états après sont potentiellement différents puisque les exécutions répétées donnent des résultats qui dépendent du contexte d'exécution.

La reprise est correcte si les aléas conduisant à l'indéterminisme du fonctionnement sont acceptés. Ex : lecture de la valeur d'une estampille de Lamport.

Reprise arrière du serveur seul

Hypothèse importante

Les rééxécutions du serveur sont toujours complètes ou équivalent à une éxécution complète correcte.

Par exemple retransmission d’appel sur délai de garde pour un mode d'exécution séquentiel des APD.

=> **Il n'y a pas de perte de cohérence de l'état du serveur**

Pas d'exécutions partielles interrompues avec des **variables globales rémanentes laissées incohérentes**.

Fonctionnement de la reprise du serveur

- Pas de sauvegarde de l'état client

On peut s'en passer (le client réalise seulement l'écriture des paramètres résultats).

- Pas de sauvegarde de l'état du serveur.

Au moment d'une tentative n l'état du serveur est donc celui qui résulte de la dernière tentative n-1.

F(n)

(C_avant,S_après**(n-1)**)->(C_après**(n)**,S_après**(n)**)

Condition à respecter par le serveur *l'idempotence*

- Les états après sont conservés si des exécutions répétées du code du serveur sur les données résultant de l'exécution précédente produisent des résultats identiques i.e. l'exécution est **idempotente (FoF=F)**.

Exemples relatifs à l'idempotence

- F est idempotente si l'exécution de la procédure ne modifie pas les variables globales ou l’état du serveur.

- F est idempotente si l’état du serveur est modifié seulement à la première exécution de F.

$S\_après^{(n)} = F^{(n)}(S\_après^{(n-1)}) =$

$S\_après^{(n-1)} = F^{(n)} \circ F^{(n-1)}(S\_après^{(n-2)})$

- De très nombreuses fonctions sont non idempotentes.

Exemples :

La lecture ou l'écriture du kième article d’un fichier sont des opérations idempotentes.

L’écriture en fin de fichier est une opération non idempotente.

L'incrémentation d'une variable est non idempotente.

Sémantique de l’appel de procédure distante du point de vue des reprises du serveur

Le protocole d'appel de procédure distante éxécute puis rééxécute automatiquement en cas de problèmes

Exactement une fois

Une seule exécution est effectuée et celle-ci réussit toujours.

Impossible au sens strict des lors qu’une hypothèse de panne est formulée.

Comportement souhaité: le plus voisin possible de celui de l'appel de procédure en mode centralisé.

Solution possible (lourde):

Une suite de n tentatives est effectuée

- Avec sauvegarde complète (état du client et du serveur) avant chaque tentative.

- Pour une procédure déterministe

Au plus une fois

- Cas d'une fonction non idempotente: on ne doit pas l'exécuter deux fois.

- Solution très répandue:

Identification des requêtes + exécution effectuée une seule fois.

. **Si tout va bien** le résultat est retourné à l'utilisateur.

La requête ne sera plus exécutée (modulo le délai de garde de l'historique).

. **Si un problème est détecté**

Il est signalé à l'utilisateur (s'il est possible de statuer).

Aucune tentative de reprise n'est effectuée automatiquement. Elle est laissée entièrement à la charge du client.

Conclusion: Ce protocole au plus une fois est probabiliste (nombre d'exécutions: une fois, pas du tout, on ne sait pas)

Au moins une fois

On se place dans l'hypothèse ou chaque exécution est réalisée sans sauvegarde de l'état du serveur.

L'exécution est lancée plusieurs fois si nécessaire (dans une certaine limite) pour obtenir une réponse correcte .

=> La procédure doit être **idempotente**.

Variante: La dernière de toutes

Même protocole que précédemment

On fait de plus l'hypothèse que les appels sont numérotés et traités séquentiellement.

On est capable d'attribuer la réussite de l'exécution à la dernière de toutes les tentatives.

Traitement des pannes du client

Abandon de traitement des serveurs en cas de panne du client.

Les serveurs doivent laisser un état cohérent des exécutions.

Extermination

- La souche client note en mémoire stable tous les A.P.D en cours.

- Lorsqu'un site client est relancé, il examine l'historique des A.P.D non terminés et demande la destruction de tous les serveurs orphelins encore actifs.

Difficultés

La recherche des orphelins peut-être longue et difficile en cas d'appels en cascade

La destruction des orphelins est conditionnée par la relance du client.

Réincarnation

Le client gère **un numéro d'époque** correspondant a ses périodes d'activité successives

Il doit être enregistré en mémoire stable et doit marquer toutes les requêtes.

Lorsqu'un client est relancé il change d'époque et diffuse sa nouvelle époque à ses serveurs qui détruisent les appels anciens.

Notion de **réincarnation douce**

- Un serveur recevant un changement d'époque scrute le demandeur des appels anciens et vérifie que le client est bien disparu.

Expiration

- L'exécution d'un serveur est toujours associée à **une surveillance périodique** du client : le serveur arme des délais de garde.

- Si l'exécution se termine avant l'échéance le serveur transmet sa réponse qui retrouve ou ne retrouve pas son client.

- **Si le quantum s'achève,** le serveur **demande a son client s'il est encore opérationnel**:

. **si le client répond** : armement d'un nouveau délai et poursuite.

. **si le client est en panne** : abandon cohérent d'exécution.

Remarque : Cette technique permet à la fois la surveillance du client par le serveur et celle du serveur par le client puisqu'après un échange question/réponse tous les deux peuvent induire que leur interlocuteur est opérationnel.

Conclusion : Conception d'une application en appel de procédure distante

Le mode appel de procédure distante est en développement important pour la conception et la réalisation des applications réparties.

- C'est le support naturel de l'approche client-serveur.

- C'est le mode de communication qui supporte la notion d'objets répartis.

Mais l'engouement actuel autour du mode **client-serveur, de l'APD** et de l'approche **objet**.

L'impression trompeuse que peuvent avoir les usagers parcequ'ils ont une **intuition forte de l'appel de procédure**

Font illusion.

Une application en APD est une application répartie qui présente à un moment ou à un autre pratiquement toutes les difficultés systèmes/réseaux:

- **de conception.**
- **de désignation** (et protection).
- **de présentation** des données échangées.
- **de synchronisation**
- **de contrôle de concurrence**.
- **de tolérance aux pannes.**
- **de performances.**
- **de disponibilité d'outils conviviaux.**
-

Le développement de protocoles RPC intégrés aux approches langages ou objets devrait encore mobiliser longtemps les énergies des chercheurs et des développeurs.

EXEMPLE INDUSTRIEL DE PROTOCOLE D'APPEL DE PROCEDURE DISTANTE

DCE

"DISTRIBUTED COMPUTING ENVIRONMENT"

Introduction OSF/DCE

L'OSF

- L'OSF "Open Software Foundation" un consortium d'entreprises d'informatique

(IBM, Bull, Hewlett Packard, Sun, Digital, Hitachi, Siemens-Nixdorf)

- L'OSF développe des produits

(ex: DCE)

- L'OSF fonctionne par "requête de technologie".

- Les entreprises membres font des propositions et reçoivent des commandes

- L'OSF assure la conduite du projet et rétrocède des licences à ses membres

DCE : L'architecture et les services

- C'est un environnement pour développer des **applications réparties en RPC**.

- DCE **s'installe sur les systèmes constructeurs**.

- DCE **peut être considéré comme un ensemble de couches hautes** analogues à l'ensemble session présentation application.

- **DCE propose des services nettement différents de l'OSI plus proche de l'INTERNET**.

<table>
<tr><td colspan="6">APPLICATION DCE</td></tr>
<tr><td rowspan="2">ACTIVITES "THREADS"</td><td rowspan="2">Appel de procédure distante RPC</td><td rowspan="2">SERVICES DE SECURITE</td><td colspan="2">SERVICES ANNUAIRE</td><td rowspan="2">SERVICES DE FICHIERS REPARTIS</td><td rowspan="2">SERVICES DE SYNCHRO D'HORLOGE</td></tr>
<tr><td>CDS</td><td>X500</td></tr>
<tr><td colspan="6">FONCTIONS DE TRANSPORT D'UNE ARCHITECTURE RESEAU
SYSTEME D'EXPLOITATION CONSTRUCTEUR</td></tr>
</table>

1 Les activités DCE

Caractéristiques des activités DCE

"Threads- Lighweight processes"

- Conforme à la recommandation POSIX "Portable Opertaing System unIX"

POSIX 1003.4a draft 4:Extension temps réel

POSIX est une tentative de normalisation UNIX par la définition d'une bibliothèque d'interface d'appels systèmes.

- En fait 51 primitives sont définies pour

. La création, la destruction, la synchronisation d'activités.

. La gestion de priorités entre activités pour introduire des aspects temps réel.

. La synchronisation entre activités.

- Quelques exemples de primitives.

La gestion des activités

Primitives de gestion des activités

Create	Création
Exit	Terminaison
Join	Wait Unix
Detach	Thread détachée

La gestion des modèles d'activités

- Notion de modèle d'activité: pour positionner des attributs par défaut d'activités.

- Au moment de la création d'une nouvelle activité la référence au modèle permet de configurer l'activité.

- Utilisation de verrous pour la protection des données partagées entre activités, et de variables de conditions pour le blocage en attente de terminaison d'autres activités.

Primitives de gestion des modèles d'activités

Attr_create	Création modèle
Attr_delete	Destruction modèle
Attr_setprio	Priorité par défaut
Attr_getprio	Lecture priorité
Attr_setstacksize	Taille de pile
Attr_getstacksize	Lecture taille pile
Attr_mutexattr_create	Création modèle mutex
Attr_mutexattr_delete	Destruction modèle mutex
Attr_mutexattr_setkind_np	Type de mutex
Attr_mutexattr_getkind_np	Lit le type
Attr_condattr_create	Modèle variable condition
Attr_condattr_delete	Détruit modèle variable

La gestion des verrous

Primitives de gestion des verrous

Mutex_init	Création mutex
Mutex_destroy	Destruction
Mutex_lock	Pose verrou réussie
Mutex_trylock	Echec si vérouillé
Mutex_unlock	Déverouiller

Autres groupes de primitives

- Gestion des variables conditions.
- Gestion de variables globales.
- terminaison des activités.
- Ordonnancement.

2 La désignation dans DCE

Caractéristiques de la gestion des noms de DCE

- Notion de cellule

Une entité de désignation.

Qui comporte un ensemble de calculateurs.

Géré par un serveur de noms: le CDS.

"Cell Directory Service"

- Au niveau global

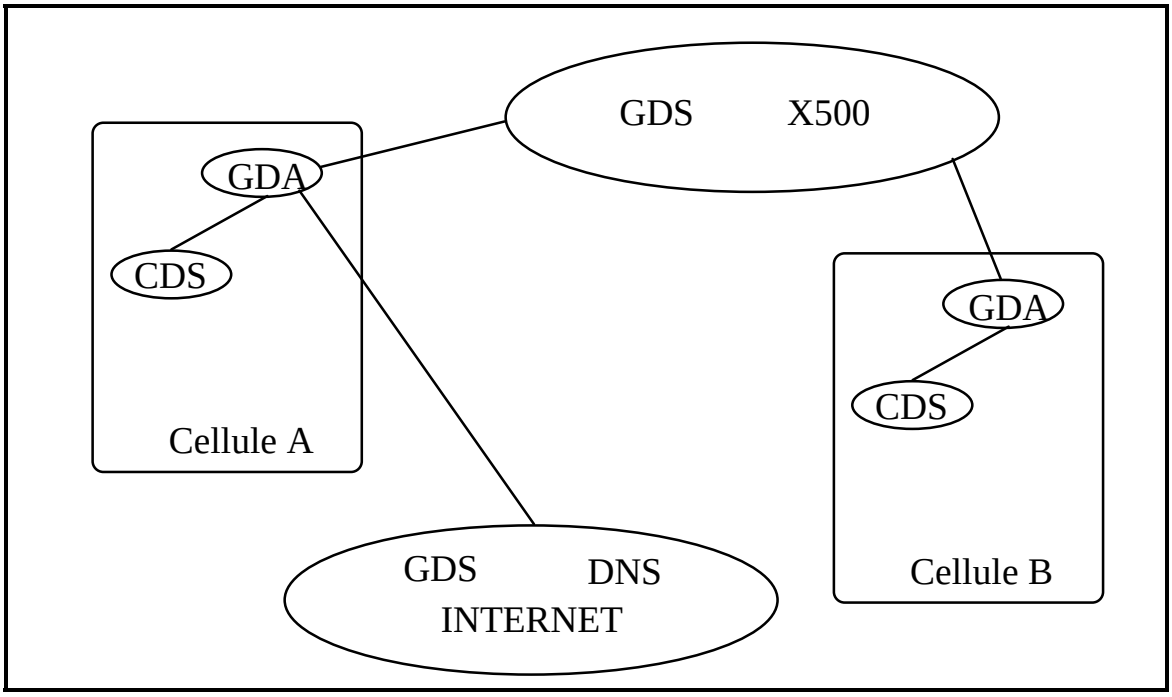
Pour chaque cellule un agent permet de sortir: GDA "Global Directory Agent"

L'adressage se fait au moyen de standards de désignation répandus GDS "Global Directory Service"

GDS - X500 Normalisé OSI

GDS - DNS Domain Name Service

Architecture de la désignation DCE



Syntaxe de désignation

A la UNIX avec des syntaxes d'exception

Désignation de sites et de fichiers

Nom global de service : Préfixé par /...

Dans la cellule de nom X500 C=FR;O=CNAM;OU=deptinfo
Un nom /UV/C_systeme en syntaxe X500
/.../C=FR/O=CNAM/OU=deptinfo/UV/C_systeme
Un nom /UV/C_systeme en syntaxe internet
/.../deptinfo.cnam.fr/UV/C_systeme

Nom contextuel de service dans une cellule : Préfixe /.:

Le même nom désigné de façon contextuelle.
/.:/UV/C_systeme

Noms de fichiers syntaxe différente (DFS "Distributed File Service") : Préfixe /:

/:/users/ensinf/gerard/.login

3 L'appel de procédures distantes DCE

Caractéristiques du RPC - DCE

-Parallelisme du client

Utilisation du service de gestion d'activités.

- Parallelisme du serveur

Deux modes autorisés au moment de l'initialisation du serveur:

- Un seul serveur à la fois: exécution séquentielle, exclusion mutuelle

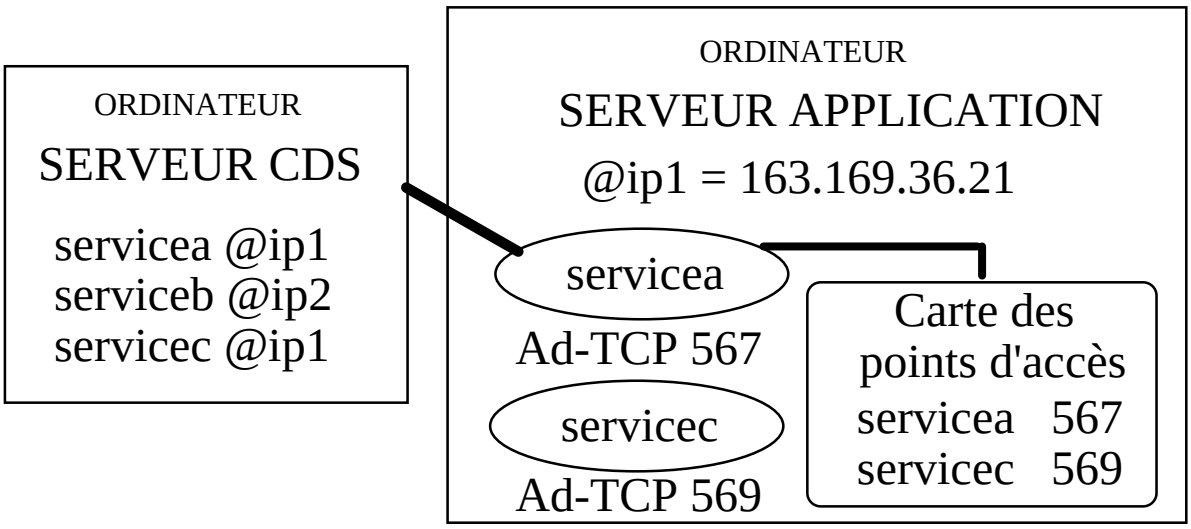
rpc_server_listen (1, &status);

- n services simultanés autorisés

rpc_server_listen (n, &status);

La désignation des services

Enregistrement d'un service: En Internet



- Nom global du site supportant le service: géré dans les tables du serveur de nom CDS

- Nom local du service sur la machine hôte du service: table "Endpoint Map" Carte des points d'accès.

Localisation d'un service: En Internet

Un client demande à son CDS de localiser la machine qui correspond au nom logique du service.

Il se met en relation avec le site du service ou un démon rpcd le renseigne sur l'adresse TCP du service.

La transmission des paramètres

Langage de spécification d'interfaces IDL-DCE

- Il reprend pour la déclaration des types la **syntaxe et la sémantique** du langage **C**.

- Il reprend les **conventions** de transmission des **paramètres entre routines de C**

=> problèmes délicats avec les autres langages

=> recommandation DCE : écrire une routine en C qui récupère les paramètres

- Le compilateur **génère les souches en C**

- Il rajoute pour ses besoins propres des constructeurs (**baptisés attributs**) placés entre crochets.

=> Véritable originalité du langage.

Quelques exemples d'attributs

. **uuid** définit l'dentificateur unique d'une interface IDL. Un outil UUIDGEN assiste l'utilisateur pour générer des noms uniques.

. **in,out** mode de passage des paramètres.

. **ignore** permet de ne transmettre qu'une partie significative d'un grand ensemble de données.

. **in_line** permet de forcer l'exécution sur place.

. **byte** permet de définir des données non converties (suites d'octets).

. **pipe** permet de définir un type flot d'octets (type tube : gros volumes de données à transmettre ou générées par un tube)

. **comm_status** paramètre résultat en cas d'erreur de communication (pour fournir un début de signalement d'exceptions).

. **fault_status** paramètre résultat en cas d'erreur du serveur (exceptions).

Exemple de programme IDL

Codage en DCE d'un appel distant d'une procédure qui se contente de retourner:

ca_marche

```
/* Texte IDL pour une procédure qui retourne "ca marche" */
/* Identiant unique de l'interface et numéro de version DCE */
[uuid(FFC349D3-3705-10EF-345FADBC789FE, version(1.0)]
/* Déclaration d'une interface pour la fonction ca_marche*/
interface ca_marche
{
/* Déclaration des types de données utilisées */
const long MAX_CHAINE = 30;
typedef [string] char type_chaine [MAX_CHAINE+1];
/* Profil d'appel pour la seule fonction implantée*/
void envoie_ca_marche ([out] type_chaine chaine_retournee);
}
```

Le développement d'une application en mode appel de procédure distante avec DCE

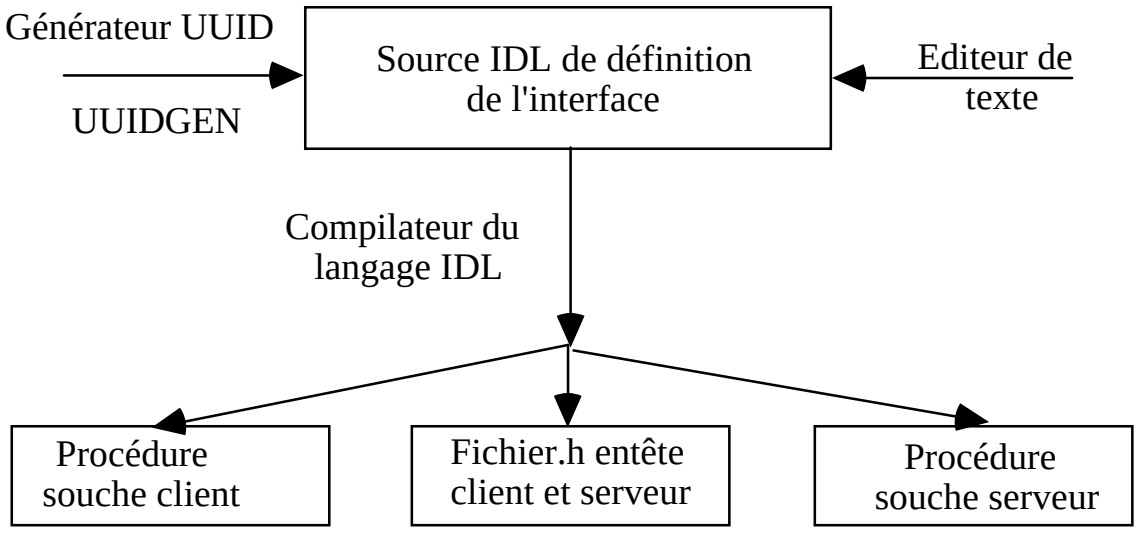
- **Développement de l'interface de communication** : spécification en langage IDL (Interface Definition Langage" des paramètres échangés lors des appels)

- **Développement du client**

- **Développement du serveur**

Le code du serveur et son enregistrement.

Chaîne de production de l'interface



La préparation des procédures clientes

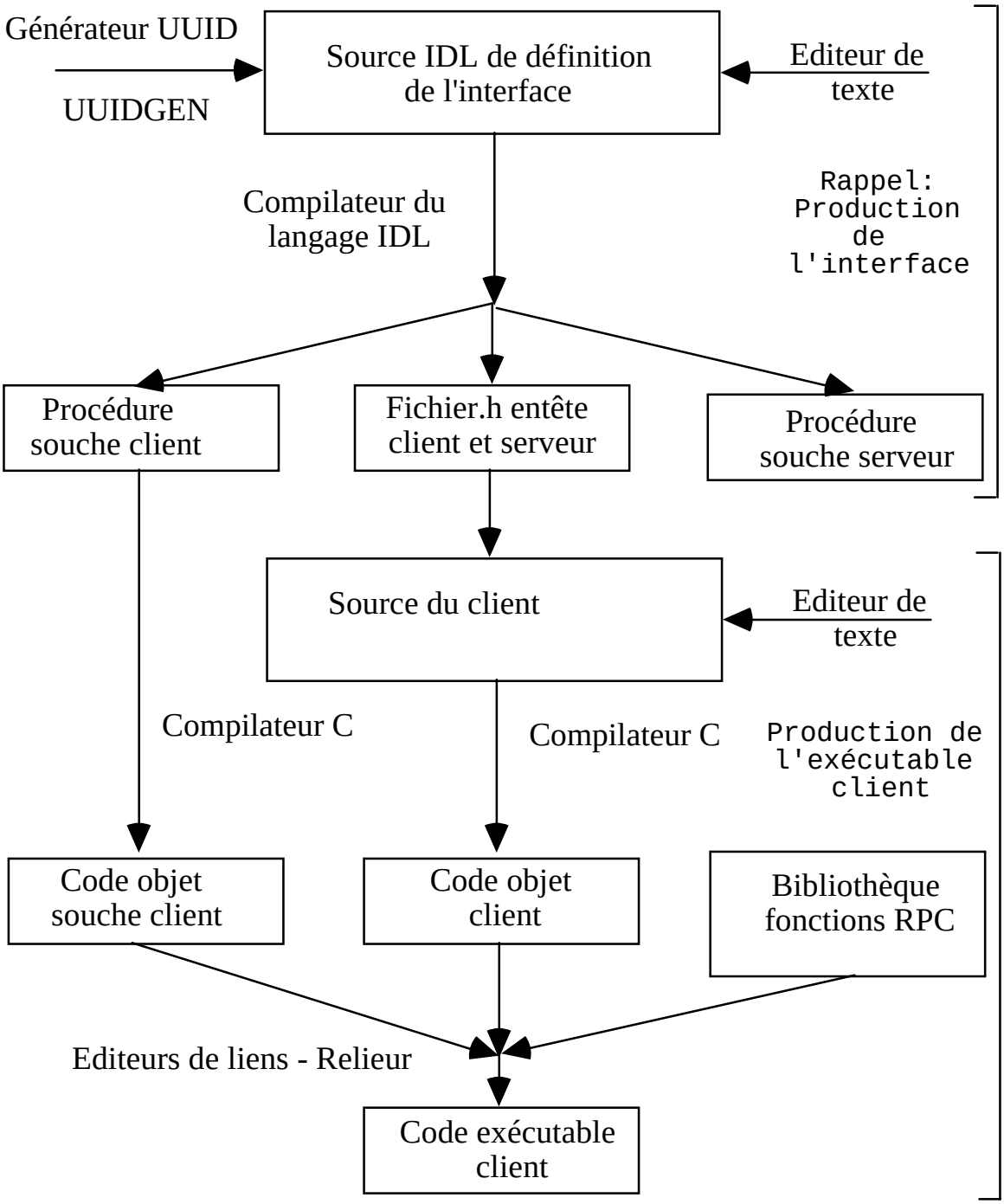
- Le client se contente d'appeler la procédure (ici envoie_ca_marche) en respectant les types définis pour l'interface.

- Le client peut gérer s'il le souhaite les exceptions liées aux erreurs (en univers réparti).

Exemple de programme client pour demander l'exécution de envoie_ca_marche

```
#include <stdio.h>
#include "ca_marche.h"
/* Déclaration de type: uniquement pour la chaine retournee*/
/* type_chaine est predefini maintenant dans ca_marche.h */
type_chaine chaine_retournee;
main()
{ printf("Début du client: \n");
envoie_ca_marche(chaine_retournee);
printf("Le client reçoit du serveur: %s\n", chaine_retournee);}
```

Chaîne de production du client



La préparation des procédures serveurs

- Le serveur se contente de renvoyer la chaîne ca_marche.

- Il doit par contre se déclarer à un serveur de nom pour que le client le trouve.

Exemple de programme serveur pour retourner la chaîne "Ca marche"

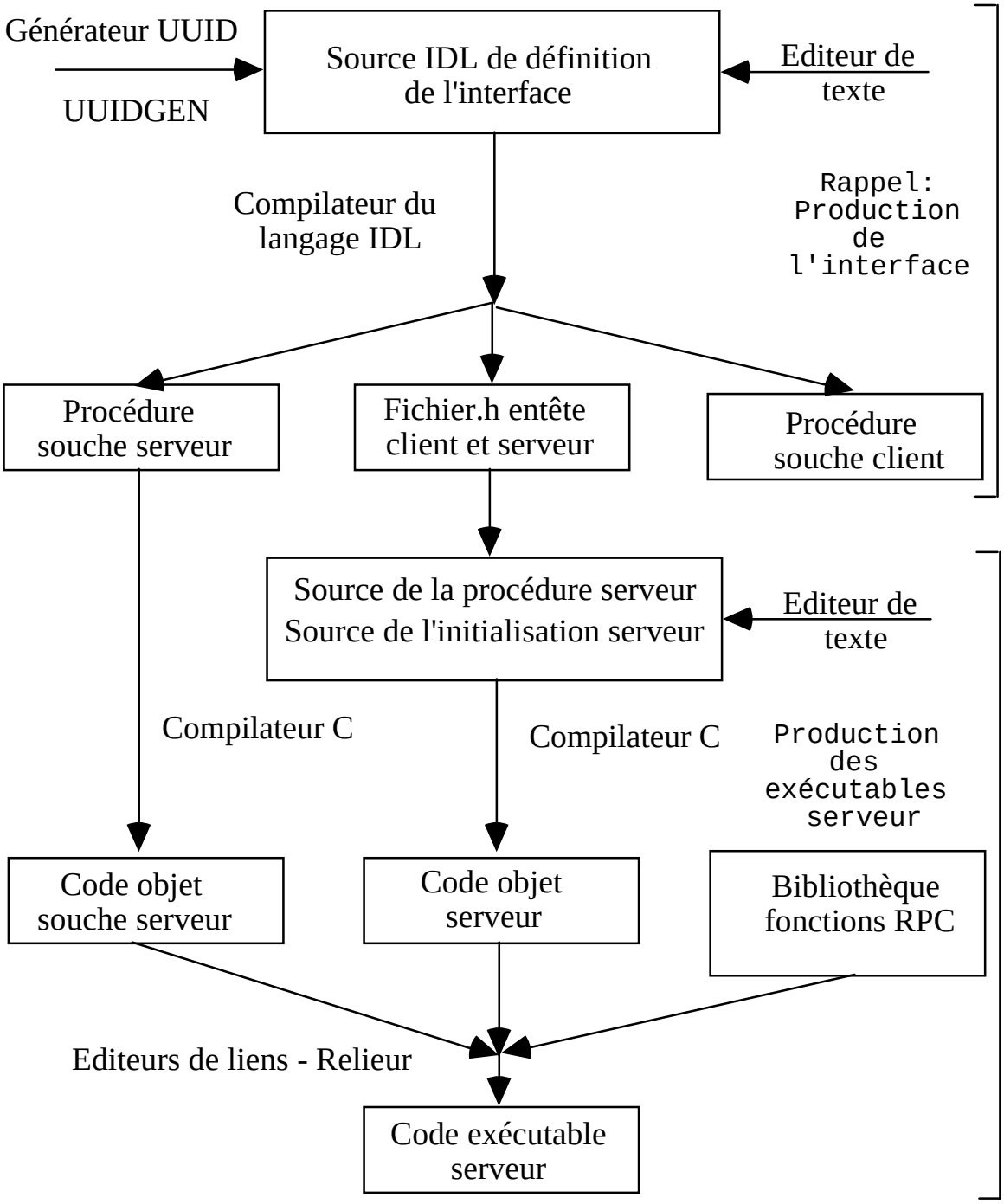
```
#include <stdio.h>
#include "ca_marche.h"
void envoie_ca_marche(chaine_retournee)
type_chaine chaine_retournee;
{
 sprintf((char*)chaine_retournee, "Ca marche");
}
```

Exemple de programme pour initialiser le service envoie_ca_marche

```
/* Ce programme doit tourner sur le serveur avant usage de
envoie_ca_marche pour enregistrer ce service        */
#include <stdio.h>
#include <ctype.h>
#include "ca_marche.h"
define LG_CHAINE 80
main ()
{
 unsigned32 code_rep;
 rpc_binding_vector *binding_vector;
 char nom_entree  [LG_CHAINE];
 char nom_hote [LG_CHAINE];

/* Appel de procédure d'enregistrement d'interface*/;
rpc_server_register_if(
      ca_marche_v1_0_s_ifspec ,
      NULL ,
      NULL ,
      &code_rep);

/* Enregistrer les protocoles utilisés*/;
rpc_server_use_all_protseqs(
      rpc_c_protseq_max_reqs_default,
      &code_rep);

/* Obtenir les informations de liaison du serveur*/;
rpc_server_inq_bindings(
      &binding_vector,
      &code_rep);
```

```
/* Nouveau nom de service dans l'espace des noms*/
strcpy(nom_entree , "/.:/ca_marche_");
gesthostname(nom_hote, LG_CHAINE)
strcat(nom_entree,nom_hote);
rpc_ns_binding_export(
        rpc_c_ns_syntax_default,
        (unsigned_char_t*) nom_entree ,
        ca_marche_v1_0_s_ifspec ,
        binding_vector,
        NULL,
        &code_rep)

/* Enregistrer le point d'entrée du nouveau service dans l'espace des
noms*/
rpc_ep_register(
        ca_marche_v1_0_s_ifspec ,
        binding_vector,
        NULL,
        (unsigned_char_t*) "Interface de ca marche",
        &code_rep);
/* Libérer l'espace occupé par le vecteur de liaison */
rpc_binding_vector_free(
        &binding_vector,
        &code_rep)
/* Mettre le serveur en attente d'un client à la fois*/
rpc_server_listen(
        1,
        &code_rep)

/* C'est enfin fini mais on a pas traité de la protection*/
}
```

Chaîne de production du serveur



CONCLUSION DCE

Nombreux autres services disponibles (plus ou moins complétement)

Service de sécurité, de temps, de fichiers

Egalement service d'administration (DME)

Situation du produit

Nombreuses fonctions opérationnelles

Mais également des fonctions encore en définition ou en développement.

Avenir

Ne peut être utilisé tel que par les utilisateurs finaux

Unique avenir comme support d'applications réparties de plus haut niveau.

BIBLIOGRAPHIE

A.S Tannenbaum "Computer Networks" Prentice Hall

A.S. Tannenbaum "Distributed Operating Systems" Prentice Hall

W Rosenberry, D Kenney, G Fisher "Comprendre DCE" Addison Wesley

A Bond, L Gasser "Readings in distributed artificial intelligence" Morgan Kaufman Publishers

JP Banâtre "La programmation parallèle"

LA TOLÉRANCE AUX PANNES

DANS LES SYSTÈMES RÉPARTIS

G. Florin

Laboratoire CEDRIC

CNAM

PLAN DE L'EXPOSE

- Introduction: concepts de base de la sûreté de fonctionnement

- Classification des pannes.

- Différents types de redondances.

- Principaux problèmes de la tolérance aux pannes.

- Conclusion.

INTRODUCTION: CONCEPTS DE BASE

DES SYSTÈMES REPARTIS

SURS DE FONCTIONNEMENT

Système sûr ("dependable")

Qualitativement on peut avoir "**confiance**" dans le service qu'il rend.

Système réparti sûr

- Ensemble de plusieurs calculateurs reliés en réseau qui collaborent pour des traitements à contrainte de sûreté

En contrôle de processus industriel
. fortes contraintes de sûreté
. contraintes d'échéances fortes/faibles
. support de dispositifs spécifiques

Mais aussi en gestion transactionnelle
. faibles contraintes de sûreté
. faibles contraintes d'échéance

Tout système réparti doit assurer la prise en compte des contraintes de sûreté (algorithmique répartie des systèmes et des réseaux, calcul intensif, ...)

Terminologie de la sûreté de fonctionnement ("Dependability")

Les fautes (erreurs de conception, accidents, malveillances)

L'existence d'**erreurs de conception** ("design errors") ou **d'accidents** physiques ("physical damage") ou de **malveillances** est inévitable.

- Erreurs de conception, programmation ou de saisie.
- Accidents dus à l'environnement.
- Malveillances intentionnelles.

Les états erronés

L'une des circonstances précédentes peut ne pas gêner le système ou le gêner en conduisant à un état erroné (non prévu).

Cet état peut rester indétecté longtemps (latence de la faute) mais conduit à court ou long terme à une défaillance ou panne.

Problème: définir la faute qui à conduit à l'état erroné.

Défaillances, pannes ("Failures")

Le système est **en panne** si suite à l'un des phénomènes précédents il ne respecte pas l'une de ses spécifications.

La panne est la manifestation au niveau du service rendu d'une faute.

Systèmes de sécurité, systèmes critiques "Safety critical systems"

Le domaine d'utilisation du système est particulièrement dangereux et met en jeu des vies humaines avec des coûts liés aux pannes qui peuvent être immenses.

Domaine des transports
- Conduite automatique de trains.
- Systèmes de contrôle en avionique.

Domaine de la production d'énergie
- Conduite de centrales nucléaires.
- Conduite de barrages.

Classification des pannes

- Pannes catastrophiques
  Elles sont inacceptables

- Pannes non catastrophiques
  Elles sont acceptables.
  Notion de système à défaillance saines
  ("Failsafe")

Techniques pour la construction de systèmes sûrs

L'évitement des fautes ("Fault avoidance")

L'ensemble des techniques de conception, de fabrication qui permettent de produire des composants informatiques de très bonne qualité (très fiable).

Le composant ne doit pas tomber en panne

- bonne conception
- bonne fabrication (génie logiciel)

  => peu d'erreurs.

La tolérance aux fautes ("Fault tolerance")

L'ensemble des techniques de conception des systèmes qui continuent de fonctionner même en présence de la panne de l'un de leurs composants.

L'ensemble de l'architecture, considérée comme un tout, continue par l'utilisation de redondances de rendre le service attendu en dépit de l'existence de fautes.

Techniques pour la validation de systèmes sûrs

L'élimination des fautes

L'ensemble des techniques permettant de minimiser le nombre de fautes résiduelles présentes dans un système (par le test).

Techniques de validation, de tests
=> moins d'erreurs.

La prévision des fautes ("Fault measurement") ("Dependability evaluation")

L'ensemble des techniques de l'évaluation prévisionnelle de la sûreté de fonctionnement (de l'existence de fautes).

L'ensemble de l'architecture considérée comme un tout continue par l'utilisation de redondances de rendre le service attendu en dépit de l'existence de fautes.

Les composantes de la sûreté de fonctionnement

- La fiabilité -

Probabilité pour qu'un système soit continûment en fonctionnement sur une période donnée (entre 0 et t).

- La disponibilité -

Probabilité pour qu'un système soit en fonctionnement à un instant t donné.

- La maintenabilité -

Probabilité pour qu'un système en panne à l'instant 0 soit réparé à l'instant t.

- La sécurité -

Probabilité pour qu'un système soit continûment en fonctionnement non catastrophique sur une période donnée (entre 0 et t).

Sûreté de fonctionnement et systèmes répartis

- La sûreté de fonctionnement d'un système réparti doit être étudiée soigneusement

Architecture de n sites en coopération
- Si le taux de panne d'un site est λ
- Chaque site est indispensable
- Le taux de panne du système est $n\lambda$.

- Au contraire une organisation efficace d'un système réparti (techniques de tolérance aux pannes) atteint des niveaux de sûreté de fonctionnement qu'aucune approche d'évitement ne peut atteindre (fiabilité, disponibilité, sécurité) à un coût acceptable.

I

Classification des pannes

Modèle des systèmes répartis étudiés

Un composant (matériel ou logiciel) est considéré comme correct s'il se comporte **de manière consistante avec ses spécifications.**

Un modèle formel (ex: automate) décrit le comportement correct du composant. Pour toute situation on connaît :

. Un ensemble possible d'événements entrants.

. Les traitements à réaliser

. Les ensembles de messages produits en résultat

. Les contraintes de délais de réponse (dans de nombreux cas cette donnée est fondamentale).

Un composant en panne ne se comporte pas selon ses spécifications de comportement et de performances

Conséquences sur les équipements matériels

Processeurs corrects

Exécution du jeu d'instruction respecté.

Respect de l'intégrité des données en mémoire.

Temps de traitement conformes aux spécifications.

Réseau de communication correct

- Topologie quelconque permettant tous les échanges nécessaires à l'application.
- Délai de transmission des messages conformes aux spécifications.

Horloges physiques correctes

. Dérive bornée (par rapport à un temps universel utilisé à titre de comparaison)

Ex. : u, v dates universelles

c(u), c(v) valeurs lues

r dérive maximum

$$(1 - r)(u - v) < c(u) - c(v) < (u - v)(1 + r)$$

Panne franche ("Crash")

Une fois le composant en panne franche il cesse immédiatement et de façon indéfinie de répondre à toute sollicitation ou de générer de nouvelles requêtes (jusqu'à une réparation).

Une panne franche est **une panne permanente.**

Ex. : .Panne franche de processeur.
. Coupure de voie physique.
. Certains types de programmes erronés (exemple boucle)
. Système d'exploitation interbloqué.

Distinction

Système silencieux sur panne "Fail-silent"

En panne silencieuse le système ne produit plus aucune sortie.

Système stoppé sur panne "Fail-stop"

Modèles de systèmes relativement au temps

Systèmes synchrones

Idée de base

Deux systèmes ne peuvent se mettre à agir à des vitesses relatives non prévues.

. Les délais de transmission des messages sont bornés (par une valeur D).

. Il existe une borne supérieure pour le temps d'exécution d'une étape par un processus.

. Les horloges matérielles ont une dérive bornée.

Hypothèses temporelles synchrones

La réponse à une sollicitation s'effectue toujours dans un délai borné ou pas du tout.

Systèmes asynchrones

On ne connaît pas de borne au temps de réponse à une requête qui peut-être arbitraire.

Aucune hypothèse temporelle n'est formulée.

Détecteurs des pannes franches.

Les hypothèses synchrones permettent l'utilisation de délais de gardes pour la construction de détecteurs de pannes par scrutation d'un processus.

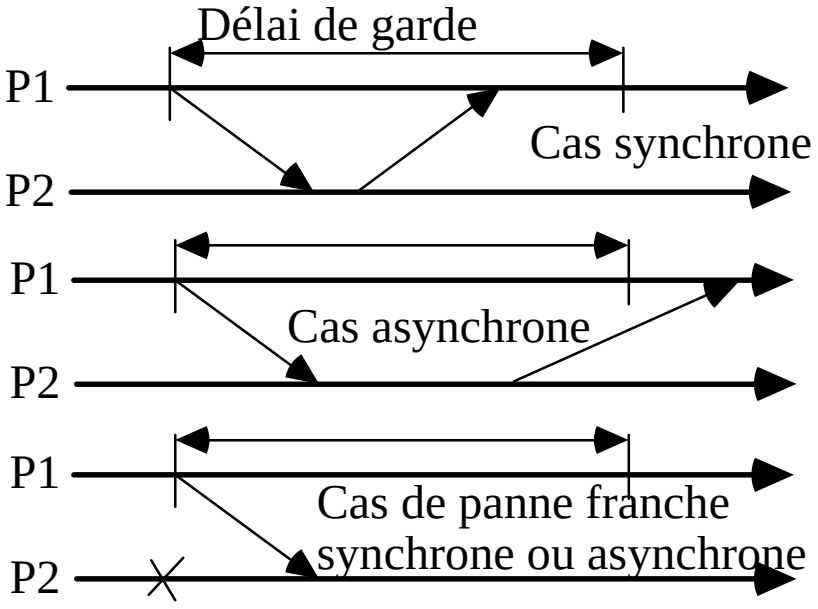
Cas des systèmes synchrones

Si le réseau ne perd pas de messages cette solution détecte les pannes franches.

Si le réseau est soumis à des pertes de messages cette solution est probabiliste.

Cas des systèmes asynchrones

On ne peut pas distinguer le cas d'un processus extrêmement lent de celui d'une panne franche.



Propriétés d'un détecteur de pannes franches.

Idée de la complétude ("Completeness")

Un processus en panne franche doit être détecté en panne.

Idée de la précision ("Accuracy")

Un processus correct ne doit pas être considéré en panne franche.

Raffinement des propriétés des détecteurs de pannes franches (Chandra et Toueg).

Complétude ("Completeness")

Complétude forte

Inévitablement tout processus en panne franche est suspecté de manière permanente par tout processus correct

Complétude faible

Inévitablement tout processus en panne franche est suspecté de manière permanente par un processus correct.

Précision ("Accuracy")

Précision forte ("Strong Accuracy")

Aucun processus correct n'est suspecté avant de tomber en panne franche.

Précision faible ("Weak Accuracy")

L'un des processus correct n'est jamais suspecté avant de tomber en panne franche.

Précision forte inévitable ("Eventual strong Accuracy")

Il existe une date au delà de laquelle aucun processus correct n'est suspecté avant de tomber en panne franche.

Précision faible inévitable ("Eventual weak Accuracy")

Il existe une date au delà de laquelle l'un des processus correct n'est jamais suspecté avant de tomber en panne franche.

Catégories de détecteurs de pannes franches selon Chandra et Toueg

Combinaison des quatre niveaux de précision et des deux niveaux de complétude.

Complétude \ Précision	Forte	Faible	Inévitablement Forte	Inévitablement Faible
Forte	Parfait P	Fort S	Inévitablement Parfait ◇ P	Inévitablement Fort ◇ S
Faible	Q	Faible W	◇ Q	Inévitablement Faible ◇ W

Avant Chandra et Toueg

Définition de problèmes de sûreté de fonctionnement résolus sous hypothèses synchrones ou asynchrones.

Après Chandra et Toueg

Définition de problèmes de sûreté de fonctionnement résolus sous hypothèses de fonctionnement des détecteurs de pannes.

Panne transitoire ou intermittente "Transient, Intermittent, Omission failure"

En réponse à un événement en entrée un composant ne délivre **jamais** la réponse attendue (le composant ne peut fournir son service habituel pendant une certaine période => perte de quelques données).

Ultérieurement il peut fonctionner à nouveau de façon correcte.

Il n'y a pas déviation par rapport aux spécifications sur ces autres réponses.

Distinction possible

Panne transitoire

Apparaît une seule fois puis disparaît.

Panne intermittente

Apparaît plus ou moins périodiquement.

Exemples. :

. Perte de messages dues au bruit.

. Destruction de message pour éviter la congestion.

. Destruction d'activités pour éviter l'interblocage ou les problèmes de contrôle de concurrence.

Panne temporelle
"Timing, Performance failure"

. Une sortie correcte associée à une requête entrante se manifeste de façon incohérente avec les spécifications temporelles.

- Trop tard ou jamais.

- Trop tôt.

Le cas le plus fréquent est celui d'une manifestation trop tardive.

Ex. :

. Surcharge d'un processeur.

. Horloge trop rapide.

. Délai de transmission trop long.

Panne quelconque ou byzantine ("Malicious, byzantine Failures")

Tout comportement s'écartant des spécifications (principalement en ce que les résultats sont non conformes) est qualifié de comportement byzantin.

On distingue quelquefois :

a) Fautes byzantines "naturelles"

Ex. : . Erreur physique non détectée (sur une transmission de message, en mémoire , sur une instruction).

. Erreur logicielle amenant une non vérification des spécifications.

b) Fautes byzantines "malicieuses"

Ex. : . comportement visant à faire échouer le système (sabotage, virus ....).

Classification complémentaire des pannes byzantines

Le cryptage (authentification ou signature) des messages entraîne une résistance aux pannes byzantines bien meilleure (surtout pour ce qui concerne les modifications quelconques qui pourraient être effectuées sur les messages du fait de la transmission via des sites malicieux).

On distingue donc parfois:

1) La classe des fautes byzantines précédemment décrites (pour lesquelles les communications sont non authentifiées).

2) La classe des fautes byzantines qui apparaissent malgré les signatures ("pannes byzantines authentifiées").

Hiérarchisation des classes de pannes

Les 4 classes sont hiérarchisées.

Panne franche:

Pas de réponse à une entrée

=> Panne transitoire

Panne transitoire:

Un délai de réponse infini.

=> Panne temporelle

Panne temporelle:

Non respect d'une échéance (spec)

=> Panne quelconque

Tolérance à une classe de pannes

On réalise des composants pour tolérer l'une des classes de pannes précédentes.

Restent non tolérées les pannes de la classe supérieure.

Cas les plus fréquents:

Tolérance aux pannes franche

Tolérance aux pannes d'omission.

II

Les différentes catégories

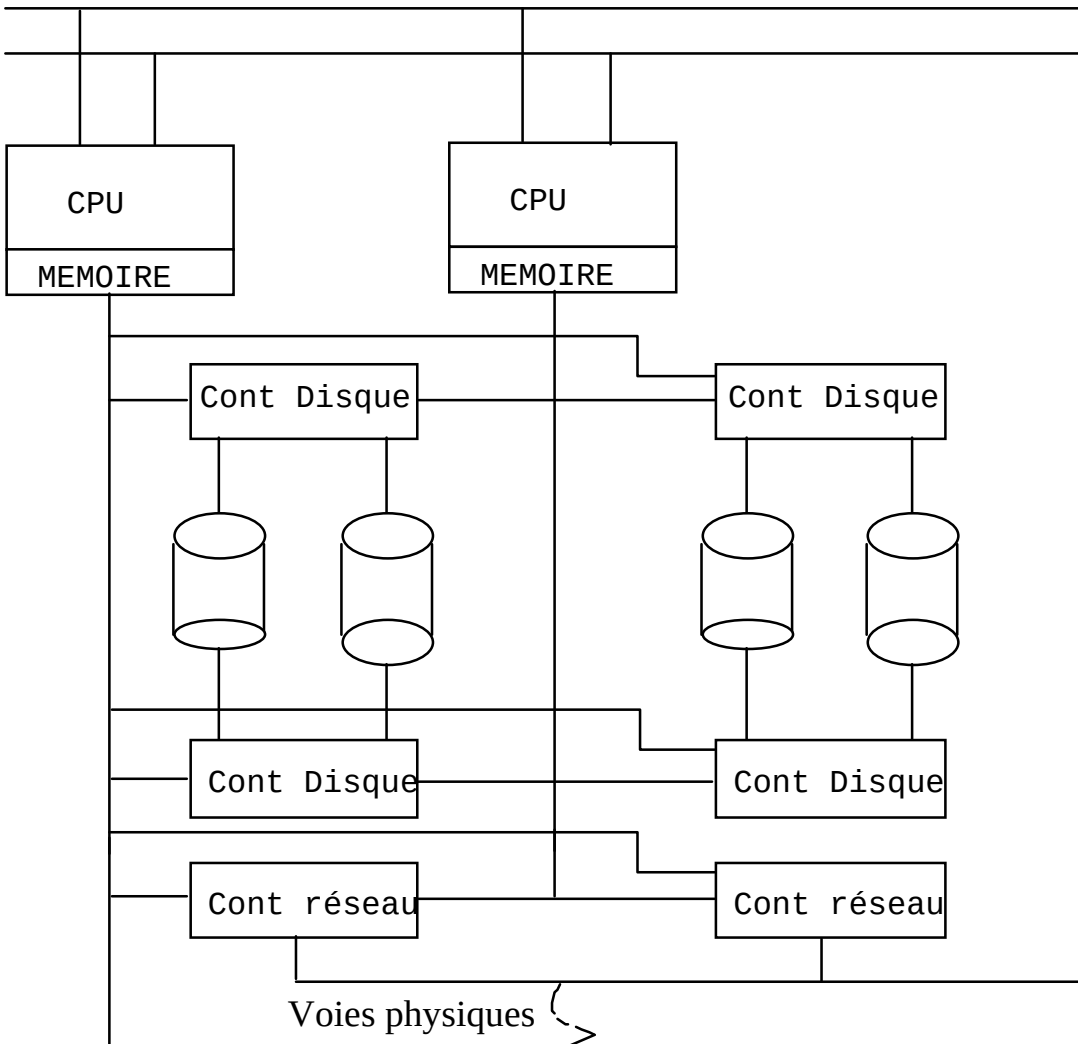
de redondance

Rappel : Architectures à redondances matérielles

Exemple type : Architecture TANDEM

- Tolérance à une panne franche matérielle destiné aux applications de gestion

- Système orienté disponibilité



Les différents types de redondances

Nombreuses propositions
Nombreux points de vue

Techniques de recouvrement d'erreurs ("Error recovery")

1) Existence d'un détecteur de panne.
2) D'un état erroné on peut retrouver un état correct puis délivrer un service correct.

Reprises arrières
Redondances temporelles
("Backward Recovery")

- Pour un composant soumis à des pannes transitoires il est courant de tenter de corriger cette panne par un nombre fixé de tentatives successives.

Nécessite la pose de points de reprise.

Z Cette technique est utilisée assez systématiquement pour les serveurs uniques et ce n'est qu'après l'échec de cette approche que l'on déclare une panne non temporaire.

Reprises avants / Traitement des exceptions / Poursuite ("Forward Recovery")

- Pour un composant soumis à des pannes il est courant de tenter de traiter cette panne en recherchant un état de cohérence du système n'ayant jamais existé (futur) ou n'ayant pas existé dans un passé récent (qui s'apparenterait à une reprise).

La reprise avant évite la pose de points de reprise mais nécessite de déterminer l'étendue des dommages causés aux système par la faute jusqu'à la détection de la défaillance.

Exemple : traitant/récupérateur d'exception

Techniques de compensation ou de masquage d'erreurs

Un état erroné comporte des redondances permettant au moyen d'un algorithme rapide de délivrer un service correct.

Redondances de données

- Pour une donnée soumise à des erreurs (stockage, transmission) il est d'usage de rajouter des informations de redondance selon un code correcteur d'erreur qui permet de corriger certaines erreurs.

Redondances spatiales ou redondances de groupes

- Un groupe de serveurs redondants g en redondance spatiale est conçu pour tolérer la panne de certains de ses membres.

En dépit de la panne à un certain niveau de certains membres de g, le service offert du point de vue global par g continue en masquant le niveau de panne visé.

Par exemple on observe des pannes quelconques si l'on masque les pannes temporelles, transitoires et franches.

- Éventuellement certaines performances sont dégradées.
- Certains membres du groupe g clairement détectés en panne sont isolés et retirés du groupe de serveurs redondants.

=> on reconfigure le groupe

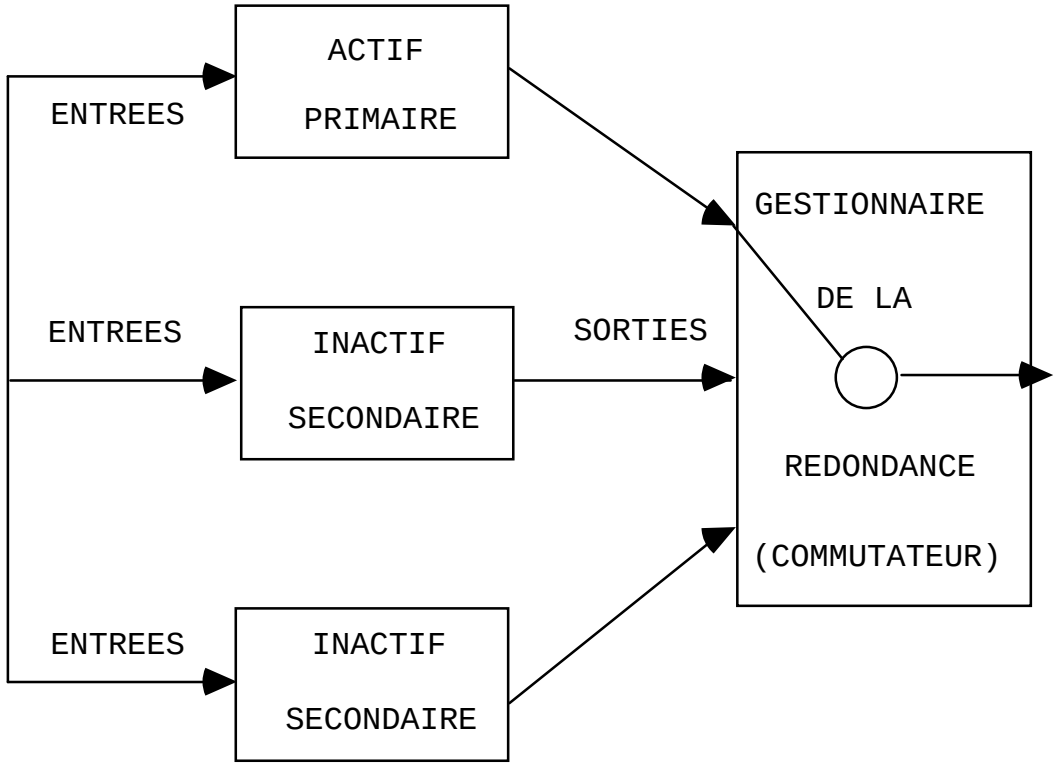
=> jusqu'à ce que la reconfiguration qui maintienne un service acceptable devienne impossible.

Différentes redondances spatiales

Redondance passive ("Standby redundancy") ("Primary backup")

Objectif poursuivi : tolérance aux pannes franches de calculateurs.

- Un seul des composants réalise effectivement les traitements et est affecté aux sorties (le primaire).
- En cas de panne du primaire l'un des calculateurs inactifs (secondaire) est sélectionné et activé pour prendre en charge le service.



Problèmes de synchronisation en redondance passive

- Problème de détection de panne du primaire.
- Problème de détermination d'un nouveau primaire parmi les alternants.
- Pour le nouveau primaire il faut reconstituer un contexte d'exécution correct.

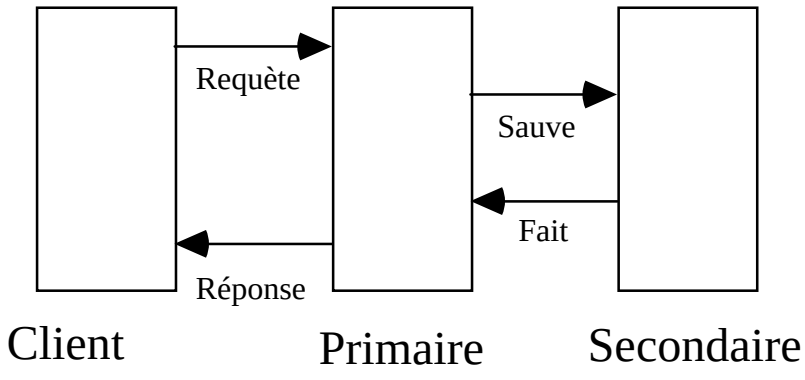
Solutions possibles

. Recopie périodique d'informations de reprise constituées par le primaire pour les secondaires.

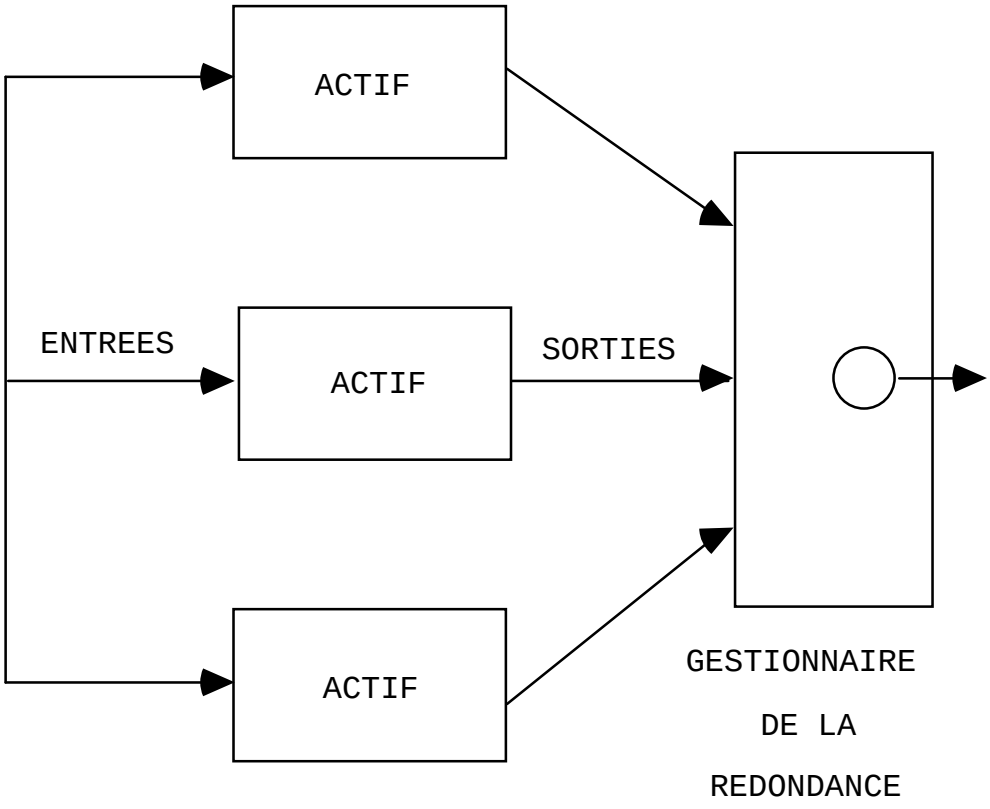
Périodes statiquement prédéterminées

Points de reprise applicatifs.

. Rééxecution des services fournis depuis le dernier point de reprise.



Redondances actives ou dynamiques ("Active redundancy")

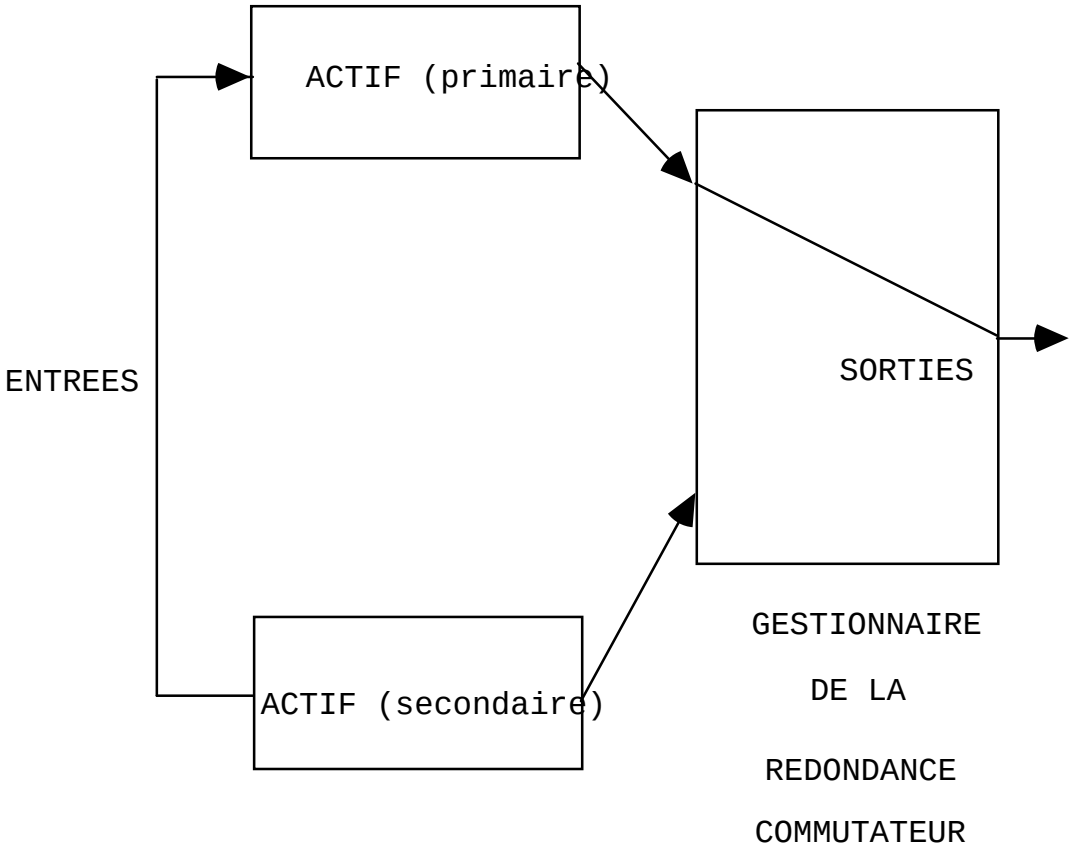


- Dans la redondance active tous les composants réalisent les traitements.

- Le gestionnaire de la redondance traite les sorties pour tolérer différentes classes de panne des serveurs.

Redondance sélective active

Tolérance des pannes franches



- Un seul serveur est affecté aux sorties

- En cas de panne du primaire le secondaire prend le contrôle (avec le contexte d'exécution complet de l'activité).

Problèmes de synchronisation en redondance sélective active

- Tout le monde doit recevoir les entrées en diffusion.

- Lors d'un basculement, l'alternant actif doit être réélu.

- Lors d'un basculement, le contexte de l'alternant actif doit être cohérent avec celui laissé par l'ancien actif: besoin d'une technique pour traiter dans le même ordre et exhaustivement les mêmes données.

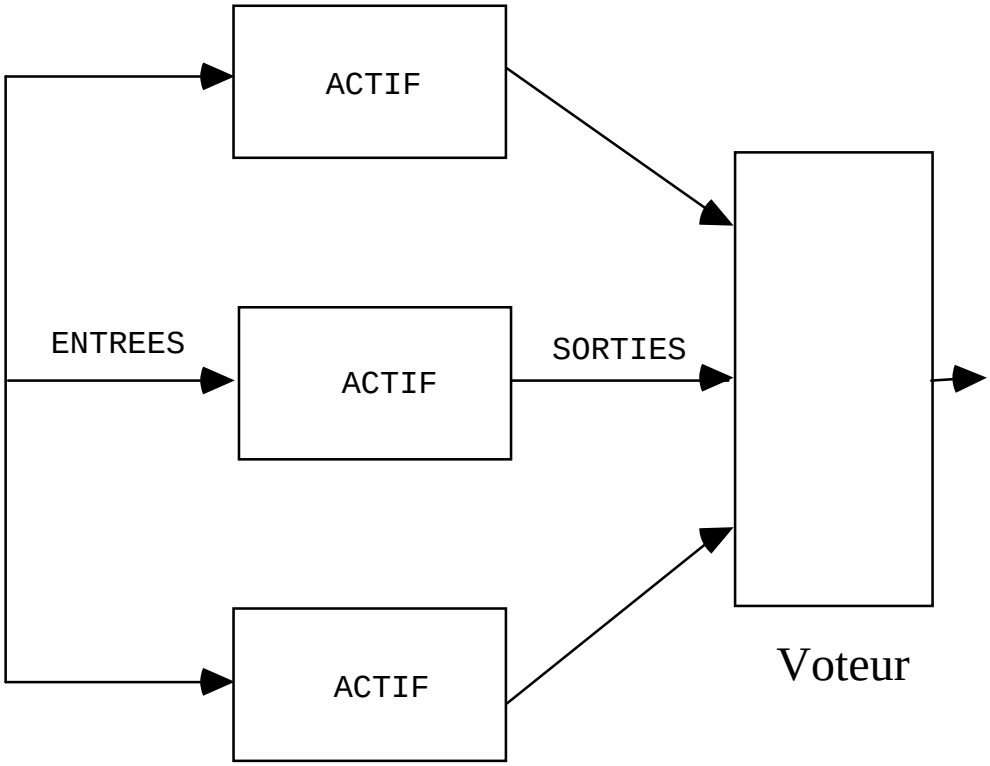
Diffusion ordonnée totalement.

Remarque :

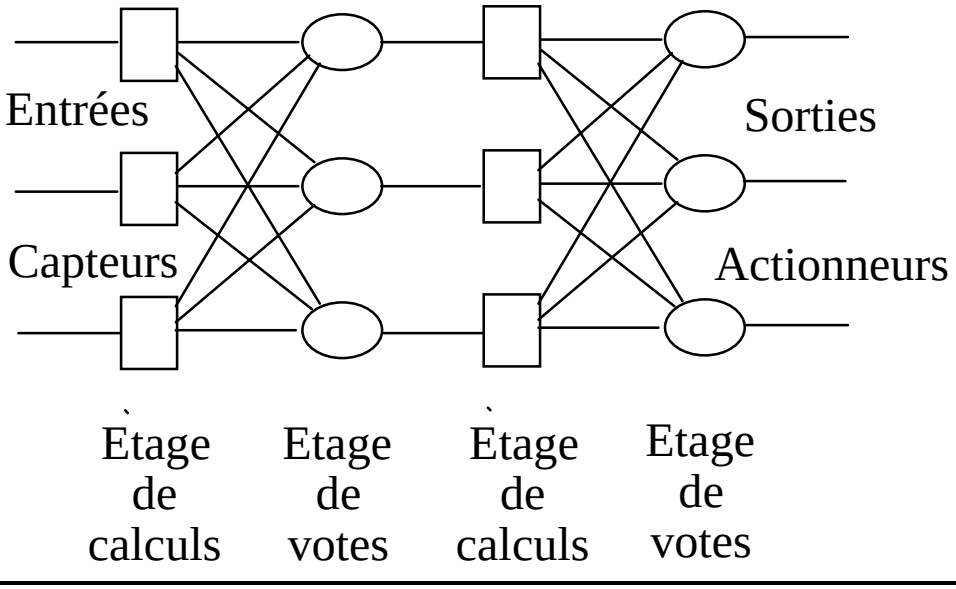
Le gestionnaire de la redondance peut-être plus complexe qu'un simple commutateur. Quand les deux composants sont actifs il peut choisir d'utiliser les résultats de l'un ou de l'autre selon les sites de résidence.

Redondance massive

Tolérance des pannes quelconques.



TMR "Triple Modular Redundancy"



Système TMR avec réplication des voteurs

Problèmes de synchronisation en redondance massive

- Tout le monde doit recevoir les entrées en diffusion dans le même ordre pour traiter les données dans le même ordre et fournir les résultats dans le même ordre aux voteurs.

- La synchronisation entre les productions de résultats doit permettre la réalisation du vote majoritaire.

Remarques:

Si l'on veut simplement tolérer des pannes temporelles on peut prendre la réponse disponible le plus rapidement.

Si l'on veut tolérer des pannes quelconques on doit voter pour exclure aussi bien les sorties non majoritaires que celles qui apparaissent trop tardivement.

Tolérance aux pannes logicielles

Si les calculateurs redondants ont la même panne le système de redondance massive ne fonctionne pas (de même que les solutions de reprise arrière).

Notion de **panne de mode commun**

Typiquement les pannes logicielles.

Solution: la **diversification** (fonctionnelle)

"N-Version programming"

- Diversification/isolement des équipes
  Spécifications détaillées.
  Développement des codes.
- Diversification des processeurs
- Diversification des systèmes.
- Diversification des compilateurs.

Variantes

- Dans le cadre des techniques de redondances temporelles
  ("**recovery blocks**" Randell)
- Dans le cadre des techniques de redondance massive.

Solutions coûteuses et difficiles à réaliser

III

Les problèmes fondamentaux des systèmes répartis tolérants les pannes

Rappel des différentes étapes d'un mécanisme de tolérance (1)

Les principaux mécanismes de la tolérance reposent sur l'existence de **services redondants** dont les éléments peuvent tomber en panne.

a) Il faut tout d'abord détecter les pannes d'un ou plusieurs serveurs.

b) Si l'on peut formuler une hypothèse de panne transitoire et si les contraintes de l'application sont compatibles avec les techniques de recouvrement d'erreur (pas trop temps réel fortement contraint) il faut appliquer une technique de rééxécution.

c) Si les redondances temporelles sont insuffisantes il faut mettre en place des redondances spatiales.

=> décider de la suite des situations d'appartenance au groupe de serveurs redondants.

Les situations successives résultent:

- des pannes des membres
- des réinsertions de composants ou des adjonctions de serveurs nouveaux.

Rappel des différentes étapes d'un mécanisme de tolérance (2)

d) Il faut assurer des transmissions fiables vers des groupes de composants redondants (diffusion d'informations de chacun des clients vers le groupe de serveurs redondants implémentant un service).

Ces diffusions doivent être réalisées sous les différentes hypothèses de pannes et satisfaire des propriétés d'ordre.

e) Pour la redondance massive il faut rechercher un consensus sur une valeur calculée n fois (vote réparti).

f) Il faut éventuellement tolérer les pannes temporelles (satisfaction de contraintes temporelles pouvant être sévères) => l'ordonnancement temps réel réparti des tâches.

Point essentiel pour la détection des pannes temporelles et pour la satisfaction des contraintes temps réel réparti:

=> l'existence d'une **synchronisation des horloges** entre les différents sites.

Rappel des différentes étapes d'un mécanisme de tolérance (3)

g) Si la programmation de l'application organise des données réparties partagées sur différents sites il faut assurer le **contrôle de l'accès concurrent** aux données (maintien de la cohérence) pour des **données dupliquées**.

h) Si la programmation de l'application comporte encore des sites centraux (redondances sélectives) il faut prévoir la **défaillance de ces serveurs**.

D'ou une liste de problèmes types à résoudre pour une algorithmique répartie de la tolérance aux pannes

LA DÉTECTION DE COMPORTEMENT FAUTIF

Notion de **composant autotestable**:
un composant qui incorpore son propre logiciel de diagnostic => qui fait passer le plus vite possible un composant de l'état de **faute latente** à l'état de **faute détectée**.

Existence de très nombreuses techniques

Quelques exemples

- Utilisant des **programmes de test**

Détection **hors ligne** (diagnostics).
Détection **en ligne** (détection continue).
Chien de garde=>surveillance mutuelle.

- **Détection des erreurs de données**

Codes détecteurs d'erreur.
Assertions/Tests d'acceptance.
Vote entre plusieurs alternants.

- **Détection des erreurs d'enchaînement**

Protection (anneaux, domaines).
Observation de points spécifiques de l'exécution => comparaison à un référentiel.
Signature de séquences
=> comparaison à une tabulation des séquences valides.

Problème classique de l'univers réparti.

Pour une application coopérative faisant intervenir une architecture quelconque de clients et de serveurs (éventuellement redondés):

=> Comment déterminer des points de reprise "cohérents" à une fréquence optimale.

=> Comment assurer si nécessaire la journalisation des messages en transit sur les canaux de communications.

=> Comment effectuer la reprise arrière en cas de panne détectée.

Difficile pour obtenir des performances satisfaisantes

PROTOCOLE D'APPARTENANCE A UN GROUPE

Objectif : Assurer que tous les usagers ayant à connaître la situation d'un groupe de serveurs atteignent un consensus sur la composition du groupe.

- La perception de cette composition est relative à des **communications de groupes** (le protocole d'appartenance à un groupe est en général utilisé conjointement avec un protocole à diffusion).

- Dans un tel protocole on admet souvent que le temps est divisé en **époques** ou la composition du groupe est fixe et identique pour tout le monde.

- A l'intérieur d'une même époque les communications en diffusion atteignent la même liste de processeurs ou échouent ce qui peut advenir en période de changement de liste.

LA DIFFUSION ET LE CONSENSUS

Objectif : Ces deux problèmes précédents sont des variantes peu différentes consistant à faire s'accorder différents composants d'un groupe sur une valeur

- Valeur diffusée par un site.
- Valeur votée après un calcul en redondance massive.

- La valeur est utilisée comme **un signal** pour déclencher un traitement (valeur binaire)

Exemple du problème de validation dans la mise à jour des données ("commit")

- La valeur est utilisée comme **une donnée quelconque**.

Exemple du problème de redondance massive sur des données calculées comme des valeurs à envoyer sur des actionneurs.

LE PROTOCOLE DE SYNCHRONISATION D'HORLOGES

Objectif : Assurer que des horloges situées sur des sites distincts fournissent une datation absolue des événements avec une incertitude définie.

On distingue dans ce contexte deux sous-problèmes :

- Assurer que différents sites arrivent à **démarrer avec la même heure absolue** (au même moment avec une incertitude connue).

- Maintenir aussi longtemps que nécessaire **les différentes horloges** dans une variation relative connue.

. En contrôlant la dérive relative

Solutions par échange de messages.

Solutions par asservissement sur une horloge hertzienne.

LE PROTOCOLE D'ÉLECTION

Objectif : Lorsqu'une solution à un problème est basée sur l'existence d'un site **coordinateur unique** la panne du coordinateur doit être tolérée.

Cas des solutions en redondance sélective passive et sélective active.

Le protocole d'élection vise à désigner un et un seul coordinateur remplaçant.

CONCLUSION

Architecture de systèmes répartis en vue de la tolérance aux pannes
Nécessité de fournir deux catégories de services.

1 Les services standards utilisés dans des systèmes répartis: micro-noyaux, systèmes d'objets répartis

- Gestion des ressources (mémoire, processeur/ordonnancement..)
- Désignation, liaison
- Création des objets, migration,
- Réalisation des interactions de base (IPC, RPC légers/distants)
- Synchronisation **…** etc

La sûreté n'est pas leur objectif principal

La sûreté/tolérance aux pannes de ces algorithmes est fondamentale car ils sont utilisés dans un système dont la sûreté doit être excellente.

2 Algorithmes utilisés dans les systèmes tolérants les pannes pour la tolérance.

Exemples vus:

- Détection de panne
- Reprises arrière
- Élection
- Diffusion fiable
- Gestion des groupes
- Vote réparti
- Synchronisation d'horloges
- Copies multiples
- etc....

- La tolérance aux pannes des solutions présentées dans un système réparti pour la sûreté de fonctionnement est un problème essentiel.

- Certains de ces algorithmes doivent être étudiés pour tolérer tout type de panne.

Bibliographie

1 R. Strong "Problems in fault-tolerant distributed systems", Publication IEEE ISBN 135-/85/0000/0300s01.00

2 F. Christian, "Issues in the design of highly available computing systems", IBM Research Report RJ 5856 (58907) 7/10/1987

3 F. Christian, "Questions to ask when designing or attempting to understand a fault-tolerant distributed system", IBM Research Report RJ 6980 (66517) 4/8/1989

4 J.C. Laprie, B. Courtois, M.C. Gaudel , D. Powell "Sûreté de fonctionnement des systèmes informatiques matériels et logiciels", AFCET Dunod Informatique 1989