

L'atelier AMI

Table des matières

L'atelier AMI.....	209
Introduction	213
1. Présentation générale.....	214
2. Environnement réseau et système.....	216
3. Etat d'implémentation de AMI	218
4. Les langages de communication.....	219
Le langage LDF et CAMI-LDF.....	220
Le langage LDE et CAMI-LDE	220
Le langage LDR et CAMI-LDR.....	221
Le langage LAQ	222
5. Réalisation de l'application d'interaction utilisateur	223
5.1. Introduction.....	223
5.2. Généralités.....	224
5.3. Niveau Plate-forme GUS.....	225
5.3.1. Le GUS dans MACAO	225
5.3.2. Gestion des utilisateurs	226
5.3.3. Gestion des formalismes	227
5.3.4. Gestion des services	228
5.4. Niveau Plate-forme GSV	229
5.4.1. Généralités	229
5.4.2. Réalisation de la connexion physique	229
5.4.3. La couche transport	232
5.4.4. Les couches session, présentation	232
5.5. Niveau interface	233
5.5.1. Le gestionnaire des dialogues.....	233
5.5.2. Le gestionnaire des fenêtres.....	233
5.5.3. Le gestionnaire de graphes	234
5.6. Niveau application.....	235
5.6.1. L'éditeur de graphes	235
5.6.2. L'interface utilisateur des services.....	242
5.6.3. Interface utilisateur d'administration et extensibilité	243
6. Réalisation sous Unix	244
6.1. Niveau extension système	244
6.1.1. Système - Répartition.....	244
6.1.2. Système - Logistique.....	250
6.2. Niveau plateforme GUS.....	251
6.2.1. Représentation des données personnelles	252
6.2.2. Représentation des données d'administration.....	253

6.2.3.	Le mécanisme client-serveur	257
6.3.	Niveau Plate-forme GSV	260
6.3.1.	La couche transport	262
6.3.2.	La couche session: le communicateur	264
6.3.3.	La couche présentation: le présenteur	265
6.4.	Niveau interface: le pilote.....	265
6.4.1.	La liaison entre la structure d'accueil et le pilote	265
6.4.2.	La liaison entre le pilote et la structure d'accueil	266
6.4.3.	Définition d'un pilote générique pour l'atelier AMI	267
6.5.	Niveau applications.....	269
6.5.1.	Système Expert MIAMI	269
6.5.2.	Intégration d'applications	270
6.6.	Les applications intégrées.....	272
6.6.1.	ARP	273
6.6.2.	COMBAG.....	273
6.6.3.	CHP.....	273
6.6.4.	GrapheCouverture	273
6.6.5.	ECS	274
6.6.6.	GreatSPN	274
6.6.7.	Le système expert MIAMI	274
6.6.8.	Formalisme RPAS	275
6.6.9.	TAPIOCA	275
6.6.10.	RdP/TAGADA.....	275
6.6.11.	Intégrations en cours	276
7.	Utilisation de l'atelier.....	277
7.1.	Introduction.....	277
7.2.	Mode autonome.....	277
7.3.	Mode connecté de Macao.....	279
7.4.	Session - Présentation	280
7.5.	Application	283
7.6.	Multi-sessions	285
7.7.	Déconnexion	287
7.8.	Reprise de connexion	287
8.	Conclusion	288
Conclusion		291

Introduction

Cette partie présente la première version d'un atelier de spécification (AMI: **A**telier de **M**odélisation **I**nteractif [Bernard and Mounier, 1989, Bernard, et al., 1988]) qui est basé sur les caractéristiques et l'architecture décrite dans la partie II. Cet atelier prend en compte les besoins de nos utilisateurs qui ont été définis dans la partie I.

L'atelier AMI utilise un méta-modèle de représentation des formalismes basé sur les graphes. Ce prototype intègre actuellement des formalismes liés à la théorie des **réseaux de Petri**. Pour ces formalismes, l'atelier AMI réunit une base d'outils interactifs de création, manipulation, transformation, d'analyse et de vérification de propriétés de réseaux de Petri jusqu'à la génération de prototypes (code Ada).

L'atelier AMI est utilisé dans le cadre des projets européens MERCHANT et IRENA pour des applications spécifiées à partir de réseaux de Petri. Il est utilisé de manière expérimentale dans les universités de Hambourg et de Toulouse (IRIT) respectivement pour des applications d'algorithmique distribuée et de théorie de graphes. L'atelier AMI permet de spécifier des problèmes télématiques par un formalisme de haut niveau utilisé par la SLIGOS (les réseaux RPAS) [Trèves, 1988]. Ces réseaux RPAS sont traduits en réseaux de Petri puis analysés.

L'atelier AMI, à travers le projet MARS, offre la perspective de nouveaux outils de modélisation, de validation et de preuve induisant de nouvelles méthodes de travail pour les utilisateurs. Les outils d'analyse peuvent suivre rapidement les avancées théoriques car l'atelier offre au développeur un cadre de travail lui permettant de se consacrer uniquement à son algorithme tout en bénéficiant de l'environnement du projet MARS et des outils accumulés par les autres chercheurs.

Après une présentation générale des outils intégrés dans l'atelier et des choix matériels, nous détaillons la réalisation de l'application d'interaction utilisateur dans l'environnement Apple (Macintosh) et du reste de l'architecture sous Unix. Un langage de communication interne (CAMI: Communication dans AMI) offre une présentation uniforme des différents langages que nous avons définis dans la partie II (Langage de description de formalismes, d'énoncés, de questions et de résultats). Nous présentons l'utilisation de l'atelier sous la forme d'un exemple complet. Cela nous permettra de mettre en évidence les solutions apportées aux problèmes de connexion/déconnexion dans un environnement distribué, la gestion multi-sessions et les liens avec les programmes d'application.

1. Présentation générale

L'atelier AMI a été au départ orienté vers les formalismes issus des réseaux de Petri. Ceux-ci constituent un modèle complet pour la spécification, la validation de systèmes parallèles. Ils sont très employés aussi bien dans l'industrie que dans la recherche en raison de leur adéquation à la modélisation de systèmes parallèles, de la lisibilité de leurs représentations graphiques et de l'étendue des résultats théoriques obtenus. Leur utilisation repose cependant sur la disponibilité d'outils logiciels [Feldbrugge, 1987]. Une première catégorie d'outils de base regroupe les logiciels de construction et édition de réseaux de Petri, tandis que la seconde est composée d'outils de simulation et analyse [Jensen, 1987].

La complexité croissante des systèmes étudiés et les exigences des modélisateurs ont amené à étendre les types de réseaux de Petri ordinaires à des réseaux de Petri de haut niveau. Les résultats théoriques sur ces modèles sont partiels et évoluent très vite. Les utilisateurs disposent d'une multitude d'algorithmes d'analyse de plus en plus puissants pour les réseaux ordinaires [Stark, 1985], colorés [Jensen, 1984] [Colom, 1986], à files [Memmi, 1985], à prédicats [Trèves, 1987, Vautherin, 1986, Wheeler, 1985], stochastiques [Chiola, 1987, Florin, 1986].

Un ensemble d'algorithmes d'analyse de réseaux réguliers [Haddad, 1986] que nous avons développé a été facilement intégré dans l'atelier AMI. D'autres outils, conçus dans d'autres universités pour d'autres environnements ont aussi été intégrés et profitent de l'interface utilisateur de l'Atelier pour l'introduction et les visualisation des résultats [Finkel, 1990, Trèves, 1987].

Les travaux antérieurs sur les réseaux de Petri stochastiques ont débouché sur la réalisation de GreatSPN, un outil de modélisation, d'analyse et d'évaluation de performance basé sur les réseaux de Petri stochastiques généralisés [Chiola, 1986, Chiola, 1987, Chiola, 1989]. Ce système, qui est l'un des logiciels les plus complets et les plus performants dans ce domaine, sert de base à plusieurs projets européens, dont IMSE. Nous avons intégré, sans le modifier, des programmes de calcul issus de GreatSPN dans l'atelier AMI. Ainsi il possible d'utiliser ces mêmes programmes à partir des deux environnements.

Par ailleurs, l'utilisation du système expert MIAMI (**M**oteur d'**I**nférence pour **AMI**) [El Fallah, 1989] offre des services de simulation, de réduction de réseaux de Petri et de transformation de formalismes avec toujours les mêmes présentations d'interfaces.

Dans l'atelier AMI, la génération de code, en Ada et OCCAM, n'est automatisée que pour les réseaux ordinaires et vers des architectures particulières. De nombreux logiciels existent dans les laboratoires ou même sont commercialisés mais leur formalisme d'exploitation est toujours limité et les possibilités de communication restent très faibles. Le prototypage à partir des réseaux de Petri a plusieurs avantages: le modèle est une spécification formelle, la génération de code intervient après la vérification et l'exploitation des résultats de l'analyse structurelle et comportementale du modèle.

2. Environnement réseau et système

Dans l'atelier AMI, nous avons utilisé la modularité de l'architecture pour implanter la partie interface utilisateur sur des machines spécialisées (Macintosh) et le reste de l'atelier sur un réseau de machines Unix (extensions Berkeley + NFS). La figure 3 met en évidence cette séparation physique de l'atelier.

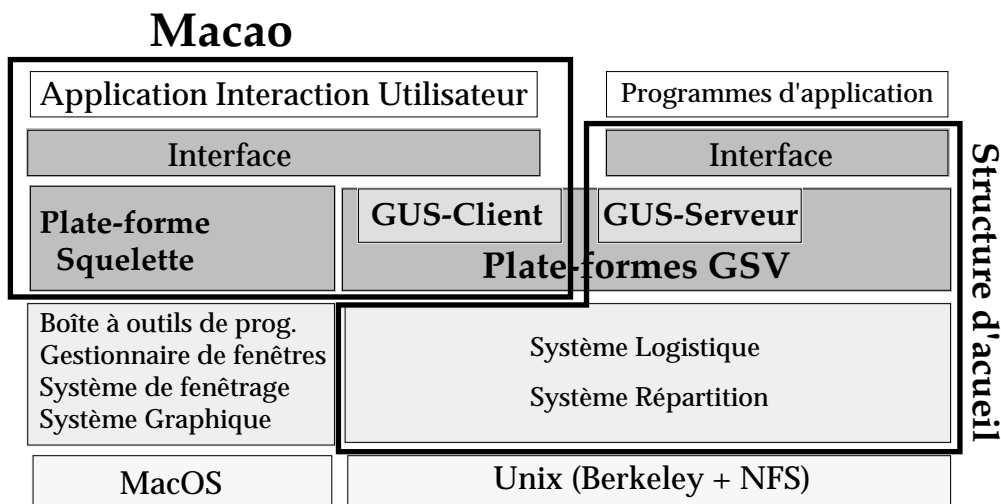


Figure 3: Séparation physique de l'architecture fonctionnelle

Architecture matérielle et environnement système

Nous avons localisé l'exécution de Macao sur un poste de travail mono-tâche, mono-utilisateur pour des raisons ergonomiques (interface utilisateur interactive et conviviale [APPLE, 1987], système d'exploitation adapté à son interface, facilité de transport et d'installation) et techniques (primitives graphiques spécialisées intégrées, spécialisation du poste de travail dédié, machine peu gourmande en puissance et en mémoire).

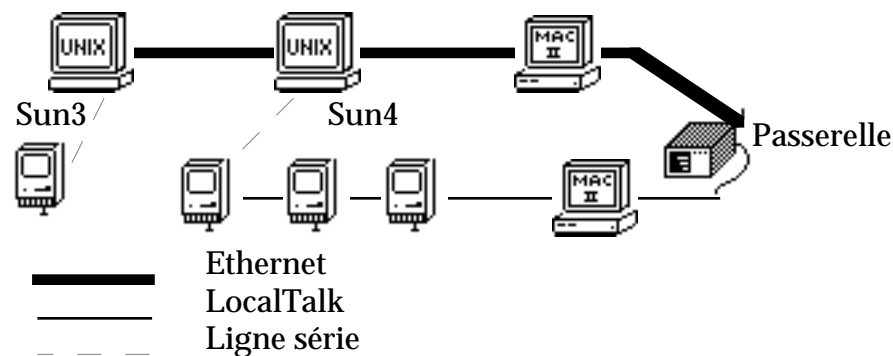
Les applications hôtes nécessitent une forte puissance de calcul et manipulent de grosses quantité d'informations. Dans un souci de performance, nous avons localisé leurs exécutions sur un parc de machines possédant un système d'exploitation *multi-tâches, multi-utilisateurs* et *multi-sites* (système UnixTM) muni d'un *système de fichiers répartis* (NFS). L'Unix Berkeley que nous utilisons nous permet d'intégrer les communications dans le réseau.

Les utilisateurs possèdent un compte Unix sur l'ensemble des machines avec une même identification (UID et GUID [Bach, 1986]). L'ensemble des fichiers des utilisateurs est accessible de toutes les machines de manière transparente par l'utilisation de NFS. Les machines supportant l'exécution de l'atelier doivent avoir les mêmes répertoires pour un même utilisateur.

Environnement réseau

Dans l'architecture choisie, des machines Unix (Sun3 et Sun4), interconnectées par un réseau Ethernet, supportent l'exécution de l'environnement et des services offerts aux utilisateurs.

Plusieurs postes de travail de type micro-ordinateur (Macintosh), connectés au réseau Ethernet par une passerelle (FastPath en l'occurrence) sont mis à la disposition des utilisateurs [Laforgue, 1988]. Chacun d'eux supporte l'exécution de l'application d'interface utilisateur. Des postes isolés peuvent être aussi utilisés.



Nous obtenons une souplesse de connexion grâce à l'utilisation, au choix, de deux protocoles (TCP/IP ou ligne série). L'utilisation de TCP/IP (par le logiciel MacTCP) permet d'accéder à l'atelier depuis n'importe quel poste de travail relié en réseau localtalk ou Ethernet. L'utilisation par ligne série permet une utilisation par modem 1200 bauds sans pénalisation de l'interface utilisateur et avec des temps de transfert raisonnables.

3. Etat d'implémentation de AMI

Nous présentons l'état d'implémentation de la version actuellement disponible de AMI (1.2). Cette version reprend toute l'architecture définie en partie II. Nous indiquons ici les points définis en partie II qui feront l'objet de la version 2.0 de AMI.

Pour permettre la mise au point des applications dans l'atelier, nous avons offert à l'utilisateur un accès direct aux applications et non pas un accès aux applications par les services. L'accès aux services n'étant pas mis en place, la communication généralisée entre application n'est pas encore réalisée.

Dans la version actuelle, les programmes d'administration ne sont pas intégrés comme cela est défini dans la partie II. De plus nous n'avons pas mis en place de gestionnaire des menus et des dialogues génériques, les enchainements des menus et des dialogues restent actuellement effectués directement par l'application d'interaction utilisateur.

Nous n'avons pas mis en place la tolérance aux pannes pour l'accès aux fichiers utilisateurs (pas de disques miroirs ni de base de donnée dupliquée...).

Le système expert n'intègre pas la notion de paquet de règles mais il fait partie de l'atelier et les applications sont écrites sous-forme de règles.

4. Les langages de communication

Nous avons défini une syntaxe unique pour tout les langages utilisés dans l'atelier. Ce langage de communication dans l'atelier AMI (CAMI Communication dans AMI) a pour caractéristique d'être rapide à analyser et indépendant des représentations internes des données en machine.

Ce langage de communication est composé de messages de la forme suivante:

NomDeCommande ([paramètre [,paramètre]]*)

Le nom de commande est composé de deux lettres majuscules pour tous les messages transitant entre Macao et la structure d'accueil. Les noms de commandes des messages CAMI transitant à l'intérieur de la structure d'accueil sont composés d'une majuscule suivie d'une minuscules.

Les paramètres sont séparés par des virgules. Ces paramètres sont soit numériques, soit énumérés (dans ce cas l'énumération commence à 1), soit alphanumérique (dans ce cas la chaîne de caractère commence par sa longueur en clair suivit de ":". Exemple "abcd" est représenté par 4:abcd).

Nous décrivons les langages de description des formalismes (LDF), le langage de description des énoncés (LDE), de résultats (LDR), le langage d'arbre de questions (LAQ). Dans chaque paragraphe, nous présentons les langages décrits dans la partie II et leur implémentation en CAMI (ex: LDF et CAMI-LDF). La nomenclature des commandes CAMI fait partie d'une documentation interne dont les paragraphes suivants sont extraits.

Le langage LDF et CAMI-LDF

Le langage LDF:

NOM: *identificateur*

TYPE_ATTRIBUT: entier, chaîne, énuméré, ...

PROPRIETE : PROPRIETE, NOMINATION

TYPE_CLASSE : CONNECTEUR, NŒUD, INFORMATION

CLASSE_FORME: nom de forme dans la classe FORME_GRAPHIQUE

LANGUE: français, anglais, allemand, espagnol...

NomFormalisme(NOM, version, LANGUE, nomInterne)

CreerClasse(TYPE_CLASSE, NomDeLaClasse, CLASSE_FORME)

CreerAttribut(NomAttribut, TYPE_ATTRIBUT, ValeurParDéfaut, PROPRIETE)

Connexion(NomDeLaClasse, NomDeLaClasseDepart, NomDeLaClasseArrivée)

Le Langage CAMI-LDF:

DA (date en secondes du fichier)..... Date
TD ()..... Début table du domaine
OB (nœud/connecteur, n°type, "nom du type de l'objet") Description Objet
AT ("nom attribut", nomination/propriété, Unicité, Perm, ["défaut"]) description ATtribut
AU (n°type connecteur, n°type objet départ, n° type objet arrivée)..... AUtilisation de connexion
FA ()..... Fin tAble des domaines

Le langage LDE et CAMI-LDE

Le Langage LDE

NomFormalisme(nom, version, LANGUE, nomInterne)

CreerObjet(nom classe, n°objet)

CreerConnection

(nom classe, n°objet connecteur, n° nœud départ, n° nœud arrivée)

ValeurAttribut(n°objet, nom attribut, valeur)

PositionObjet(n°objet nœud, position)

PositionPointIntermediaire(n°objet connecteur, n°point, position)

Le Langage CAMI-LDE

DB ()	Début Batch
DO ("formalisme")	Formalisme
CN ("type de nœud", n°)	Créer Nœud
CA ("type de l'arc", n°, nœud départ, nœud arrivée)	Créer Arc
CT ("type attribut", n° objet appartenant, "contenu")	Créer Texte
PO (n°objet, posH, posV)	Position Objet
PI (n°arc, posH, posV)	Point Intermédiaire
FB ()	Fin Batch

Le langage LDR et CAMI-LDR

Le Langage LDR

```

DebutReponse(nom_service, [nom_option_service])
DebutEnsemble(["nom de l'ensemble"])
FinEnsemble()
ReponseTextuel("texte" [,valeurs]*)
ChangerAttribut(n°objet,nom attribut,valeur)
SupprimerObjet(n°objet)
CreerObjet(nom classe, n°objet)
CreerConnection
    (nom classe, n°objet connecteur, n° nœud départ, n° nœud arrivée)
FinReponse()

```

Le Langage CAMI-LDR

DR ()	Début Réponse
DE (["nom de l'ensemble"])	Début Ensemble
FE ()	Fin d'ensemble
RT ("ligne de texte")	Résultat Texte
XA (n°objet,"type de l'attribut","nouveau contenu")	Changer Texte
SU (n°objet)	Supprimer Objet
CN ("type de nœud", n°)	Créer Nœud
CA ("type de l'arc", n°, nœud départ, nœud arrivée)	Créer Arc

Le langage LAQ et CAMI-LAQ

Le Langage LAQ

nom : identificateur

TYPE_DE_SELECTION: [au plus un, un et un seul, au moins un, indifférent]

NomArbre(nom_application, LANGUE)

CreerQuestion(nom_service, nom_interne)

CreerOption(nom_service, nom_option_service)

OptionTransformation(nom_option_service, formalisme)

OptionSelection(nom_option_service, TYPE_DE_SELECTION)

OptionSelectionFine

(nom_option_service, TYPE_DE_SELECTION, nom_classe_objet)

Le Langage CAMI-LAQ

DQ ().....Début Question
AQ ("Libellé elt", "Libellé s-elt", type de question, [type de choix], [validation], [dialogue], [mode d'exécution])..... AjoutQuestion
HQ ("Libellé élément", "message d'aide") Help Question
FQ ()..... Fin Question
DT ()..... Début sélection Question
PQ ("Libellé racine", "Libellé élément", suite)..... Poser Question
DE (["nom de l'ensemble"])..... Début Ensemble
QT ("ligne de texte")..... Question texte
QO (n°objet)..... Question Objet
FE ()..... Fin d'ensemble
FT ()..... Fin sélection Question

5. Réalisation de l'application d'interaction utilisateur

5.1. Introduction

L'application d'interface utilisateur Macao (**M**odélisation **A**nalyse et **C**onception **A**ssistées par **O**rdinateur) a été réalisée pour l'introduction des données et la visualisation des résultats sur une station de travail, utilisable soit en mode autonome soit en mode connecté. Macao permet de construire et de manipuler simultanément plusieurs graphes.

La conception d'une interface utilisateur se fait par de multiples itérations du cycle évaluation-modification d'un prototype et par des adaptations ou des imitations d'effets visuels de produits similaires éprouvés. La réalisation du "design" d'une interface utilisateur peut souvent ne pas satisfaire tout le monde à la fois, il faut maintenir un équilibre entre le "design" et les buts de l'interface utilisateur donc un équilibre entre la faisabilité et l'implémentation. C'est pourquoi l'application d'interaction utilisateur Macao devra sans cesse évoluer pour suivre les besoins des utilisateurs.

Macao, dans sa version actuelle, est implémenté sous forme d'une application Macintosh pouvant s'exécuter sous MacOS ou AU/X (le système Unix d'Apple). L'application, écrite en MPW-C (**M**acintosh **P**rogramming **W**orkshop **C**) et MPW-C++, se compose d'une quarantaine de modules (programmes sources). Chaque module est responsable d'une structure de données (classe) ou d'une fonctionnalité et comporte des points d'entrée (méthodes) décrits sous forme de prototypes (ANSI-C) dans un fichier d'entête ".h". Les modules du squelette gérant les fenêtres et les menus ont été développés par extensions du squelette d'application de Paul Dubois¹ (TransSkel).

¹ Wisconsin Regional Primate Research Center, 1220 Capitol Court, Madison, WI 53715-1299 USA. dubois@primate.wisc.edu.

Après quelques généralités sur le squelette d'application, nous présentons les niveaux plate-forme de Gestion des Utilisateurs et des Services (GUS), de Gestion de Station de travail Virtuel (GSV), les niveaux interface et application avec en particulier une description de l'éditeur de graphe.

5.2. Généralités

La réalisation du programme d'interaction utilisateur de l'atelier AMI est basée sur la contrainte fondamentale que ce programme doit toujours être dans l'attente d'un événement utilisateur et le traitement de cet événement doit être le plus court possible. En effet, l'utilisateur doit pouvoir à tout moment intervenir sur l'interface pour effectuer une action.

Dans une programmation mono-processus, une boucle d'événements est parcourue en permanence en attente du prochain événement à traiter. Des **événements extérieurs** peuvent être la conséquence d'actions de l'utilisateur comme la lecture ou l'écriture d'un fichier. Dans la mesure du possible, les actions longues en temps de calcul sont décomposées en actions élémentaires indépendantes et vues comme des **événements internes**.

Pour rendre Macao indépendant de la station de travail, toutes les parties de Macao liées au système d'exploitation de base et au système graphique (Boite à outils de programmation) passent par un squelette d'application comme nous l'avons défini dans l'architecture fonctionnelle de MARS (§II.4.2 Le niveau plate-forme - squelette). Les points d'entrée du squelette sont définis mais l'implémentation du squelette est dépendante de la boite à outils de programmation d'interface utilisateur utilisée (Macintosh, OPEN LOOK, OSF/Motif). Le **squelette Macintosh** est écrit en C et fait appel aux points d'entrée de la **toolbox Macintosh**. Le rapport technique [SUN, 1990] propose des exemples de portage d'applications Macintosh sur SunOS.

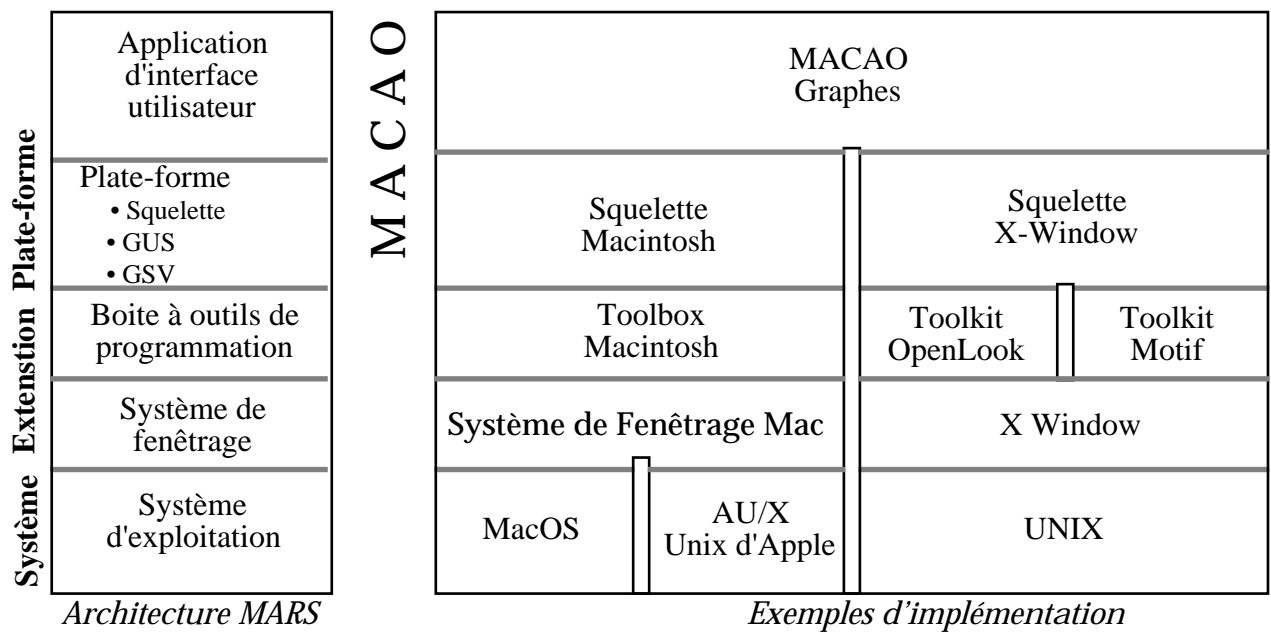


Figure 5.2: Exemples d'implémentation de l'architecture de MARS

Les utilitaires de base communs à de nombreux modules sont inclus dans le squelette. Nous pouvons citer par exemple, le gestionnaire de listes. Ce même gestionnaire de listes est d'ailleurs utilisé par tous les programmes de l'atelier AMI, aussi bien sur Macintosh que sous Unix.

5.3. Niveau Plate-forme GUS

5.3.1. Le GUS dans MACAO

La gestion des utilisateurs n'est utilisée que par le gérant des utilisateurs de la couche transport pour l'identification de l'utilisateur. En effet, Macao n'est l'interface que d'un utilisateur à la fois. Cette gestion est effectuée par le module `Utilis.c`.

La gestion des formalismes est composée de deux modules, l'un gérant les noms des formalismes (`Domaines.c`²) et l'autre les objets du formalisme (`FMode.c`). Le module `Domaines.c` est utilisé par le gestionnaire des menus de l'éditeur de graphe (§II.7.4.2.a) pour présenter à l'utilisateur le nom du formalisme choisi. Le module `FMode.c` est utilisé par le module de dessin des objets (`Caract.c`³).

La gestion des services (module `Interlocuteur.c`⁴) réceptionne la liste des services disponibles et est utilisée par le gestionnaires des menus des services (§II.7.4.2.b).

5.3.2. Gestion des utilisateurs

Le GUS de Macao reprend l'architecture présentée dans le cahier des charges d'un atelier de spécification. Macao implémente la partie GUS-client, il est donc responsable de l'identification de la personne et du choix de la langue utilisée. L'identification de la personne est réalisée par appel au système d'exploitation pour en connaître son nom et son mot de passe. Sur la version 6.0 du système du Macintosh, le nom de l'utilisateur se trouve dans une ressource système mise à jour par le "sélecteur". Dans cette version le GUS propose le nom et demande le mot de passe par une fenêtre de dialogue. Dans la version 7.0 de MacOS, le mot de passe est géré par le système. La détermination de la langue utilisée se fait de la même façon, par appel au système ou par fenêtre de dialogue.

Lorsque l'on étendra l'atelier vers une collaboration multi-utilisateurs pour la production d'un énoncé commun ou pour la réutilisation de sous-énoncés exportés, il faudra rajouter la notion de groupe d'utilisateurs coopérants et de nouvelles fonctionnalités au GUS local.

Module Utilis.c (extrait)

Identification de l'utilisateur avec demande de mot de passe éventuel.

```
void    UtOpen    (Boolean    withPassword)
```

² Les noms des modules ne correspondent pas toujours avec le nom de la classe qu'ils manipulent, ceci pour des raisons historiques. Ici "domaine" était l'ancien nom pour "formalisme" d'où le module `domaine.c` gérant les formalismes.

³ Ce module est responsable des caractéristiques des objets dessinés, en particulier la méthode de dessin. Ce module est appelé pour dessiner les objets à l'écran lors du rafraîchissement de l'écran.

⁴ Même remarque que pour `domaines.c`, l'ancien nom de "services" était "interlocuteur"!

5.3.3. Gestion des formalismes

Le GUS-client pour la gestion des formalismes comporte une partie serveur conformément à l'architecture fonctionnelle (partie II, §5.10 Architecture du GUS dans un environnement distribué). Le GUS-client possède donc une description de tous les formalismes de la station de travail. Il établit et vérifie les liens avec les formalismes reçus du GUS-serveur. La gestion des formalismes du côté GUS-client consiste à mettre en correspondance les noms des formalismes ainsi que les objets et leur description graphique. En cas de non correspondance (le formalisme a évolué), le GUS-client signale à l'utilisateur toutes les anomalies et l'utilisateur devra s'adresser à l'administrateur du formalisme pour obtenir la dernière version du formalisme.

Pour permettre la personnalisation (partie II, §5.3.4 Modification esthétique d'un formalisme) des formalismes locaux de l'utilisateur, Macao permet de modifier des éléments esthétiques (ex: taille de objets, couleurs...). Lorsqu'un utilisateur particularise son formalisme, tous les énoncés créés dans ce formalisme bénéficient des nouvelles particularités (ex: taille des caractères). Cette personnalisation se trouve dans le fichier de description du formalisme local à la station de travail.

Un problème se pose lorsqu'un utilisateur A transmet un fichier (énoncé) à un utilisateur B: Faut-il faire cohabiter sur la même interface utilisateur les deux particularisations A et B ? Nous répondons par l'affirmative car ces particularités interviennent dans des énoncés différents. Par contre, certaines particularités doivent se plier à la volonté de l'utilisateur B lors d'opérations dynamiques. C'est le cas, par exemple, si l'utilisateur B effectue une opération de **couper/coller** d'une partie de l'énoncé A vers son énoncé car la somme des particularités n'aurait aucun sens. Le couper/coller est cohérent, il porte sur le type de l'objet sans ses particularités. Un autre exemple est l'apparence graphique des objets sélectionnés (couleur de mise en évidence) et non sélectionnés dans un énoncé: cette apparence est globale pour tous les énoncés sur l'interface de l'utilisateur de façon à ne pas la rendre incohérente. Cette personnalisation globale se trouve dans un fichier de préférences.

Le couper/coller d'objets est la seule opération permettant d'importer des objets d'un énoncé dans un autre; il pose un problème si on désire copier des objets depuis l'énoncé dans un formalisme vers un énoncé dans un autre formalisme. Cette opération doit-elle être interdite ? Quand a-t-elle sa raison d'être ? A priori, lorsque les objets à coller correspondent nom pour nom aux objets du formalisme de réception, l'opération est effectuée sans problèmes. Si la correspondance des attributs n'est pas totale, après autorisation de l'utilisateur, seuls certains attributs seront collés. Si la correspondance des objets n'est pas complète, un dialogue plus sophistiqué demande à l'utilisateur quelles sont les conversions à effectuer. Cela permet la réutilisation d'énoncés et le changement de formalisme d'un énoncé au cours de sa conception.

Ces opérations de couper/coller sont prévues pour fonctionner dans la version 2 de l'atelier.

Module Domaines.c (extrait)

```
    Réception de la liste des formalismes.
Boolean CcVD      (Ptr pNomDomaine,long tNomDomaine,Ptr pAProposDomaine,long
                  tAProposDomaine,Boolean fromHost);
    Compose l'article du menu "Atelier" contenant le nom du formalisme.
void DmCalculMenu(MenuHandle theMenu,short itemAtelier,short itemDomaine);
```

Module FMode.c (extrait)

```
    Ajoute un nouvel objet.
void FmAttache( short classe, short ind, HTypeObjet hTypeObjet);
    Rend le type de l'objet pour le dessiner.
HTypeObjet FmGetTypeObjet(short classe, short ind);
```

5.3.4. Gestion des services

Le gestionnaire des services réceptionne la listes des services offerts pour l'utilisateur et lui propose de choisir. Il indique par des messages d'aide, à quoi correspondent les services et si des résultats sont disponibles pour l'énoncé courant. Lorsque l'utilisateur a choisi un service, le gestionnaire initialise et demande l'ouverture de la session. Par la suite, le gestionnaire conserve la liste des services en cours d'utilisation pour pouvoir les arrêter lorsque l'utilisateur demande de quitter définitivement l'atelier (Menu Quitter de Macao).

Module Interlocuteurs.c (extrait)

```
    Ajoute un nouveau service.
Boolean CcVI      ( Ptr pNomApplication, long tNomApplication, Ptr pAPropos, long
                  tAPropos, short resultatCalcul );
```

5.4. Niveau Plate-forme GSV

5.4.1. Généralités

Dans une implémentation multi-processus, la programmation de protocoles de communication se fait par des processus en attente de messages sur des ports d'entrée-sortie (lecture bloquante). Dans l'implémentation mono-processus de Macao, il ne faut pas être en attente car cela bloquerait l'interface utilisateur.

Parce que l'initiative vient toujours de l'utilisateur et pour rendre minimal le temps de réponse de l'interface utilisateur, des mécanismes de communication que nous avons réalisés privilégient l'interface utilisateur par rapport aux communications avec les services (asymétrie du dialogue).

Pour mettre en place ces mécanismes de communication, il existe deux solutions qui passent par une simulation de "tâches de fond". Les **tâches de fond** sont des processus qui vont être réveillés régulièrement pour voir si il y a quelque chose à faire.

Dans Macao, ces processus sont simulés par des procédures appelées lorsqu'aucun événement utilisateur n'est à traiter dans la boucle d'événements gérée par le squelette d'application (§II.4.3 Squelette d'application).

Pour réaliser la connexion de Macao, il faut dans un premier temps réaliser une connexion physique entre le Macintosh et l'environnement Unix supportant le reste de l'atelier.

5.4.2. Réalisation de la connexion physique

Nous offrons deux moyens de connecter Macao à une machine Unix.

Le premier moyen est une **connexion point à point** par ligne série. Cette connexion est très utile car on peut la mettre en œuvre dans quasiment tout environnement Unix. En particulier, il est possible d'utiliser l'interface utilisateur par un modem ou dans un environnement minimal composé d'une station Unix et d'un macintosh.

Le deuxième moyen de **connexion dans un environnement distribué** est l'utilisation d'un protocoles de communication TCP/IP. Cette implémentation a été réalisée par MacTCP et elle nécessite soit un réseau LocalTalk et une passerelle LocalTalk/Ethernet permettant d'effectuer le routage entre les deux protocoles, soit un Macintosh comportant une carte Ethernet.

Dans les deux cas de connexion, il doit y avoir **identification** de l'utilisateur. La connexion par TCP/IP n'identifie pas l'utilisateur car la connexion est autorisée à toute personne sur un port TCP déterminé. Son nom et son mot de passe sont transmis pour que les applications qu'il demandera s'exécutent à son nom. La connexion par ligne série permet à l'utilisateur d'obtenir une émulation de terminal "tty". Dans cette fenêtre, il effectue un "login" sur la machine Unix et entre son mot de passe. La reconnaissance de l'utilisateur est alors faite par le système Unix.

La mise en place du **protocole de communication**, c'est à dire la possibilité d'échanger des messages CAMI, est mise en place. Dans le cas d'une communication TCP/IP, on utilise TCP pour transmettre les messages. Dans le cas d'une communication par ligne série, l'émulation de terminal est transformée en protocole de communication. Pour réellement mettre les deux applications en communication, l'utilisateur demande "Exécuter Serveur" du menu "Atelier". Le programme serveur est exécuté sur la machine Unix et il envoie un message "Eclipse⁵" indiquant au Macintosh que le protocole peut commencer. Le GSV reçoit ce message et initialise son protocole de ligne série.

Conformément à la partie II, (§5.2 Gestion des utilisateurs), nous avons interdit qu'un même utilisateur se connecte simultanément depuis plusieurs stations de travail. Une sous-couche transport de pilotage du protocole vérifie **l'unicité de connexion**. Cela est réalisé par le test de présence d'un fichier précis (.ami) dans le répertoire de l'utilisateur. Dans le cas de la connexion TCP, cette sur-couche identifie l'utilisateur (association nom utilisateur atelier, nom système UID) et rend **compatibles** les deux modes de connexion pour les couches supérieures. C'est à dire qu'elle prend les messages TCP et les envoie à la couche transport comme s'il s'agissait de messages venant d'une ligne série.

a. Communication point à point hétérogène

S'exécutant sur un système mono-processus privilégiant l'interface utilisateur, Macao peut ne pas être en état de recevoir ou de traiter assez rapidement les messages. L'indisponibilité du système mono-tâche pourrait ainsi engendrer des problèmes lors des transferts de données (pertes, déséquences de messages). Nous avons donc utilisé un protocole de communication avec contrôle de flux et détection des erreurs.

Nous avons conçu, au niveau de la couche transport, un protocole (pomme-soleil) garantissant la **fiabilité des transferts**. Il intègre aussi un mécanisme de **contrôle de flux** implicite à base d'une **fenêtre d'émission** et un **contrôle périodique des connexions** basé sur l'utilisation de temporisateurs et l'envoi de messages de synchronisation **traités de façon prioritaire**. La fenêtre d'émission permet d'autre part d'optimiser l'utilisation de la ligne de communication. Le contrôle des erreurs est effectué par un mécanisme classique de "**crc**".

⁵ Le soleil se cache !

La programmation sur le Macintosh mono-tâche et sur l'Unix multi-tâches (un processus assurant l'émission et un autre la réception) nous a conduit à spécifier les échanges de variables du protocole entre l'émission et la réception pour que les programmes sources (en langage C), sur le Macintosh et sur la machine Unix, soient identiques. La seule contrainte d'utilisation est l'utilisation d'une ligne à huit bits sans parité. Cette connexion a été testée de différentes manières: En liaison directe de 9600 bauds à 19200 bauds, par modems 1200 bauds et par modem plus Transpac.

En résumé: le protocole développé pour l'atelier est basé sur un protocole de type HDLC adapté à un problème de transmission asynchrone. Ses principales caractéristiques sont:

- utilisation d'une fenêtre en émission de taille ajustable,
- trames de taille variable, pas de déséquence de messages,
- algorithme de détection de coupure de ligne ou indisponibilité du partenaire,
- acquittement des trames en différé pour optimisation,
- possibilité d'extension (RNR (Receive Not Ready), changement dynamique des temporisations).

b. Communication dans un environnement distribué

La couche transport correspond à la gestion d'une connexion TCP/IP. Ce protocole assure le contrôle de flux, et le transfert fiable de donnée. Il garde l'état de la connexion.

Le modèle utilisé correspond au modèle client-serveur, le Macintosh étant client et les serveurs sur les machines Unix de l'atelier. L'utilisateur peut demander la connexion à un serveur. Pour cela le GSV effectue une demande de serveur sur le réseau (diffusion). Il attend la réponse du premier serveur et se connecte à ce serveur. Nous considérons que le premier serveur répondant est celui qui est le plus disponible, ceci entraînant un début d'équilibrage de charge.

5.4.3. La couche transport

Une solution pour implémenter le protocole de communication consisterait à **scruter** régulièrement par une tâche de fond s'il y a suffisamment de caractères arrivés sur le port et de lire ces caractères. La lecture peut être bloquante car on est sûr que les caractères sont présents. Cette solution, simple mais gourmande en temps de calcul, risquerait d'engendrer des pertes de messages si la tâche de fond n'était pas appelée assez régulièrement.

La solution naturelle que nous avons adoptée consiste à travailler par **interruptions**. Une demande de lecture est faite de manière asynchrone et l'adresse d'une procédure est passée en paramètre à cette demande d'entrée sortie. La fonction sera appelée sous interruption lorsque les caractères seront arrivés. Cette solution est efficace mais il faut faire attention à ne pas passer trop de temps dans la fonction sous interruption. La fonction se contente d'accuser réception de la commande reçue et l'ajoute à une liste des commandes à traiter. Les commandes sont ensuite traitées en différé par une tâche de fond.

5.4.4. Les couches session, présentation

Lors de l'ouverture de communication, nous vérifions la compatibilité des numéros des versions respectives de Macao et du reste de l'atelier (partie II §6.4.2 Couche session). En cas d'incohérence, l'interface utilisateur refuse d'ouvrir la session pour éviter d'endommager des données de l'utilisateur ou dialoguer incorrectement avec le reste de l'atelier.

Cette vérification est d'autant plus importante dans notre environnement de travail qu'il existe des copies multiples de Macao et que l'ensemble de l'atelier évolue. Si un utilisateur étourdi exécute une ancienne version de Macao, il en est informé dès l'ouverture de communication.

Module Utilis.c

```

    Ouverture de communication
void    UtOC    (const StringPtr pNomLogiciel,const StringPtr pNomVersion,const
                StringPtr pNomUtilisateur,const StringPtr pPassWord,short langue)
    Vérification de l'autorisation de connexion
Boolean CcUA(Ptr pMachine,long tMachine,short autorise)
    autorise= known user, unknown, unknown version of Macao, you're still loggin somewhere else.
```

En raison de l'implémentation mono-processus, les différentes couches du GSV (session, présentation, application) sont liées les unes aux autres. Ainsi les messages (commandes CAMI) en provenance de l'extérieur (lecture de fichier énoncés) ou en provenance de la structure d'accueil passent de couche en couche jusqu'à ce qu'une couche les reconnaisse. Cela est effectué par le passage de l'adresse mémoire du message reçu. Chaque couche analyse les premiers caractères de la commande CAMI et la traite si elle lui est adressée ou la passe à la couche supérieure par appel de procédure. La table de transcodage de la couche présentation est vide car Macao interprète directement les commandes CAMI.

5.5. Niveau interface

5.5.1. Le gestionnaire des dialogues

Nous n'avons pas mis en place de gestionnaire de dialogue dans Macao. La gestion des dialogues est répartie dans chacun des modules ayant besoin d'un dialogue avec utilisateur (GUS, gestionnaire de graphe). Cette gestion des dialogues ne représente environ que 5% du programme source de Macao.

5.5.2. Le gestionnaire des fenêtres

Les énoncés Macao apparaissent dans des fenêtres mais les fenêtres ne sont qu'au bas de la hiérarchie des objets de Macao.

Tout en haut se trouve le **fichier**: il représente tout l'énoncé et se compose de **feuilles** (par analogie aux feuilles de papier pour dessiner l'énoncé). Il existe deux sortes de feuilles: la feuille principale (racine) et, en cours d'exécution, les feuilles résultats. A une feuille correspond un formalisme. Ensuite viennent les **fenêtres** qui sont des vues d'une partie des feuilles à une certaine échelle. Une même feuille peut être vue dans deux fenêtres différentes à des échelles différentes. Enfin, les feuilles contiennent les **objets** du graphe gérés par le gestionnaire de graphes.

Dans une version hiérarchique, chacun des nœuds du graphe pourra être ouvert en une nouvelle feuille.

Module Cfenetre.c (extrait)

 Crée une fenêtre.

HandleFenetre FnCreerFenetre(Handle hFeuille, StringPtr pNomFichier, int noFenetre);

 Rend la feuille d'appartenance de la fenêtre.

Handle FnGetFeuille(HandleFenetre hFenetre);

 Rend la position de la fenêtre par rapport à la feuille.

void FnGetPositionsCourante(short *pValueX, short * pMaxX, short *pValueY, short *pMaxY);

5.5.3. Le gestionnaire de graphes

Le gestionnaire de graphes gère les **objets du graphe** (nœuds et connecteurs) ainsi que les textes composants le graphe. Il assure les fonctions de base utilisées dans l'éditeur de graphes (changement de coordonnées, dessin, liste des arcs d'un nœud, liste des attributs). Il fait le lien entre les objets du graphe et leur feuille d'appartenance.

Tous les objets d'un graphe comportent des **numéros internes** (§II.6.4.3.d Le langage de description des énoncés) tous différents qui servent à leur identification. Ces numéros d'objets servent surtout lors de la communication du graphe vers les programme d'application. Les numéros internes sont attribués par ordre croissant sans préoccupation de récupération de numéros libres. Lors des opérations de copier/coller, de nouveaux numéros sont attribués aux objets collés.

Module Cobjet.c (extrait)

```

    Rend la position d'un objet par rapport à sa feuille d'appartenance.
void    ObGetPos(Point    *pCoor, HObjet hObjet);
    Dessine un objet.
void    ObDessine(HObjet hObjet, Rect *pRectObjet, Boolean front, short
           action,HandleFenetre hFenetre);
    Parcours tous les connecteurs.
Handle  ObParcoursArc(Action, HObjet hObjet, Handle param1, Handle param2 );
    Parcours tous les attributs d'un objet.
Handle  ObParcoursText(Action, HObjet hObjet, Handle param1, Handle param2 );
    Rend un nouveau numéro interne.
Identificateur ObGetNextIdent(void);
```

5.6. Niveau application

5.6.1. L'éditeur de graphes

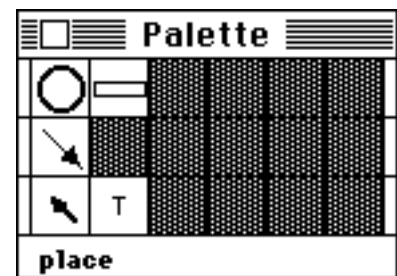
L'éditeur de graphes de Macao, peut se distinguer des autres éditeurs de graphes par sa configurabilité. Tout comme la structure d'accueil, Macao est indépendant de tous formalismes (Objectif 15) et ne possède donc pas, dans sa version actuelle, de fonctionnalités graphiques spécifiques à un formalisme mais possède des fonctionnalités communes à tous les graphes typés à attributs.

Nous allons voir que Macao est simple d'emploi, qu'il respecte l'interactivité de l'interface utilisateur et qu'il s'intègre parfaitement dans l'environnement de travail de l'utilisateur.

Simplicité d'emploi

La palette

L'utilisation d'une **palette** contribue à la simplicité d'emploi de Macao. En effet cette palette contient l'ensemble des formes graphiques des nœuds et des connecteurs du formalisme ainsi que des opérateurs de base comme l'opérateur de sélection et de manipulation de textes. Dans la figure ci-contre, la palette correspond au formalisme réseau de Petri, elle contient deux nœuds (une *place* et une *transition*) et un connecteur (un *arc*). L'objet sélectionné est la *place* (indiqué en bas de la palette).



La palette est liée à la définition du formalisme (partie II, §5.3 Gestion des formalismes). L'ensemble des classes nœud et connecteur y est représenté. La classe **information** (de forme graphique "logo") est instanciée en un objet dans la fenêtre de l'énoncé. Cet objet ne peut pas être supprimé et comme il ne doit être instancié qu'une seule fois, sa forme graphique n'est pas représentée dans la palette.

Création d'objets

Le procédé de manipulation est simple: l'utilisateur sélectionne un **nœud** dans la palette, le curseur de la souris prend la forme graphique de ce nœud⁶, en effet il est très important d'indiquer à l'utilisateur dans quel mode il se trouve. L'utilisateur clique alors dans la fenêtre à l'endroit où il veut positionner un nœud. Il peut ainsi positionner différents nœuds sur la feuille.

⁶ Ceci n'est pas toujours possible lorsque la forme graphique est complexe.

La création de **connecteur**, est conditionnée par la présence d'un nœud de départ et un nœud d'arrivée. L'utilisateur clique dans la palette sur la forme graphique du connecteur désiré, il clique ensuite sur le nœud de départ et maintient le bouton de la souris appuyé jusqu'à ce qu'elle arrive sur le nœud de destination. Si la destination est sémantiquement correcte, elle se met en évidence par un changement de couleur en temps réel sinon le connecteur n'est pas créé. Cette technique garantie qu'un connecteur provient bien d'un nœud créé et autorisé et arrive bien sur un nœud et autorisé (respectant l'ensemble des liens possibles (classe connecteur §II.5.3.1 Le Langage de Description des Formalismes)).

Création d'attributs

La création des **textes** obéit aux règles sémantiques du formalisme: tous les textes sont typés, c'est à dire que chaque texte représente un attribut d'un objet du formalisme.

La création des textes se fait par sélection de l'outil texte dans la palette, en cliquant sur l'objet dont on veut ajouter un attribut et en tirant jusqu'à la position désirée du texte par rapport à l'objet. Ensuite on saisit le contenu du texte dans une zone dont le nom est le type de l'attribut. On peut associer des textes aux nœuds, aux connecteurs ainsi qu'à l'objet d'information.

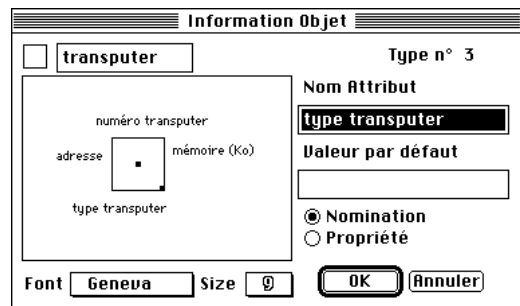


La définition d'un formalisme permet de donner des valeurs par défaut à certains attributs. Macao présente les valeurs par défaut au moment où l'utilisateur va saisir une nouvelle valeur. Si l'utilisateur ne saisie pas de valeur, c'est la valeur par défaut qui sera transmise à la structure d'accueil. Les valeurs par défaut ne sont en général pas affichées dans le dessin du graphe pour préserver la lisibilité.

Pour des raisons d'efficacité, et bien que ce ne soit pas très ergonomique, Macao offre une deuxième façon de saisir les textes d'un objet. Cette façon est beaucoup plus rustique (ou classique) puisque qu'elle consiste à demander des informations sur l'objet et à saisir ou modifier tous les attributs de l'objet, les uns en dessous des autres, à la manière d'un tableur. Après la saisie, les attributs viennent se positionner sur le graphe à des positions définies par défaut dans la partie esthétique du formalisme.

Modifications esthétiques

La modification de la partie **esthétique** du formalisme est tout à fait intuitive puisque Macao propose, dans une fenêtre, le dessin de l'objet et de l'ensemble de ses attributs dans leurs positions par défaut. L'utilisateur sélectionne simplement les attributs et les met graphiquement à une nouvelle position par défaut.



Interruption d'opération

L'utilisateur peut arrêter à tout moment une opération en cours sans que cela produise d'incohérence et le curseur de la souris lui indique en permanence l'opération qu'il peut effectuer (diminution de la charge cognitive).

Suppression

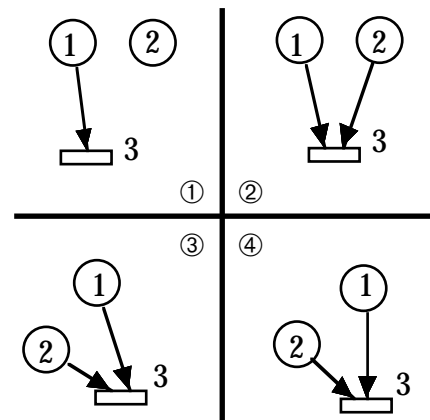
La **suppression** des objets graphiques suit les règles sémantiques des formalismes. C'est à dire que la suppression des objets entraîne la suppression automatique des textes. De même la suppression des nœuds entraîne la suppression des connecteurs reliant ces nœuds.

Routage et points de connexion

Nous avons constaté que l'utilisateur attachait souvent une importance au placement des nœuds du graphe. Nous lui avons donc laissé la charge de positionner les nœuds, par contre nous avons voulu rendre le plus automatique possible le dessin des connecteurs reliant ces nœuds. Actuellement, le placement automatique des nœuds et le **routage** automatique ne sont pas encore réalisés. Toutefois, nous avons déjà mis en place un système automatique d'**organisation des points de connexion** des connecteurs sur les nœuds.

La plupart des éditeurs de graphes se contentent de faire arriver les connecteurs vers le centre des nœuds. Cette technique est tout à fait valable lorsque les nœuds sont ronds mais elle ne correspond pas exactement à la façon dont les utilisateurs positionnent les **points d'arrivée** (de connexion) des connecteurs sur des objets de forme rectangulaire. Macao possède en interne plusieurs algorithmes de calcul des points d'arrivée en fonction de la forme des nœuds. Ces algorithmes font partie de l'esthétique du formalisme et ceci constitue une des originalités de Macao.

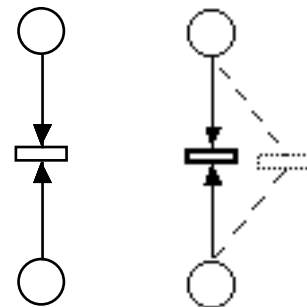
Dans la figure ci-contre, l'objet 1 est connecté à l'objet 3: le point d'arrivée de la flèche se situe exactement au centre de l'objet 3. Au moment où l'on relie l'objet 2, les deux flèches se répartissent équitablement le long de l'objet 3. Dans le troisième dessin, le fait de déplacer l'objet 2 à gauche de l'objet 1 entraîne une permutation des deux flèches.



Le dernier cas montre l'utilisation d'un connecteur primaire, prioritaire sur les autres connecteurs au moment du calcul du point de connexion. Le connecteurs entre 1 et 3 est un connecteur primaire. Il se trouve donc au centre de l'objet 3 et le connecteur venant de 2 se place au centre de l'espace restant.

Déplacement d'objets

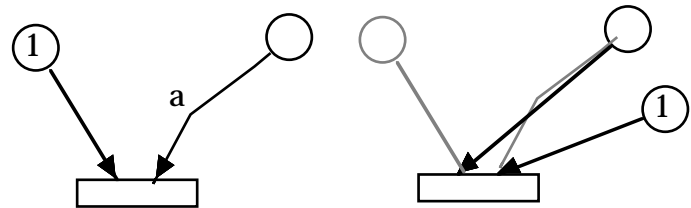
Pour **déplacer** les objets, il faut les d'abord les sélectionner, ensuite cliquer sur un objet, tirer vers une position finale puis relâcher le bouton de la souris. Pendant le déplacement, Macao calcule les contours de chacun de nœuds et ce contour va suivre, en temps réel, le déplacement de la souris. Pendant tout le déplacement, les connecteurs reliant des objets non sélectionnés aux objets qui se déplacent suivent le mouvement. Cette opération permet de donner un feed-back visuel à l'utilisateur mais elle est très coûteuse et n'aurait pas pu être implémentée de manière efficace si nous étions passé par de lourds protocoles réseaux comme X-Window !.



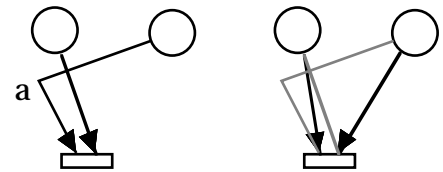
Points intermédiaires

Macao permet de placer des **points intermédiaires** sur les connecteurs. Le placement d'un point intermédiaire est simple puisqu'il suffit de sélectionner un connecteur et de tirer vers la position du point intermédiaire pour qu'il apparaisse. La suppression des points intermédiaires s'effectue soit de manière manuelle, en sélectionnant le point et en demandant sa suppression, soit de manière automatique. La suppression automatique est une des originalités de Macao dont le principe est simple mais dont la mise en œuvre est difficile en raison des nombreux cas rencontrés. Le principe est qu'un point intermédiaire est supprimé dès qu'il n'est plus nécessaire, c'est à dire lorsqu'il se trouve dans l'alignement de deux autres points du même connecteur. La combinaison de cette fonctionnalité avec la réorganisation des points de connexion complique la situation.

Dans cet exemple, l'utilisateur déplace l'objet (1) ce qui entraîne une réorganisation des points de connexion. Cette réorganisation fait que le point intermédiaire (a) se trouve aligné et doit être supprimé. Ici réorganisation \Rightarrow suppression.



Dans ce deuxième exemple, l'utilisateur supprime le point intermédiaire (a). Cette suppression va faire que les deux arcs se croisent, il faut donc réorganiser les points de connexion. Ici suppression \Rightarrow réorganisation.



Ces deux exemples montrent que l'on peut obtenir des enchainement suppression, réorganisation, suppression. Il faut s'arrêter lorsque la suppression ou la réorganisation n'aura rien changé par rapport à la situation initiale.

Alignement

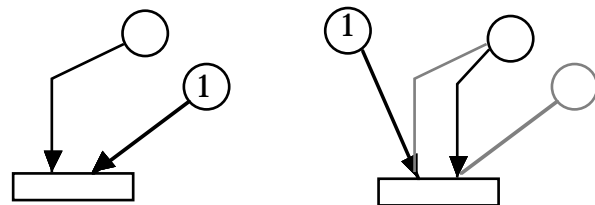
L'alignement des objets est une fonction essentielle d'un programme de dessin. En général, de nombreuses possibilités sont offertes sous forme de grille ou de commande d'alignement. L'application Macao ne se contente pas de reproduire ces fonctionnalités mais elle les améliore en tenant compte de la spécificité des graphes et en reproduisant les habitudes des utilisateurs.

La fonction de **grille**, par exemple, aligne le centre des objets et non pas leurs cotés, elle possède une définition horizontale et verticale paramétrable de manière autonome. Elle peut aussi se calculer en fonction d'une répartition d'objets sur une surface donnée. Certains objets, comme les textes ou les points intermédiaires, pourront bénéficier d'un autre type de grille moins rigide permettant d'aligner des points intermédiaires en fonction des points de connexion.

Les fonctions **d'alignement** des programmes de dessin sont en général complètes mais complexes car elles ne peuvent pas s'appuyer sur une sémantique. A l'opposé, Macao n'offre que le strict nécessaire: une seule fonction: Aligner. Pourquoi ? Premièrement parce qu'avec des fonctions complexes, l'utilisateur a tendance à se tromper et deuxièmement parce que la fonction voulue par l'utilisateur peut être calculée par le programme. Par exemple, lorsque l'utilisateur sélectionne un ensemble d'objets les uns au dessus des autres, on peut supposer qu'il veut les aligner verticalement. Alors pourquoi lui proposer un alignement horizontal qui ramènerait les objets les uns sur les autres. D'autre part, si ces objets sont des nœuds, il veut de préférence les aligner sur leurs centres et si ce sont des textes, c'est sur leur gauche qu'ils seront le mieux présentés. (Macao tiens bien sur compte du cadrage des textes et du sens d'écriture de l'utilisateur par utilisation du "Script Manager" du Macintosh).

Mais la simplicité de Macao dans les fonctions d'alignement ne s'arrête pas là. En effet, il serait désagréable, dans un programme de dessin, de demander un alignement et, après avoir déplacé certains objets, de devoir le demander à nouveau. Nous avons introduit le concept de **contrainte mémorisée**. Macao se rappelle des différents alignements et maintient les objets alignés en temps réel lors de leurs déplacements. Ce choix délibéré, peut bien sur être temporairement ou définitivement inhibé pour certains objets.

L'exemple ci-contre présente une nouvelle fonctionnalité de Macao qui, bien qu'intuitive, n'est pas toujours prise en compte dans les programmes de dessin de graphes. Il s'agit d'un point intermédiaire parfaitement aligné à la verticale d'un point de connexion. Lorsque l'utilisateur déplace l'objet (1), la réorganisation des points de connexion entraîne un déplacement sur la droite du point intermédiaire.



Extensibilité

La séparation entre le gestionnaire et l'éditeur de graphes permet d'ajouter de nombreuses fonctionnalités à Macao sans en changer l'architecture. De cette manière a été ajouté un programme de placement semi-automatique écrit en C++. Le placement semi-automatique consiste en un positionnement en un centre de gravité puis un centre géométrique d'un ensemble de nœuds libres par rapport à un autre ensemble de nœuds fixes [Collet, 1990]. Ce programme sophistiqué a pu être intégré dans Macao sous la forme d'une commande dans un menu qui a rendu obsolète une ancienne commande moins sophistiquée implémentée dans un module autonome. Nous avons montré que Macao était **extensible**.

Interactivité

Comme nous l'avons vu précédemment, aucune opération longue ne doit être entreprise de façon à laisser l'interactivité maximale au niveau de l'interface utilisateur. Comme le **dessin** des objets graphiques intervient dans plusieurs fenêtres simultanément (par exemple différentes vues à différentes échelles sur la même feuille), Macao envoie des "événements de mise à jour" de zones de fenêtres (**update**) pour les fenêtres en arrière plan et un ordre de dessin immédiat pour la fenêtre en premier plan. Avec cette technique, l'utilisateur voit une modification immédiate et peut dessiner d'autres objets sans attendre le dessin des fenêtres en arrière plan. De plus l'ordre de mise à jour est un ordre cumulatif c'est à dire que plusieurs ordres de mise à jour sont regroupés pour une seule mise à jour effective.

Lors du **déplacement** des objets, il y a toujours de nombreux calculs à effectuer pour les points de connexion et les points intermédiaires des connecteurs. Si on faisait ces calculs immédiatement, l'interface utilisateur serait pénalisée car l'utilisateur devrait attendre la fin du routage pour effectuer un autre déplacement. La technique employée dans Macao est d'envoyer des messages aux connecteurs déplacés. Ces messages seront traités en différé par une tâche de fond. Au moment du traitement, les connecteurs agissent, comme dans un système d'acteurs, indépendamment les uns des autres. Ils ont une vision locale et traitent leurs points d'intermédiaires, ensuite ils envoient, si nécessaire, des messages à leurs nœuds adjacents qui à leur tour traitent leurs points de connexion.

Dans un système de fenêtrage, les messages envoyés peuvent être gérés par le système sous forme "**d'événements d'application**". Cette technique, adoptée dans un premier temps dans une version préliminaire de Macao, a trouvé rapidement ses limites. La première est que le nombre de types d'événements d'application est limité, que le système limite aussi la liste des événements en attente qui dans notre cas il pouvait être importante. Une deuxième raison est, qu'une fois en attente dans le système, un événement ne peut pas être enlevé alors qu'il est toujours possible que l'utilisateur supprime des objets en cours de routage. Pour ces raisons, le routage dans Macao n'utilise pas les événements d'application. Par contre, les "événements de mise à jour" sont employés car ils sont cumulatifs, donc peu nombreux et qu'ils ne portent pas sur des objets mais sur des surfaces dans une fenêtre: si l'objet a été supprimé, l'ordre de mise à jour n'aura simplement aucun effet.

Intégration dans l'environnement de travail

Dans l'analyse des besoins des utilisateurs, nous avons vu qu'il était essentiel que l'application d'interaction utilisateur soit parfaitement intégrée dans l'environnement bureautique utilisé (Objectif 11). Le respect du "look and feel" de l'interface utilisateur est bien sûr impératif mais il faut aussi que les énoncés créés par Macao puissent être transférés vers d'autres programmes.

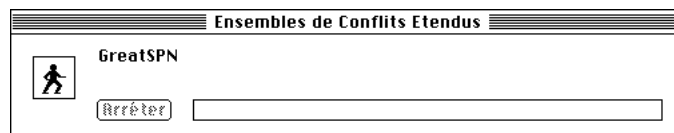
L'intégration de l'application dépend forcément de l'environnement et des autres programmes. Dans le monde Unix, il semble qu'une interface PostScript fasse l'unanimité. Dans le domaine du Macintosh, c'est une interface QuickDraw qu'il faut fournir.

Macao n'est pas un programme généraliste de dessin. Pour compenser cela, il offre une interface originale vers un programme plus généraliste MacDraw. Cette interface actuellement réalisable par copier/coller pourra prochainement être effectuée par des liens dynamiques entre applications: MacDraw est capable d'interpréter n'importe quelle commande QuickDraw mais il offre d'autres fonctionnalités au moyen de "commentaires QuickDraw" comme les associations d'objets ou les lignes courbées (courbes de Bézier). Macao transmet ces commentaires et permet d'obtenir des courbes parfaitement calquées sur les habitudes des "experts" de MacDraw.

5.6.2. L'interface utilisateur des services

Macao, en tant qu'interface utilisateur des services, maintient la correspondance entre les fenêtres à l'écran et les sessions ouvertes vers les applications d'analyse. Il met à jour ses menus et les complète par ceux de l'application distante.

Lorsque l'utilisateur demande l'exécution d'un service, Macao, devant offrir un feedback immédiat, anticipe l'exécution par affichage d'un message d'attente (Figure ci-dessous) et envoie la demande d'exécution.



La fonction de Macao en tant qu'interface utilisateur des services est détaillée dans le paragraphe d'utilisation de l'atelier (§7.5 Application).

5.6.3. Interface utilisateur d'administration et extensibilité

La **création de nouveaux formalismes** est prise en charge en respectant les formalismes existant dans l'atelier (c'est le travail de l'administrateur des formalismes). Cette création passe par une description graphique des objets du formalisme. Macao comporte un certain nombre de nœuds et de connecteurs prédéfinis. S'ils conviennent, l'administrateur des formalismes doit spécifier la taille des différents objets, la police de caractère, la taille et la position par défaut de l'ensemble des attributs textuels par rapport aux objets. Si les objets ne conviennent pas, il est possible d'en ajouter à Macao de manière externe (sous forme de ressource) par copie de n'importe quel graphique. Ce graphique deviendra un nouveau nœud. Cette technique est très simple mais elle pose des problèmes de portabilité d'application c'est pourquoi il est aussi possible d'ajouter à Macao de objets sous formes de nouvelles méthodes. Dans ce cas il faudra recompiler le programme Macao.

6. Réalisation sous Unix™ de la structure d'accueil

6.1. Niveau extension système

6.1.1. Système - Répartition

Gestion des processus

Un des points forts du système Unix est la notion de processus. Tout programme qui s'exécute dans un système Unix est un processus. Un processus "fils" est créé par duplication "fork" d'un processus "père". Cette primitive fork permet la création dynamique d'un nouveau processus qui s'exécute de façon concurrente avec le processus qui l'a créé. Le processus fils est une copie exacte du processus père. Pour que le fils exécute un nouveau programme, il fait appel aux fonctions "exec" [Rifflet, 1986]. Lors du "fork" le fils hérite du contexte système de son père: liste des descriptifs et de certains signaux. Sur une machine Unix, tout processus possède un identificateur système unique (pid) et s'exécute avec les droits de l'utilisateur (uid et guid).

Les processus s'exécutent de manière concurrente et il est impossible (en Unix Berkeley) qu'un processus interagisse directement sur les données d'un autre processus. Nous avons utilisé cette particularité pour regrouper dans un processus toutes les primitives qui effectuent des opérations sur des objets (encapsulation). Cette technique de programmation procure aux programmes ainsi réalisés une autonomie et une sûreté de fonctionnement. Toute l'architecture de l'atelier est ainsi réalisée par cette technique. Pour permettre une utilisation uniforme des processus, nous avons défini un module de gestion des processus (SVprocess.c).

Dans l'atelier, toute création d'un processus passe par ce module commun de gestion des processus. Un avantage de cette programmation par processus est la non propagation des erreurs d'un programme à un autre. En effet, en programmation classique ou tous les modules sont regroupés dans un même programme, un effet de bord d'une fonction est toujours possible et l'erreur ainsi produite est difficile à cerner. Avec notre méthode de programmation une erreur est localisée à un processus. De la même manière cette méthode facilite la mise au point des programmes. En effet, pour suivre le déroulement d'un programme, il suffit de lancer un dévermineur (debugger) sur le processus, sans affecter les autres processus de l'atelier.

L'atelier AMI s'exécutant dans un environnement de machines hétérogènes sous Unix, l'utilisation de processus, facilite le lancement distribué des programmes. Avec l'utilisation de processus, la mise au point d'un programme dans cette environnement est aussi simple que dans la cas d'un fonctionnement sur une seule machine.

Les programmes échangent des données ou accèdent à des objets gérés par d'autres processus. Nous avons défini un moyen de communication unique entre les processus dans tout l'atelier. Le seul inconvénient de la programmation par processus, lors de la mise au point d'un programme est de forcer les synchronisations du processus par rapport aux autres. C'est pourquoi nous avons d'abord réalisé un protocole d'échange de messages inter-processus dans un environnement distribué, fiable et sans problème de synchronisation.

Communication inter-processus

Nous avons présenté dans le chapitre II (§3.2.1) les moyens de communication entre processus, ainsi que les entités échangés. En résumé cette communication asynchrone doit être bidirectionnelle avec contrôle de flux, fiable et le système doit fournir en temps réel un état de la connexion.

Pour mettre en œuvre un tel mécanisme de communication sous Unix, nous utilisons des canaux de communication en mode connecté, les “sockets streams” du protocole TCP/IP [Comer, 1988]. Ce protocole assure une fiabilité de transmission et un contrôle de flux. Une socket (porte) est une adresse de transport sur une machine. Pour un processus, la création d’une porte et la désignation de cette adresse de transport dans l’environnement distribué est réalisée par des appels systèmes. Pour utiliser ce moyen de communication, le système Unix doit posséder les extensions du système Unix Berkeley 4.2 [Leffer, 1989]. Ces extensions offrent un mécanisme général de communication entre deux processus quelconques, qu’ils appartiennent à un même système ou à deux systèmes distincts. Le système Unix[Rifflet, 1990] rend totalement transparent les accès aux portes et au système de fichiers par une homogénéisation des accès par une table de descriptifs. Ainsi un processus écrit de la même manière dans une porte (socket) que dans un fichier. Après création d’une porte, un processus y accède par un numéro de descriptif et les opérations élémentaires qu’il peut effectuer sont l’écriture et la lecture asynchrone d’octets. Pour une telle communication, un processus crée une porte et se met en attente de connexion d’un autre processus qui connaît le nom de cette porte. Le mécanisme de création et connexion est dissymétrique.

Le système Unix [SUN, 1986] gère le protocole TCP/IP et quand deux processus réalisent une communication par socket stream, le système assure une transmission fiable des octets et une surveillance de l’état de la connexion. Dans le cas d’une rupture de communication par l’un des deux processus (arrêt d’un processus), le processus restant reçoit du système un message de fin de communication (EOF). Quand un processus se termine de façon anormale, le système assure la fermeture de tous les descriptifs. Le contrôle de flux est géré par le système et se traduit dans notre cas par le ralentissement du processus émetteur, le système stockant les octets à envoyer.

Le mécanisme de communication “socket stream” est la base de toutes les communications dans l’atelier AMI, un module spécifique de gestion des sockets (*SVsocket.c*) est commun à toutes les applications. Pour rendre les applications le plus indépendantes des demandes de lecture et d’écriture dans les sockets nous avons défini un module (*SVcomIP.c*) assurant l’écriture et la lecture d’un ensemble d’octets contenu dans un tampon (buffer).

Une application qui veut écrire plusieurs octets dans une socket utilise une fonction d’écriture (*SVecritDesc*) qui assure l’envoi du bon nombre d’octets. Dans le cas où le système ne peut accepter tous les octets en bloc, la fonction réitère la demande. L’application est ainsi sûre que sa demande d’écriture a été prise en compte par le système. L’utilisation d’un module de lecture et écriture d’octets dans des descriptifs liés à des sockets permet de cacher la notion de socket aux applications.

De la même manière quand un processus fait une demande de lecture de n octets, la fonction de lecture bloquante (`SVlectDesc`) récupère les octets dans un buffer et réitère les demandes de lectures si le système ne lui fournit pas en bloc les n octets. Cette situation est fréquente dans un environnement distribué hétérogène: elle dépend, d'une part, de la vitesse d'émission de l'écrivain et de la vitesse du lecteur, et d'autre part, de la rapidité des machines où s'exécute le lecteur et l'écrivain.

Un processus lecteur est obligé de faire une demande de lecture d'un nombre d'octets prédéterminé or les messages que nous utilisons dans la communication inter-processus ne sont pas de taille fixe. Nous avons été obligés de créer un moyen de communication au dessus des sockets streams, gérant la notion de message. Un message est une suite d'octets précédée par sa taille (codée sur un octet). La fonction d'envoi d'un message (`SVenv`) est donc basée sur la fonction d'écriture d'octets dans un descriptif (`SVecritDesc`). Pour recevoir un message, une fonction (`SVrec`) assure la réception d'un message par appels successifs à la fonction de lecture d'octets `SVlectDesc`. Dans un premier temps la fonction `SVrec` demande une lecture d'un octet qui correspond à la taille du message. Dans un second temps elle demande une lecture de la taille du message et retourne la taille du message lu et le message. Dans le cas où, pendant la réception du message, la communication s'est rompue (arrêt de la machine ou s'exécute l'écrivain, fin anormale du processus), la fonction retourne une taille négative. Quand un processus appelle la fonction de lecture d'un message, il est bloqué tant qu'un message n'est pas reçu.

Les messages émis par un processus sont bufferisés par le système (contrôle de flux) et transmis au processus lecteur. Dans notre environnement distribué, il se crée une désynchronisation, suivant les vitesses du lecteur et de l'écrivain. Quand l'écrivain décide de s'arrêter, le système Unix n'assure pas l'envoi des messages en cours de transmission puisque le processus n'est plus présent.

Nous avons donc réaliser un mécanisme de terminaison pour éviter la perte des messages entre l'écrivain et le lecteur. Ce mécanisme ne peut utiliser la gestion des signaux du système Unix, puisque le système ne permet pas l'envoi de signaux d'une machine à une autre.

Quand un processus écrivain décide de s'arrêter, il demande explicitement la fermeture de la communication par l'appel d'une fonction (SVcloseLiaison) qui envoie au lecteur un message de longueur nulle et attend une confirmation de fermeture par le système. Le lecteur en attente de message est en cours d'exécution de la fonction de réception de message (SVrec) qui interprète le message de longueur nulle. Le processus lecteur est donc informé de la demande de fermeture de la communication avec l'écrivain, il envoie un accusé de réception de fermeture à l'écrivain en appelant la fonction (SVacceptCloseLiaison). Cette fonction ferme la porte de communication correspondante par un appel systeme. Cette fermeture par le système est concrétisée par une rupture de communication et le processus écrivain reçoit du système une fermeture de communication. Naturellement, la fonction d'envoi de message interdit toute émission de message nulle.

Pour assurer la communication d'un processus avec plusieurs autres, nous avons réalisé une fonction qui teste la présence d'un message sur ensemble de portes. Elle arbitre ainsi la lecture du message. Cette fonction retourne le numéro de descriptif Unix associé aux portes qui ont des messages en attente de lecture.

L'utilisation de modules spécialisées pour la communication de messages inter-processus qui cachent à l'application les moyens de communication employés, nous permettent d'envisager d'utiliser d'autres techniques de communication de bas niveaux comme les files de messages en System V d'ATT [Presotto, 1990] pour ne pas lier la réalisation de l'atelier aux systèmes Unix qui contiennent les extensions de communications Berkeley.

Communication par messages CAMI

Nous avons construit un module plus général au dessus du module de gestion de communication inter-processus par messages. Ce module assure la construction, l'envoi, de réception, de décomposition et d'analyse de messages CAMI. Il intègre à cet effet des fonctions d'analyse.

Tous les protocoles de communications de la plate-forme Gestion de Station de travail Virtuelle utilisent les messages CAMI et d'autre part que la communication dans la structure d'accueil est aussi réalisée par messages CAMI.

Nommage

Nous avons réalisé les fonctions de création de processus, de portes, de communications de messages dans un environnement distribué. Il faut aussi définir des moyens de nommage des portes de communications pour les processus qui veulent dialoguer entre eux.

Dans protocole de communication TCP/IP, une adresse de transport est une association entre le nom de la machine et le numéro de la porte. Le système Unix gère dynamiquement l'élaboration des numéros de porte, mais l'administrateur système peut réserver dans des tables systèmes des numéros de portes pré-déterminés pour une machine. Pour avoir les mêmes numéros réservés sur l'ensemble des machines d'un site informatiques, l'administrateur système doit recopier sur toutes les machines les tables ou utiliser des outils distribués d'administration systèmes (NIS Network Information Service (ex Yellow Pages) de Sun). Le mécanisme de création et connexion est dissymétrique dans la mesure où un processus crée une porte et l'autre s'y connecte.

Pour faciliter l'utilisation des numéros pré-déterminés sur un site informatique, des noms externes peuvent être associés à ces numéros (fichier `/etc/services`). Cette technique de nommage du port par un nom est très utile dans le cas où des processus accèdent toujours à une même porte, comme pour un processus démon qui attend des connexions d'autres processus (modèle client-serveur).

Dans notre cas, cette technique ne peut pas être utilisée pour tous les processus de l'atelier, car le nombre de processus et le nombre de connexions n'est pas déterminé au départ. Il dépend du nombre d'utilisateurs connectés à l'atelier à un instant donné et du nombre de programmes d'applications en cours d'utilisation. Dans la plupart des cas, les communications se font entre processus père et processus fils. Un processus père crée une adresse de transport (porte) puis, crée un fils par duplication (`fork`). Le père se met alors en attente d'une connexion à cette adresse. Au moment du `fork` le père donne l'adresse de transport en paramètre au processus fils. Le fils en exécutant l'application (`exec1`) récupère ce paramètre et établit une connexion avec son père. Dans l'atelier, la création d'un processus passe par le module `SVprocess.c` qui réalise le passage de l'adresse de transport entre père et fils.

Les modules de gestion de processus (`SVprocess.c`) et de gestion des sockets (`SVsocket.c`) sont communs à toutes les programmes de l'atelier, ils assurent les problèmes de nommages et d'établissement de communications entre processus dans l'environnement distribué.

Temporisateur

Les protocoles de communication emploient souvent des temporisateurs. Pour programmer ces protocoles, nous avons réalisé un module responsable de la gestion des temporisateurs. Il gère un ensemble d'alarmes concernant un processus. A l'expiration d'une alarme, ce module exécute des actions associées à l'alarme. Ce module gère un échéancier des listes d'actions à exécuter.

Les seules opérations de manipulation d'alarmes accessibles au programme sont l'ajout et la suppression d'une alarme. Ce module prend en compte les problèmes d'interruption sur horloge du processus. Durant la manipulation de l'échéancier, il interdit, par exclusion mutuelle, toute interruption d'une alarme.

Gestion des fichiers distants

L'extension système détaillée dans le chapitre II (§3.2.1 Système - Répartition) a pour but d'assurer un partage d'informations entre sites de manière transparente et de rendre uniforme l'accès aux fichiers des utilisateurs sur l'ensemble des sites supportant l'atelier.

Comme nous l'avons décrit dans l'architecture matérielle, l'accès aux fichiers est assuré par le système NFS de SUN [NFS, et al., 1985]. Ce système permet l'exportation de système de fichiers vers d'autres machines et rend transparent l'accès aux fichiers distants. Dans cette version de l'atelier nous n'avons pas envisagé l'utilisation de serveurs NFS de secours en cas d'arrêt du serveur gérant les données des utilisateurs et de l'atelier.

6.1.2. Système - Logistique

Gestion de configuration

Le module de gestion de configuration conformément à l'analyse de la partie II (§3.2.2 Système - Logistique) possède des fichiers de descriptions des machines et des logiciels employés.

Le fichier "machine", stocké dans une partition NFS, contient la description des machines qui supportent l'atelier. Les informations stockées sont:

- le nom de la machine ("hostname")
- le type de la machine (sun3, sun4, sun386i...)

Le fichier "softs", stocké dans une partition NFS, décrit l'emplacement de logiciels dans le système de fichiers pour chaque machine.

Un exemple du fichier softs:

```
ada apollon /usr/distant/ada/ada-4.3
ada ostar /usr/local/ada-4.3
lelisp dionysos /usr/distant/bin/lelisp
```

Le module de gestion de configuration intègre des fonctions d'accès aux informations des fichiers de configuration et des fonctions donnant les caractéristiques réseaux et matériels d'une machine (adresse internet et ethernet, type de machine...).

Gestion de la répartition

Le module gestion de la répartition cherche à équilibrer automatiquement la charge. Le mécanisme de migration de tâches mis en œuvre reste totalement transparent aux applications. L'introduction d'un tel mécanisme dans l'atelier permet d'obtenir un parallélisme effectif et une répartition de charge plus équitable.

Le module gestion de répartition que nous avons réalisé, intègre plusieurs fonctions assurant la distribution de processus. Deux techniques sont employées, l'une fournie avec le système SunOS, l'autre (GATOS) développée dans le laboratoire.

- Open Network Computing ONC [Garlick, et al., 1988]
ONC regroupe un ensemble de primitives réalisé par SUN, facilitant le lancement d'un processus sur une autre machine en spécifiant le nom de la machine. Le processus exécuté hérite du contexte système de processus initiateur de l'exécution.
- GATOS [Foliot and Ruffin, 1989]
Le système GATOS assure, par un gérant réparti de tâches, l'optimisation de l'utilisation des processeurs sur l'ensemble des machines. Son rôle consiste à déterminer, pour chacun des programmes, les machines susceptibles de supporter leurs exécutions. Les critères de choix d'une machine sont la charge, le type du processeur et la disponibilité des ressources logicielles (présence du code).

6.2. Niveau Plate-forme GUS

Dans cette partie, nous nous intéressons à la plate-forme Gestion des Utilisateurs et des Services et tout particulièrement au GUS-serveur que nous avons définie dans la partie II (§5.10 Architecture du GUS dans un environnement distribué). La plate-forme gère deux catégories de données, les données d'administration de l'atelier et les données personnelles liées aux énoncés des utilisateurs (partie II, §5.2 Gestion des utilisateurs).

Les données personnelles sont les énoncés et les résultats des applications pour chaque utilisateur. Elles sont représentées par les historiques sur les énoncés des utilisateurs sous forme arborescente (partie II, §5.8.3 Les résultats). La plate-forme GUS offre des primitives de mise à jour de ces historiques. L'ensemble de ces primitives d'accès aux données personnelles est réalisé par un module du GUS.

Les données d'administration regroupent les informations sur les utilisateurs, les formalismes et les applications. Conformément à l'analyse, la plate-forme Gestion de Station Virtuelle (GSV) et les programmes d'applications interrogent dynamiquement le GUS-serveur pour accéder aux données d'administration. Comme tout les programmes de l'atelier sous Unix, la plate-forme GSV et les programmes d'applications sont réalisés par des processus s'exécutant dans l'environnement distribué. Dans cette environnement, la plate-forme GUS assure l'accès de ses objets d'administration (utilisateurs, formalismes, applications) à partir de n'importe quelle machine de l'environnement. Pour connaître une information sur un objet, un processus (client) de l'atelier interroge la plate-forme GUS-serveur. Un élément majeur dans la gestion dynamique des objets est que l'administrateur de l'atelier peut à tout instant, d'une part modifier des informations sur les objets comme par exemple changer les droits d'un utilisateur ou passer une application de l'état test à l'état exploitation, et d'autre part ajouter des formalismes ou des applications.

Un des points faibles de beaucoup d'outils actuellement développés sous Unix, réside dans une mauvaise utilisation de l'arborescence Unix aussi bien pour accéder à leurs ressources, que pour créer les fichiers résultats (en particulier par des accès absolus aux fichiers). Pour nous affranchir de ces contraintes, la plate-forme GUS gère l'ensemble des objets de l'atelier et chaque processus interroge dynamiquement la plate-forme GUS pour accéder à ces informations.

Nous présentons la gestion des données personnelles des utilisateurs. Ensuite, nous décrivons les données d'administration gérés par la plate-forme GUS et les structures utilisées pour les stocker.

6.2.1. Représentation des données personnelles

Nous avons réalisé une bibliothèque assurant la gestion du contexte dynamique de l'utilisateur et la gestion des historiques sur les énoncés. Les données de l'utilisateur sont stockées dans une arborescence $Ami\{n^{\circ}version\}$ (Figure 6.2) dans le répertoire principal de l'utilisateur (~). La localisation de ce répertoire dans le système de fichier NFS est décrite dans un fichier d'administration du GUS (fichier password). Ce répertoire contient un fichier ".ami" (le contexte dynamique), des répertoires de nom $nom_formalisme.nom_énoncé$ représentant les énoncés utilisés et contenant les fichiers résultats des applications ($nom_application.nom_fichier$) et un fichier .info comportant la date d'estampille de l'énoncé.

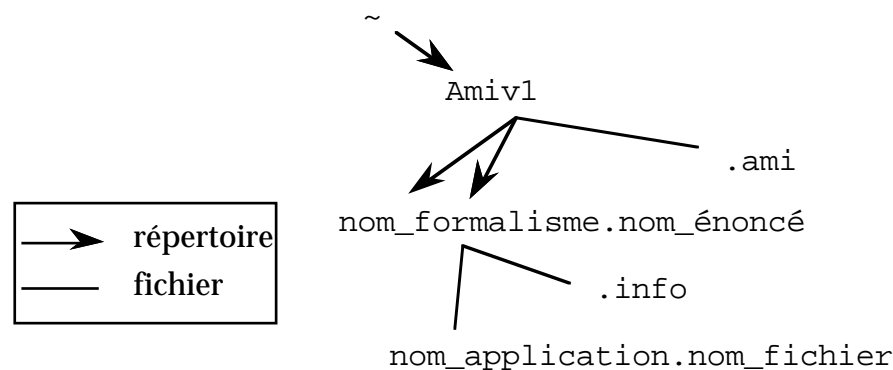


Figure 6.2. : Arborescence dans AMI de l'historique d'un énoncé

L'arborescence de la figure 6.2 reprend les caractéristiques définies dans la partie II figure 5.4.b.

Les processus ayant besoin d'accéder à l'arborescence des énoncés sont liés à la bibliothèque que nous venons de décrire.

Des outils d'administration sont nécessaires pour détruire les anciens énoncés et résultats devenus inutiles pour un utilisateur. Il peut être envisager d'utiliser les outils Unix pour détruire régulièrement les fichiers non utilisés depuis un certain temps (utilisation de la `crontab`).

6.2.2. Représentation des données d'administration

Le GUS-serveur met en œuvre des mécanismes assurant la gestion des utilisateurs, des formalismes et des services. Dans l'atelier AMI, la plate-forme GUS offre un accès aux noms des programmes d'applications intégrés dans l'atelier pour chaque formalisme. Nous intégrerons ultérieurement un mécanisme accès aux applications à partir de leurs services conformément à notre analyse au chapitre II.

Notre travail ne s'est pas orienté sur les problèmes de bases de données dans un système distribué. Les liens entre les objets utilisateurs, formalismes et applications sont représentables dans l'arborescence des systèmes de fichiers Unix. Les informations précises sur chacun des objets sont réunies au niveau des feuilles de l'arbre.

Les données d'administration de l'atelier sont structurées dans une arborescence dont la racine a pour nom `mars`. Ce répertoire contient trois fichiers: `erreur`, `machines`, `softs` et quatre sous-répertoires: `Formalismes`, `Applications`, `Utilisateurs`, `commun`.

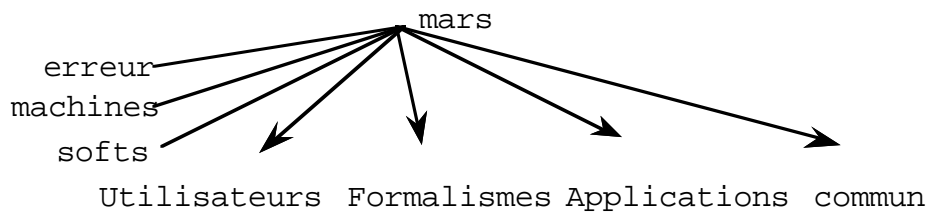
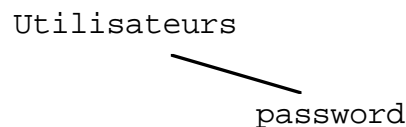


Figure 6.2.1.a: Structure arborescente des données du GUS

Le fichier `erreur` est géré par la plate-forme GUS et contient l'ensemble des erreurs qui sont intervenues lorsque l'atelier a accédé à cette arborescence. Les fichiers `machines` et `softs` font partie de la gestion de configuration.

a. Les utilisateurs

Le répertoire `Utilisateurs` contient un fichier `password` décrivant les utilisateurs de l'atelier.



Chaque ligne du fichier associée à un utilisateur est formée de trois champs séparés par ":".

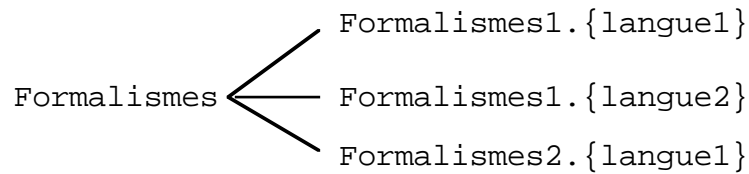
```

ligne utilisateur= nom_utilisateur:nom_login_unix:niveau:
niveau=ADMINISTRATEUR
      | (nom_application <droits>[;nom_application <droits>]*)
droits= développement | test | [exploitation]
  
```

L'administrateur de l'atelier est responsable de la cohérence des informations du fichier `password`. Pour chaque utilisateur de l'atelier, il lui associe des droits d'accès à chaque application qui correspondent au cycle de vie des applications (partie II, §5.7.1 Cycle de vie d'une application). Un droit d'accès `ADMINISTRATEUR` est défini pour que l'administrateur de l'atelier ait une vue complète de toutes les applications intégrées dans l'atelier.

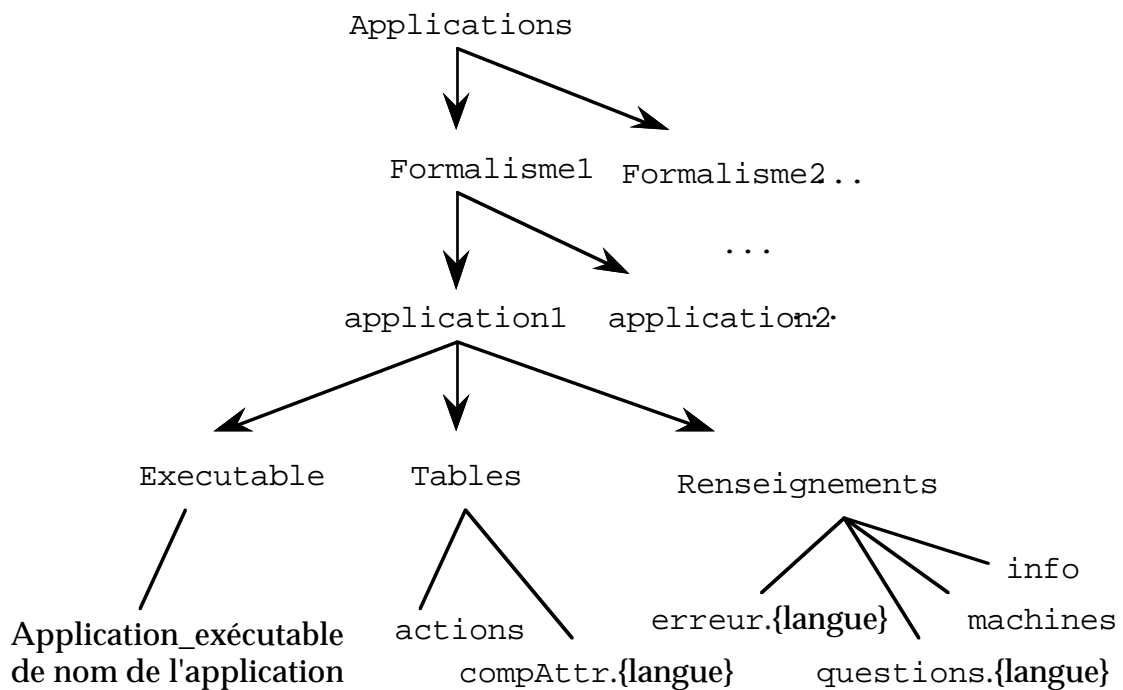
b. Les formalismes

Le répertoire `Formalismes` contient les fichiers décrivant les différents formalismes reconnus dans l'atelier. Ces fichiers sont écrits en langage de description des formalismes (CAMI-LDF). Ces fichiers sont suffixés par "." suivi d'un chiffre représentant la langue. La langue est codée sur un chiffre dans tous les programmes de l'atelier sous Unix.



c. Les applications

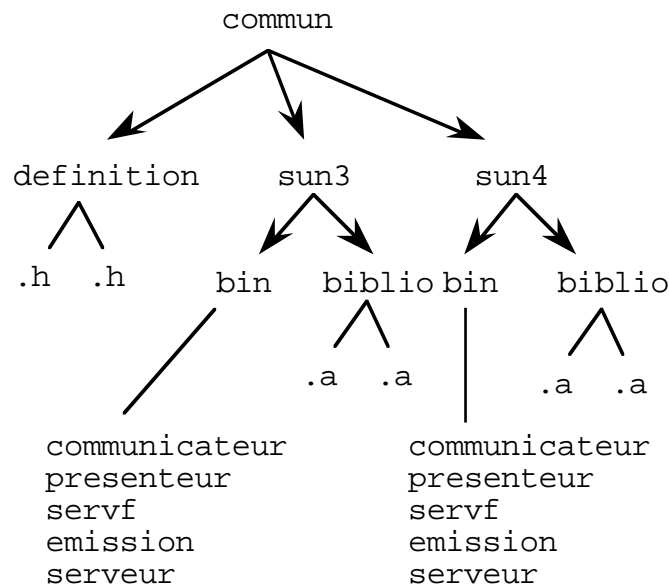
Le répertoire Applications décrit l'ensemble des applications de chacun des formalismes de l'atelier. Chaque formalisme est un sous-répertoire. Dans ces sous-répertoires, chaque application possède une arborescence (du nom de l'application). Chaque arborescence est divisée en trois sous-arborescences: Executable, Renseignements et Tables.



- Le répertoire “Executable” contient le code exécutable de l’application ainsi que ses fichiers de configuration.
- Le répertoire “Renseignements” sont les données de renseignement de l’application (§II.5.7.3 Renseignements sur les applications). Il contient quatre types de fichiers prédéfinis: les fichiers info, erreur.{langue} et questions.{langue}, machines.
 - Le fichier “info” donne l’état du service (mise au point...§II.5.7.1 Cycle de vie d’une application), une description succincte du service (une ligne par langue précédée du code de la langue) et des informations d’administration.
 - Le fichier erreur.{langue} est composé d’un ensemble de messages du services au format: n° de message message. Ce fichier contient tous les messages envoyés à l’utilisateur (erreurs, résultats,...)
 - Le fichier questions.{langue} contient l’arbre de questions en langage LAQ (§II.5.8.1 Le Langage d’Arbre de Questions).
 - Le fichier machines n’est pas obligatoire, il contient la liste des machines où l’application doit s’exécuter. Il correspond donc à une restriction des sites d’exécution.
- Le répertoire Tables contient le fichier actions décrivant la table de transcodage (§II.6.4.3.e Interface dynamique) pour l’application. Ce répertoire peut contenir un répertoire lexique (§II.6.4.3.e Interface dynamique) utilisé lors de résultats d’une question dans un autre domaine. Ce répertoire contient des fichiers de noms des formalismes résultats. Ce répertoire peut aussi contenir des fichiers compAttr.{langue} de complément de la table des attributs.

d. Les données communes

Le répertoire `common` contient des répertoires composés des bibliothèques (dont les points d’entrée sont décrits dans les fichiers “.h” du sous-répertoire `definition`) et des exécutables nécessaires à l’utilisation de l’atelier. Les noms de ces répertoires représentent le type de machine (ex: `sun3`, `sun4`...). Chacun de ces répertoires est formé de deux sous-répertoires `biblio` et `bin` correspondant aux bibliothèques, aux exécutables et aux fichiers prédéfinis pour les programmes en langage interprétés. Les fichiers symbolisés par `.a` sont des fichiers de bibliothèque Unix.



6.2.3. Le mécanisme client-serveur

Pour respecter les contraintes de gestion et d'interrogation dynamique du GUS-serveur, nous avons utilisé la méthode de client-serveur. Ce modèle de [Benzekri, 1989] règle les problèmes de gestion d'accès aux ressources communes.

Un client est un élément actif de l'environnement désirant opérer sur un objet. Le serveur interagit avec les clients en traitant leurs requêtes les unes après les autres et en leur rendant un compte rendu (réponse). Le serveur, quant à lui, s'occupe de la gestion matérielle de l'objet. Ce serveur connaît les données (objets) et fait appel à la gestion des fichiers distants (NFS) pour y accéder.

Nous avons décrit les objets gérés par la plate-forme GUS sous forme arborescente. L'arborescence des systèmes de fichiers Unix autorise la représentation des objets de la plate-forme GUS-serveur. L'utilisation du système NFS que nous avons définie dans notre extension système, assure l'accès au système de fichier supportant les données du GUS à partir de n'importe quelle machine du réseau.

La manipulation des objets entre les clients et le serveur est réalisée par le biais de primitives de communications (interface et gestion des communications) qui mettent en œuvre des échanges de structures d'objets avec des séquences de contrôles. Chaque programme de l'atelier interroge dynamiquement le GUS-serveur pour accéder à ces objets. L'administrateur de l'atelier peut ainsi modifier l'emplacement ou l'accès des objets du GUS-serveur sans avoir à modifier les applications. Le concept principal du mécanisme client-serveur est de rendre les accès aux fonctions assurées par le GUS indépendants de leurs traitements.

L'architecture du serveur GUS (Figure 5.3.1.a) que nous avons réalisée, est formée d'une partie assurant la gestion des communications et d'une partie traitant des requêtes et accédant aux données par le système NFS. Un client (service ou plateforme) fait une requête au GUS-serveur par l'intermédiaire d'une interface de communication. Nous avons réalisé une bibliothèque qui constitue l'interface entre le client et le serveur. Cette bibliothèque (`bibSV`) comprend l'ensemble des requêtes possibles. Son rôle est de construire les requêtes du côté client, de réceptionner la requête et de renvoyer la réponse, du côté serveur. Le programme client fait appel aux fonctions de la bibliothèque `bibSV` qui elle-même font appel à des fonctions spécialisées (interface de communication) pour le dépôt des requêtes. L'utilisation de cette bibliothèque par un programme offre la possibilité de transmettre des requêtes et de recevoir des réponses complexes. Les structures échangées entre le client et le serveur sont prédéfinies. Les structures sont composés des arguments passés au serveur et d'une sous-structure qui contient la réponse du serveur.

Dans notre réseau de machines Unix, nous avons choisi d'implanter sur chaque machine supportant l'atelier un serveur-GUS. Les processus clients accèdent aux objets du serveur par un ensemble de requêtes prédéfinies.

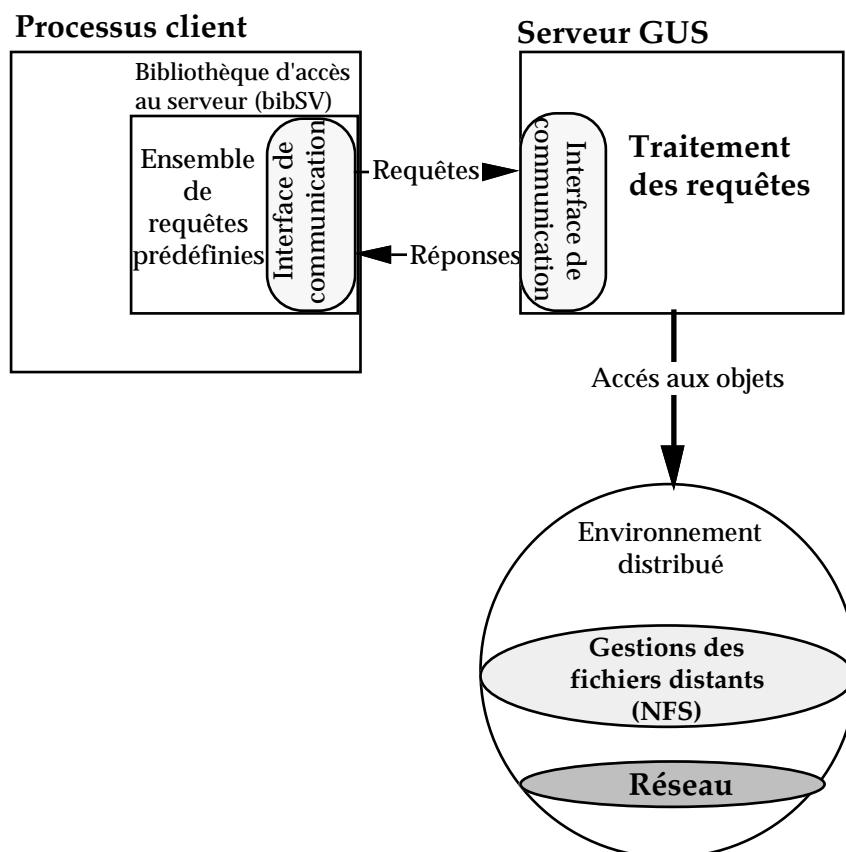


Figure 5.3.1.a : Architecture du GUS-serveur

L'utilisation du modèle client-serveur rend les requêtes complètement indépendantes de leurs traitements et évite de faire une édition de liens du processus client lors de modification de la réalisation d'une requête. L'indépendance des clients vis à vis de la méthode de stockage autorise l'administrateur de l'atelier à modifier dynamiquement aussi bien les objets du GUS-serveur que les techniques de stockage.

L'ajout de nouvelles fonctions d'accès aux objets ou de nouveaux objets se traduit par la modification de la bibliothèque bibSV en créant de nouvelles requêtes. Ce mécanisme autorise toujours les anciens programmes clients à interroger le serveur. Pour que les anciens clients accèdent aux nouvelles requêtes, il suffit de faire une édition de liens.

Pour rendre les applications de l'atelier indépendantes de la localisation de leurs fichiers d'initialisation, le serveur GUS gère l'emplacement de leur fichiers d'initialisation. Nous avons créé des requêtes particulières d'administration des objets du GUS.

Le serveur-GUS est un processus serveur cyclique non spécialisé (il attends une demande de service quelconque, rend le service et se remet en attente). Il est utilisé pour l'exécution des fonctions de la bibliothèque bibSV qui dépendent de l'arborescence NFS de l'atelier. Ainsi une modification de cette arborescence n'influera ni la bibliothèque bibSV ni les applications: il suffira de recompiler le programme serveur. Les traitements rendus par le serveur GUS constituent une bibliothèque spécialisée dans la gestion des objets de l'atelier (bibf). Une description détaillée de la réalisation sous Unix du serveur GUS avec un mécanisme de reprise sur erreur dans le cas d'une faute dans le traitement d'une requête est donnée dans "Réalisation d'un serveur d'objets dans l'atelier AMI" [CHEHAIBAR Ghassan, juillet 1987].

Les requêtes du serveur GUS

Nous présentons un certain nombre de requêtes de la bibliothèque (bibSV) d'accès au serveur GUS:

SVNomLogin (nomUtilisateur)

Cette fonction authentifie l'utilisateur vis à vis de l'atelier et du système Unix. Si l'utilisateur a le droit d'accès à l'atelier, cette fonction renvoie son nom de login sous Unix.

SVListeFormalisme ()

Rend la liste des formalismes connue de l'atelier.

SVCheminFormalisme (nomFormalisme, langue)

Rend le chemin absolu (Unix) du fichier qui décrit le formalisme en langage de description du formalisme (LDF) dans la langue de l'utilisateur.

SVListeApplication (nomFormalisme,nomUtilisateur)

Liste des applications pour un formalisme accessible à un utilisateur.

SVCheminApplication(nomFormalisme,nomApplication)

Rend le chemin absolu (Unix) du code exécutable de l'application.

SVCheminQuestion (nomFormalisme,application,langue)

Rend le chemin absolu (Unix) du fichier contenant l'arbre de questions en langage LAQ pour une application. Cette fonction prend en compte la langue de l'utilisateur pour accéder au fichier dans la bonne langue.

SVCheminTableTranscodage (nomFormalisme,application)

Rend le chemin absolu (Unix) du fichier contenant la table de transcodage pour une application.

SVRechercheFichier (nomFormalisme, nomApplication)

Rend le chemin absolu (Unix) du répertoire qui contient les fichiers d'initialisation de l'application.

SVListeMachineExec (application)

Retourne une liste des machines sur lesquelles s'exécutent l'application passée en argument. Cette requête est traité par le serveur en utilisant les fichiers liés à la gestion de configuration.

SVRechercheSoft (nomLogiciel,nomDeMachine)

Retourne le chemin absolu sur la machine pour accéder à un logiciel. Cette requête est traité par le serveur en utilisant les fichiers liés à la gestion de configuration.

6.3. Niveau Plate-forme GSV

L'analyse de la partie II de la plate-forme GSV (Gestion de Station de travail Virtuelle) (§II.6.3 Architecture en couches de la plate-forme GSV) montre que ses couches s'exécutent de manière concurrente. Les couches du niveau transport à application dialoguent par messages CAMI en utilisant les extensions systèmes de communication. La gestion multi-sessions engendre plusieurs exécutions d'entités d'une même couche (§II.6.4.2.c Multi-fenêtres et multi-sessions).

Nous avons donc programmé ces couches en affectant à chaque entité d'une couche un processus. Ainsi, de façon précise, on définit les fonctions de chaque processus. Chaque processus manipule seulement les objets définis pour chaque sous-couche et pour échanger des informations utilise des messages CAMI spécifiques pour chaque couche. Les services offerts par chaque processus sont donc définis par le traitement de ces messages CAMI. Pour la réalisation des processus, nous avons utilisé au maximum la notion de modularité du langage C. Un groupe de modules définit les objets manipulés ainsi que les méthodes appliquées sur ces objets. Un module reconnaît les messages CAMI et un autre module initialise le processus (ouverture des liaisons de communication avec les couches adjacentes, armement des signaux systèmes...). La création des entités des couches est faite de la couche transport à la couche application.

Cette technique de réalisation de la plate-forme GSV garantit un masquage et une protection des objets d'un couche. Il est ainsi possible de modifier le protocole d'une couche sans affecter le fonctionnement des couches adjacentes. La mise au point d'un protocole est simplifiée par l'analyse du déroulement du protocole en utilisant un dévermineur sur l'entité de la couche sans affecter le reste des couches de la plate-forme GSV. De la même manière, nous avons réalisé des outils de trace des protocoles associés à chaque couche du GSV.

La figure 6.3.a. présente l'architecture en couches du GSV.

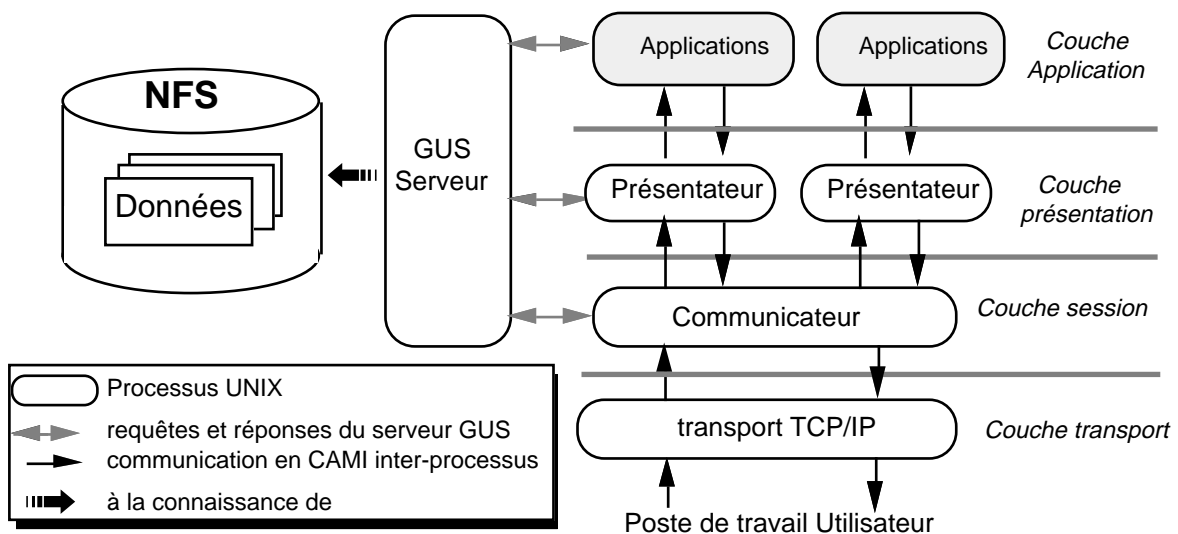


Figure 6.3.a: Réalisation de l'architecture en couches du GSV par processus

Nous présentons la réalisation des couches transport, session et présentation en détaillant les problèmes résolus dans la couche transport et les techniques employées pour la gestion de l'arbre de questions et de la table de transcodage de la couche présentation.

6.3.1. La couche transport

Conformément à l'analyse de la partie II, la couche transport intègre deux sous-couches, le gérant des utilisateurs et le transport d'information. Nous présentons la réalisation dans un système multi-tâches des deux protocoles de connexion associés à la sous-couche de transport d'information. Nous avons réalisé plusieurs protocoles de connexion présentant des interfaces uniformes pour mettre en communication l'utilisateur et la structure d'accueil au moyen de plusieurs types de liaisons. La couche transport sous Unix autorise une connexion par ligne série ou par TCP/IP.

Ligne série

La couche transport réalise une connexion point à point et assure le transport fiable d'informations. MACAO s'exécutant sur un système mono-processus privilégiant l'interface utilisateur peut ne pas être en état de recevoir ou de traiter assez rapidement les messages provenant de la structure d'accueil. L'indisponibilité du système mono-tâche et la communication par ligne série risqueraient d'engendrer des problèmes lors des transferts de données (perte, déséquence des messages). C'est pourquoi, nous avons conçu, au niveau de la couche transport, un protocole (Pomme-Soleil) de liaison Apple-Sun garantissant la fiabilité des transferts. Il intègre aussi un mécanisme de contrôle de flux implicite à base d'une fenêtre d'émission [HDLC] et un contrôle périodique des connexions basé sur l'utilisation de temporisateurs.

Dans le cas d'une connexion par ligne série, l'utilisateur sur le Macintosh est obligé de se connecter sur la station Unix, pour cela l'application d'interaction utilisateur comporte un émulateur de terminal (en mode tty) pour permettre à l'utilisateur de se "loguer" sous Unix. l'utilisateur est donc obligé dans ce cas de rentrer son nom de login puis son mot de passe. Ensuite, il exécute un programme de l'atelier qui envoie un code à l'application d'interaction utilisateur qui valide la mise en place du protocole Pomme-Soleil de communication par ligne série.

Le but de ce paragraphe n'est pas décrire le protocole Pomme-Soleil mais de montrer les techniques employées pour réaliser un protocole de communication sur une machine multi-tâches. Nous avons réalisé le protocole Pomme-Soleil en langage C, les modules de gestion des fenêtres et des mécanismes de réémission sont identiques sur la station Unix et le Macintosh.

Le protocole Pomme-Soleil permet l'émission et la réception simultanées de trames de longueurs variables. L'utilisation d'un système multi-tâches permet de d'utiliser deux tâches concurrentes indépendantes, une pour la réception de trame et l'autre pour l'émission. Le protocole Pomme-Soleil étant basé sur le protocole HDLC, le processus responsable de l'émission possède une fenêtre d'émission. Le processus de réception reçoit les trames qui contiennent les accusés de réception des trames émises. Il est donc nécessaire de passer les accusés de réception au processus émetteur ainsi que les acquittements. Les processus émetteur et récepteur s'échangent ces informations en utilisant des messages spécifiques CAMI.

La figure 6.3.b met en évidence les deux processus (émetteur, récepteur) ainsi que la communication par messages CAMI. Tant que la fenêtre d'émission n'est pas remplie, le processus émetteur prend les messages du communicateur (§6.3.1 La couche session: le communicateur) pour les émettre. Le processus de réception est toujours en attente de trames sur l'entrée série. Quand une trame arrive, il la désencapsule, transmet le message au communicateur (couche session) et envoie au processus émetteur un message CAMI qui contient les accusés de réception. Le processus émetteur peut donc recevoir à la fois des messages du communicateur et du processus récepteur. Des essais, nous ont montré que la politique employée par le processus émetteur pour le choix de réception de ces messages influe sur la performance du protocole Pomme-Soleil. Nous avons donc privilégié la réception de message provenant du processus récepteur.

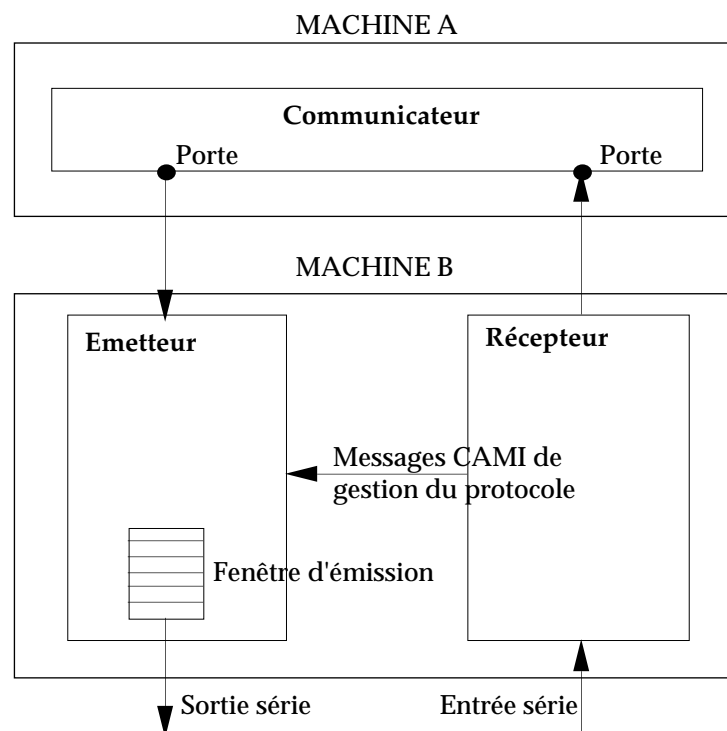


Figure 6.3.b: Réalisation de l'architecture en couches du GSV par processus

Par TCP/IP

La connexion par TCP/IP est réalisée par un serveur (serveur de communication) s'exécutant sur les stations Unix. Un serveur de communication est en attente d'une demande connexion à une porte prédéfinie. Quand la connexion est établie entre le Macintosh et ce serveur, MACAO transmet en CAMI le nom de l'utilisateur pour l'atelier et son mot de passe Unix. Par appel au GUS-serveur, le serveur de communication identifie l'utilisateur (§II.6.4.1 Couche transport) et vérifie l'unicité de connexion de l'utilisateur (présence du fichier .ami dans l'historique des énoncés). Dans le cas où l'utilisateur est reconnu et peut se connecter, le serveur de communication utilise une fonction de l'extension système pour créer un nouveau processus assurant le transport de l'information. Ce nouveau processus est responsable de la réception et de l'émission de messages CAMI vers MACAO. Il teste, par appel au module du GUS de gestion de l'historique des énoncés, la présence d'une entité d'une couche session (processus communicateur) sur une des machines du sites (adresse de transport de la couche session dans le fichier .ami). Si le processus communicateur est présent alors il s'y connecte sinon il le crée.

6.3.2. La couche session: le communicateur

Le communicateur est le processus traitant toutes les fonctionnalités de la couche session (§II.6.4.2 Couche session). Il vérifie le numéro de version et assure l'ouverture et le contrôle simultanés de plusieurs sessions par l'utilisation d'une liste. Lors de sa création le processus communicateur met à jour le contexte dynamique de l'utilisateur par appel au module du GUS de gestion de l'historique des énoncés. Ce module du GUS crée alors le fichier ".ami" dans l'historique des énoncés de l'utilisateur.

L'utilisateur peut terminer sa session de travail en laissant des services en arrière plan (§II.6.4.2.d Fermeture de la couche session). Dans ce cas, l'émetteur et le récepteur se termineront, le communicateur restera en place et le fichier ".ami" contiendra les adresses des portes du communicateur pour une reconnexion ultérieure (Figure 6.3.b).

6.3.3. La couche présentation: le présenteur

Le présentateur est le processus traitant toutes les fonctionnalités de la couche présentation. Il compare les droits des utilisateurs aux différentes applications pour lui en proposer certaines. Le présentateur connaît l'état global de l'arbre de questions de l'application (gérant des dialogues). Il informe l'utilisateur sur de l'état de ses questions en cours d'exécution. L'interface dynamique de présentation des données facilite l'intégration d'applications n'ayant pas été conçues pour accepter la description textuelle des graphes en langage CAMI. Elle transcode, par des règles de réécriture, le sous-ensemble des commandes CAMI de cette couche (langage LDE), en des actions propres à chaque application. A cette étape, les énoncés sont décrits par des commandes CAMI énumérant tous les nœuds (CN) puis tous les arcs (CA). Chacun de ces objets est suivi par la liste de ses attributs (CT) issus de la table du formalisme. Le présenteur assure l'exécution des programmes d'applications conformément à la description de la partie II (§6.4.3.b Gérant des services)

6.4. Niveau interface: le pilote

6.4.1. La liaison entre la structure d'accueil et le pilote

L'énoncé parvient au pilote sous la forme d'une suite de messages parenthésée par des messages de contrôle qui sont "DB()" et "FB()".

Lors de la transmission d'une question, la plate-forme GSV transmet en fait un message de type chaîne de caractères dans lequel est codé l'arbre de questions. Après réception du message, le pilote reconstitue l'arborescence.

A une question nous associons un certain nombre d'options. L'atelier transmet à l'application un arbre de questions correctement initialisé. Une question est un message. C'est une chaîne de caractères débutant par un caractère particulier : '*', suivi par le nom de la question, puis des différentes options à prendre en compte, éventuellement. Chacune de ces options peut se redécomposer sur plusieurs niveaux en d'autres options. Ce phénomène d'imbrication est reflété à l'aide du parenthésage adéquat. A chaque parenthèse ouvrante, on descend d'un niveau d'option. On connaît le nombre d'options pour chacun des niveaux d'options. On associe une variable à chaque option. Lorsqu'elle est active, la variable est à 1 sinon elle est à 0. Nous avons donc des messages du type : * question (0 ((1 0) 1) 0)

Le pilote peut réceptionner plusieurs questions, donc, plusieurs messages. Ces messages de questions sont parenthésés par des messages de contrôle qui sont "DT()" et "FT()". Toutefois, l'application et le pilote doivent être construits de telle sorte qu'ils puissent gérer plusieurs questions en même temps.

6.4.2. La liaison entre le pilote et la structure d'accueil

Le mode de communication sur la liaison structure d'accueil vers pilote et pilote vers structure d'accueil sont différents. Dans le premier cas, on communique par messages et dans le second on est totalement dépendant des fonctions qui existent au niveau de l'atelier. Ces fonctions sont ajoutées au fur et à mesure des besoins.

Lors de la sélection par un utilisateur d'un service, l'atelier lance le pilote correspondant. Celui-ci s'initialise : il demande à établir une communication avec la plate-forme GSV. Son initialisation débute par l'appel de la procédure "InitCAMI_a()" et prend fin lors de l'appel de la procédure "FinInitCAMI_a()". A noter que la procédure "InitCAMI_a()" retourne un booléen permettant de déterminer si l'application a déjà travaillé sur l'énoncé traité. Une telle information manque de finesse mais c'est au pilote d'affiner un tel diagnostic puisqu'il est très "proche" de l'application. Lorsque le pilote veut rompre la communication il fait appel à la procédure "ArretCAMI_a()".

Par ailleurs, il est intéressant de renseigner l'utilisateur sur l'état d'une question. Pour cela, nous offrons d'une procédure :

```
EnvoiEtatQuestion(NomQuestion,EtatQuestion,Message)
```

Un état est déterminé pour une question donnée (un nom de question), un état choisi parmi les états prédéfinis au sein de l'atelier (inactif, en cours, terminé...) et un message destiné à l'utilisateur via une fenêtre textuelle. Cette procédure sert aussi lors de l'initialisation du pilote. En effet, celui-ci détermine les questions que l'application a déjà résolues et envoie l'état correspondant qui sera visualisé côté utilisateur. Ici, le pilote a un rôle de diagnostic.

Enfin, cette liaison est le support de transmission des résultats au demandeur du service. Le pilote dispose, grâce à l'environnement de l'atelier, d'une gamme variée de fonctions lui permettant de transmettre ces résultats ou de mettre en évidence un objet élémentaire. Grâce à une fenêtre textuelle (fenêtre historique), le pilote émet des commentaires et affiche des résultats bruts.

Il peut également mettre en évidence tel ou tel objet dans l'énoncé source. Il peut créer un nouvel objet, en détruire un autre. Mais bien souvent, les résultats sont structurés en ensembles d'objets élémentaires. Le pilote demande à créer une vue sur l'énoncé pour chaque ensemble.

6.4.3. Définition d'un pilote générique pour l'atelier AMI

Un pilote générique pour AMI, c'est un pilote associé à une classe d'énoncé. Les variantes d'un tel pilote dépendent du type de l'application associée, ce qui peut faire varier son architecture. Le pilote est chargé de coordonner des sous-pilotes chargés d'un traitement particulier. Nous avons défini les sous-pilotes suivants: le sous-pilote d'initialisation, le sous-pilote de diagnostic, le sous-pilote d'énoncé, le sous-pilote de questions, le sous-pilote de résultats, le sous-pilote d'appel.

Le sous-pilote d'initialisation

Le sous-pilote d'initialisation est chargé d'établir la communication avec l'atelier. Il prépare, si nécessaire, l'environnement d'exécution de l'application, initialise des variables d'environnement, crée des répertoires. Toutefois, si l'application a déjà travaillé sur l'énoncé, une telle démarche est évitée puisque, déjà réalisée.

```
SP_Initialisation()  
{  
    déjà_travaillé = InitCAMI_a() ;  
    if (not déjà_travaillé) PréparerEnvironnementApplication() ;  
    FinInitCAMI_a() ;  
}
```

Le sous-pilote de diagnostic

Il s'agit pour ce sous-pilote de déterminer toutes les questions ayant été résolues par l'application. Pour cela, il accède au contexte de l'application et teste la présence de fichiers résultats. Le sous-pilote est toujours très au fait, des conventions implicites de l'application et tient donc compte de la non-standardisation des résultats au sein de l'atelier. Le sous-pilote envoie un état de la question en utilisant la procédure :

```
EnvoiEtatQuestion(NomQuestion, QuestionCalculée, MessageEventuel)
```

```
SP_Diagnostic()  
{  
    Pour toutes les questions possibles  
    Faire  
        Si question déjà calculée  
        Alors Mettre à jour les informations fournies par l'interface utilisateur  
    FinFaire  
}
```

Le sous-pilote d'énoncé

Il y a autant de sous-pilotes d'énoncés qu'il y a de classes d'énoncés définies pour l'atelier. Chaque sous-pilote est alors chargé de réceptionner les messages CAMI émis grâce à la table de transcodage. La structure d'accueil du sous-pilote est basée sur les listes chaînées. Nous avons prévu en général un maximum d'attributs même si l'application n'en utilise qu'une partie. Ceci a l'avantage de ne pas trop particulariser le sous-pilote vis à vis d'une application donnée afin de le réutiliser pour un groupe d'applications.

```
SP_Enoncé()  
{  
    RéceptionClasseEnoncé() ;  
}
```

Le sous-pilote de questions

Le sous-pilote de question a deux tâches : il réceptionne la question sous la forme d'un message, reconstitue l'arbre de questions, puis ensuite l'analyse pour déterminer le service demandé à l'application.

```
SP_Question()  
{  
    RéceptionQuestion() ;  
    AnalyseQuestion() ;  
}
```

Le sous-pilote de résultats

Le sous-pilote de résultats accède au contexte de l'application pour y retrouver les résultats générés par l'application. Il les traduit dans le formalisme de sortie vers l'atelier. Ensuite, il envoie ces résultats vers l'utilisateur en utilisant différents procédés, par exemple, des mises en évidence d'objets, du texte, des créations d'attributs.

```
SP_Résultats()  
{  
    AnalyseRésultats() ;  
    EnvoiRésultats() ;  
}
```

Le sous-pilote d'appel

Le sous-pilote traduit l'énoncé dans le formalisme de l'application et le lui transmet soit de façon directe (pilote interne), soit de façon indirecte via un moyen de communication. Il fait de même avec la question posée avant d'appeler l'application. Il peut s'agir d'un appel de procédure (pilote interne) ou d'un appel de processus (pilote interne et externe).

```
SP_Appel()  
{  
    TransmissionEnoncéVersApplication() ;  
    TransmissionQuestionVersApplication() ;  
    AppelApplication() ;  
}
```

6.5. Niveau applications

Les applications dans l'atelier sont divisées en deux catégories, d'un côté les applications d'édition d'énoncés et de l'autre les outils d'analyse. Pour chacune de ces catégories, des applications ont été spécifiées et développées: Une interface graphique (MACAO) décrite ci-dessous et un système expert (NIMA).

Les applications d'analyse sont elle-mêmes divisées en deux classes; celles qui construisent dynamiquement le réseau créé par l'interface utilisateur (applications incrémentales) et celles qui n'acceptent que le réseau dans sa totalité (applications atomiques). Une application incrémentale doit être capable de vérifier, à la demande, la sémantique et la validité du réseau partiel analysé.

6.5.1. Système Expert MIAMI

La réalisation des outils est effectuée soit de manière algorithmique, soit en utilisant un système expert spécialisé. En effet, d'une part, la complexité des analyses des modèles, des traductions de spécification et de la génération de code et, d'autre part, l'intégration rapide de nouvelles propriétés ou classes de réseaux rendent souhaitable un système expert d'aide à la formulation et à la déduction des propriétés de réseaux.

Les moteurs d'inférence classiques s'avèrent mal adaptés à ces domaines d'utilisation en raison des difficultés combinatoires, de la lenteur d'interprétation des règles et de l'inefficacité des représentations statiques des connaissances. Il est primordial que les énoncés de la théorie soient exprimables de manière assez naturelle et efficace dans les règles.

Un moteur d'inférence [Beldiceanu, 1987, Beldiceanu, 1988b] d'ordre 1 original a été conçu et réalisé en tenant spécialement compte des problèmes spécifiques posés par les graphes et les réseaux. L'atelier AMI intègre ce système expert MIAMI spécifique d'analyse de graphes et réseaux faisant coopérer des algorithmes et des règles de déductions. L'utilisation de MIAMI facilite l'intégration de théorèmes complexes sur les réseaux de Petri à l'aide de règles très proches de leurs définitions mathématiques. Le système expert s'étend par ajout de nouvelles actions, contraintes et règles. Les services de l'atelier sont ainsi réalisables en combinant les programmations algorithmiques et logiques.

6.5.2. Intégration d'applications

Nous montrons que l'intégration d'une application dans un atelier de spécification est soumise à des contraintes et qu'elle se déroule en plusieurs étapes.

6.6.1 Les contraintes de l'opération d'intégration

Il y a en premier lieu une contrainte de **temps**. L'intégration d'une application doit être réalisée le plus rapidement possible.

La contrainte d'**efficacité** est tout aussi fondamentale : l'opération d'intégration doit produire le résultat attendu à un coût minimum. L'application intégrée doit réaliser les mêmes fonctions que l'application autonome. Par ailleurs, le coût de l'intégration d'une application doit être bien inférieur à son coût de développement. Pour être rentable, l'intégration d'une application dans un atelier de spécification doit être au moins aussi efficace et surtout plus rapide que la réécriture spécifique de l'application pour l'atelier.

Il faut définir une méthode d'intégration pour faciliter l'ajout de nouvelle application, à partir de laquelle sera défini l'enchaînement systematique des étapes de l'opération d'intégration. Cependant, la définition d'une méthode d'intégration se heurte à deux difficultés majeures : la grande variété des applications et la diversité des groupes d'acteurs humains impliqués dans l'intégration d'une application.

Les acteurs de l'intégration

L'intégration d'une application dans un atelier logiciel est un travail d'équipe. Elle nécessite la collaboration d'acteurs humains aux profils différents, et appelés chacun à jouer un rôle précis. Nous avons mis en évidence quatre type d'utilisateurs: les utilisateurs finals, l'administrateur des formalismes, les concepteurs d'applications et l'administrateur de l'atelier. D'après les droits de ces utilisateurs (§5.2 GUS: gestion des utilisateurs), les concepteurs jouent un rôle privilégié dans l'intégration des applications. Cette catégorie se décompose en deux sous-catégories: les concepteurs d'applications et les intégreurs.

les concepteurs d'applications

Les concepteurs d'applications n'ont pas nécessairement de contact avec un atelier de spécification. Ils travaillent parfois en marge, et les applications qu'ils programment sont souvent conçues pour être autonomes.

L'intégration implique différemment le concepteur de l'application suivant qu'il s'agisse d'une **intégration a priori** ou d'une **intégration a posteriori** (§II.8.1.3 L'intégration d'application). La différence réside dans l'impact de la perspective d'intégration sur la conception de l'application. Si le concepteur "pense" son application en vue de son intégration (intégration à priori), il oriente sa conception en vue de simplifier l'opération d'intégration et de réduire le travail d'interface.

Par contre, si l'application existe déjà (intégration à posteriori), le concepteur ne participe pas systématiquement à l'intégration. Dans le meilleur des cas, il précise certains détails portant sur la conception de son application et nécessaires à l'opération d'intégration.

Dans la mesure où l'opération d'intégration permet à une application, de s'associer à d'autres applications, afin de réaliser, mais aussi de bénéficier d'un ensemble de services, le concepteur d'applications voit l'intégration comme une fonction déterminante pour la diffusion et l'utilisation généralisée de ses applications.

les intégrateurs d'applications

Les intégrateurs d'applications sont des programmeurs qui ont déjà intégré des applications. Ils sont spécialisés dans l'aide à l'intégration de nouvelle application.

Les intégrateurs sont préoccupés par l'aspect opératoire de l'intégration. Ils ont pour rôle de définir et d'appliquer l'opération d'intégration en tenant compte de ses contraintes. Les intégrateurs ne perçoivent pas la fonction d'intégration, ils se contentent de voir l'intégration comme un assemblage de composants logiciels.

Les étapes de l'intégration

L'opération d'intégration [Foughali, 1990] est une succession d'étapes qui sont chacune une séquence incluse dans la suite d'actions réalisant la fonction d'intégration.

Par souci d'efficacité des d'intégrations, toute étape inutile doit être absolument évitée. Par conséquent, une étape commencée ne doit jamais échouer. C'est pourquoi des évaluations critiques préliminaires de l'intérêt de l'application et de son coût d'intégration doivent être faites très soigneusement. Le passage à l'étape suivante est subordonné à une vérification déterminant si l'opération doit continuer, ou au contraire, si elle doit être arrêtée.

Même pour notre atelier spécialement conçu pour assurer l'objectif d'ouverture (Objectif 16), l'intégration d'une application non conçue pour l'atelier est une opération difficile. Afin d'éviter de s'engager dans des opérations non rentables des tests d'intérêt et d'intégrabilité ont été introduits.

Le **test d'intérêt** consiste à mesurer la pertinence de l'intégration d'une application. Il évalue d'une part, l'intérêt de nouveaux services ou de services plus efficaces, et d'autre part, les avantages potentiels pour l'application.

Le **test d'intégrabilité** met en question la faisabilité de l'opération d'intégration. Cette étape permet de rejeter une application non intégrable dans l'atelier logiciel considéré. Les critères de bonne intégrabilité sont:

- une interface utilisateur indépendante,
- un traducteur "identifiants internes-identifiants externes" indépendant,
- une architecture logicielle faiblement couplée,
- un comportement déterministe,
- une documentation complète.

L'étape d'**intégration effective** consiste d'abord en la phase de réalisation du pilote de l'application, une interface chargée d'assurer le dialogue entre l'application et la plate-forme GSV. Puis, la phase d'assemblage rattache physiquement l'application et son pilote, à la structure d'accueil de l'atelier.

La dernière étape le **test de validation** permet essentiellement de faire un bilan général de l'opération d'intégration.

6.6. Les applications intégrées

Ce chapitre a pour but de montrer la variété des applications déjà intégrées dans l'atelier, soit développées spécifiquement pour l'atelier par d'autres chercheurs du laboratoire MASI, soit en provenance de plusieurs autres laboratoires de recherches. Les différentes techniques que nous avons utilisées dépendent de la manière dont ces applications étaient conçues et surtout de leurs interfaces avec le système d'exploitation.

6.6.1. ARP

ARP (Analyse de Réseaux de Petri) est la toute première application à avoir été intégrée dans l'atelier AMI [Bernard, et al., 1985, Hein, 1988]. Elle sert à l'analyse de réseaux de Petri ordinaires et permet notamment de calculer les flots d'un réseau.

6.6.2. COMBAG

COMBAG (COMputation of a BAsis and a set of Generators of semi-flows) est une application regroupant un certain nombre de méthodes de calcul de familles génératrices de semi-flots pour les réseaux de Petri ordinaires et les réseaux de Petri colorés [Trèves, 1987]. COMBAG permet aussi de calculer les flots d'un réseau de Petri. Il s'agit d'une application développée au L.R.I (Université d'Orsay Paris XI) et intégrée donc à posteriori [Foughali and Masouni, 1990].

L'interface utilisateur de COMBAG n'étant pas dissocié de l'application, le problème a été reporté au niveau du pilote qui a donc simulé le comportement utilisateur. De même que pour CHP, il y a une visualisation graphique des semi-flots.

6.6.3. CHP

CHP (Couvreur Haddad Peyre du nom de ses concepteurs) est une application intégrant un nouvel algorithme de calcul des familles génératrices de semi-flots dans les réseaux à prédicats / prédicats unaires [Couvreur, 1990]. Il est également envisagé d'y ajouter le calcul des trappes et verrous ainsi que les mêmes services pour les réseaux de Petri ordinaires.

Le pilote dispose d'un certain nombre de primitives de mise en valeur graphique des résultats et qui ont été mises à contribution. Par ailleurs, il a dû se charger des problèmes de nommage des objets

Ce sont les concepteurs qui ont eux-mêmes procédé à l'intégration de leur application dans l'atelier AMI.

6.6.4. GrapheCouverture

GrapheCouverture calcule le graphe de couverture d'un réseau de Petri ordinaire [Finkel, 1990].

6.6.5. ECS

ECS (Earliest State Graph) est une application qui permet de calculer le graphe d'états au plus tôt d'un réseau de Petri T-temporisé [Amarger, 1988]. Ce logiciel avait fait l'objet d'une première étude à l'institut de mathématiques appliquées d'Angers. Il s'agit donc d'une intégration à posteriori.

6.6.6. GreatSPN

GreatSPN (Great Stochastics Petri Nets) est une application permettant, entre autre, de vérifier un grand nombre de propriétés structurelles (ensembles de conflits structurels, exclusion mutuelle, ...) sur les réseaux de Petri stochastiques ainsi que sur les réseaux de Petri temporisés. GreatSPN a été développée à l'Université de Turin (Italie) [Chiola, 1986].

GreatSPN ayant ses propres conventions de nomination des objets, c'est le pilote qui se charge de faire la correspondance avec les objets AMI. L'ensemble des résultats est visualisé graphiquement.

6.6.7. Le système expert MIAMI

MIAMI est un Moteur d'Inférences pour l'atelier AMI [Dagron, 1989a]. MIAMI permet principalement, de générer des applications à partir de paquets de règles qui sont moulinés par le moteur d'inférences. Actuellement, MIAMI peut utiliser deux formalismes : les réseaux de Petri ordinaires et les réseaux RPAS. Ainsi, pour un réseau de Petri ordinaire, nous pouvons:

- faire de la simulation,
- vérifier des propriétés structurelles (réseau pur, graphe d'événements, graphe simple,...),
- procéder à des réductions (réductions "de Berthelot",...).

Dans un futur proche, vont être implémentés la décomposition d'un réseau de Petri ordinaire et le calcul des trappes et verrous.

Concernant l'intégration de MIAMI dans l'atelier AMI, il y a d'abord eu une traduction du source initial (écrit en modula2) en langage C et ensuite la phase classique d'intégration [Bernard, 1989]. Il s'agit donc d'une intégration à posteriori. Le pilote a la charge de faire la correspondance des noms Macao et ceux de MIAMI. Les créations et destructions d'objets ainsi que les mises en valeur et sélections d'objets sont intensivement utilisées notamment lors des réductions successives d'un RdP. De plus, le pilote de MIAMI a eu à résoudre les problèmes de transmission de faits.

A noter que MIAMI permet d'éviter la phase d'intégration puisque les applications éventuellement générées par MIAMI sont de fait déjà intégrées à AMI.

6.6.8. Formalisme RPAS

AMI a permis de définir de nouveaux formalismes, l'un d'entre eux étant le formalisme RPAS [Dagron, 1989b] qui modélise des canaux évolués rendant transparent les types de communication.

MIAMI permet de valider des réseaux RPAS après les avoir traduit en réseaux de Petri ordinaires. Après analyse, il y a remontée des résultats au niveau RPAS en faisant la correspondance entre les objets des deux formalismes [Dagron, 1989c].

De plus il existe une application permettant de passer des spécifications textuelles du formalisme RPAS (LPAS) dans le formalisme RPAS lui-même [Dagron, 1989, Dagron, 1990].

6.6.9. TAPIOCA

TAPIOCA (Traduction Analyse et Prototypage Interactif en code OCCAM) est une application qui travaille à partir de réseaux de Petri ordinaires et d'un certain nombre d'informations comme les flots du réseau [Bréant, 1990]. A partir de là on calcule les processus figurant dans le réseau d'où la génération de code OCCAM. On recherche une solution optimale avec un dialogue interactif avec l'utilisateur pour un raffinement successif. On peut ainsi faire de la modélisation d'architecture avec placement sur celle-ci. L'architecture cible est une architecture à base de transputers. De plus, grâce à AMI, on peut procéder à la visualisation des graphiques correspondant aux réseaux de transputers.

6.6.10. RdP/TAGADA

TAGADA (Traduction Analyse et Génération en code Ada) permet à partir d'une classe de réseaux de Petri ordinaires d'obtenir du code Ada exécutable [Kordon, 1990]. Au départ, une décomposition en processus doit être associée au réseau pour obtenir un prototype exécutable en Ada. A noter que le pilote de l'application permet de faire appel au compilateur Ada et de lancer l'exécution du programme obtenu.

Parallèlement à TAGADA, il existe une autre application dérivée (Animation) qui permet de visualiser une exécution d'un PABX, sous une forme graphique en tenant compte des ordres donnés par le programme Ada et destinés à Macao qui se charge de l'interface graphique.

6.6.11. Intégrations en cours

L'atelier AMI est utilisé de manière expérimentale dans les universités de Hambourg et de Toulouse (IRIT) respectivement pour des applications d'algorithmique distribuée écrites en C et de théorie de graphes écrites en Lisp.

7. Utilisation de l'atelier

7.1. Introduction

Nous détaillons une utilisation complète de l'atelier AMI en montrant l'évolution dans le temps de deux entités: l'interface utilisateur Macao et la structure d'accueil.

Le scénario choisi montre l'utilisation de Macao en mode autonome, la phase de connexion à l'atelier avec les négociations nécessaires pour la reconnaissance de l'utilisateur, l'ouverture d'une session pour le dialogue avec un service, les possibilités de multi-sessions (commutation de session) la déconnexion et enfin la reconnexion.

7.2. Mode autonome de Macao

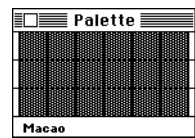
Comme pour toutes les applications sur Macintosh, l'utilisateur dispose de plusieurs méthodes pour ouvrir et créer un fichier. Soit il ouvre un ancien fichier en cliquant directement sur ce fichier, soit il ouvre d'abord l'application et demande ensuite l'ouverture de son fichier ou la création d'un nouveau fichier. Nous allons utiliser la deuxième méthode dans cet exposé.

L'utilisateur ouvre l'application Macao en cliquant sur son icône. Après l'initialisation deux fenêtres apparaissent: une **fenêtre d'historique** et une fenêtre **palette** vide.

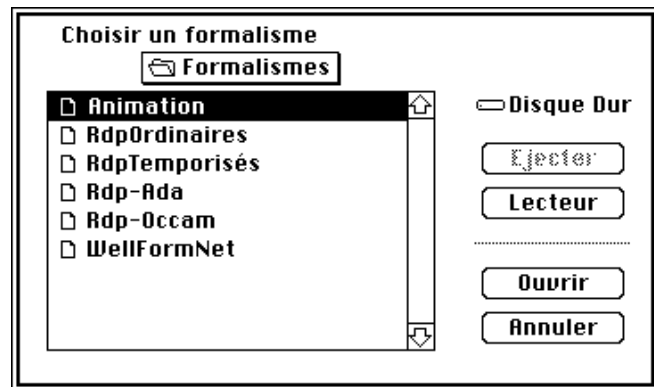


La **fenêtre d'historique** permet à Macao d'informer l'utilisateur d'un certain nombre d'évènements (erreurs, étapes en cours,...). Elle est surtout utile lors de l'utilisation de Macao en mode connecté (§6.3 Connexion)

La **palette**, située initialement dans la partie supérieure droite de l'écran, contient l'ensemble des outils de base de Macao. Son contenu est grisé, pour indiquer qu'aucune manipulation graphique n'est possible pour l'instant. En effet, aucun document n'est encore ouvert.

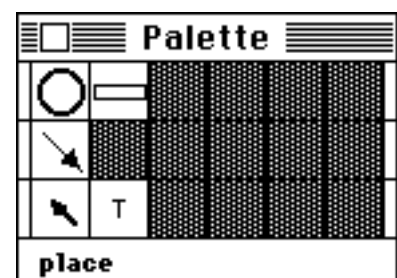


A partir du menu Fichier, il est possible de créer un nouveau document. Une zone de dialogue s'affiche. Ce type de cadre apparaît lorsque il faut fournir un complément d'informations afin que la commande choisie puisse s'exécuter. En l'occurrence, il faut choisir le **formalisme** de ce nouveau fichier.



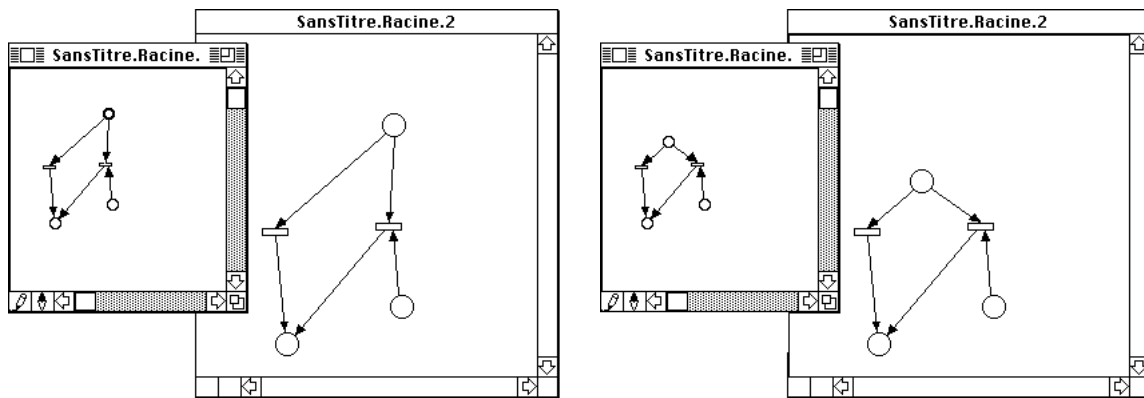
Le formalisme est actuellement contenu dans un fichier sur le poste de travail de l'utilisateur. Il contient à la fois une description du formalisme en CAMI-LDF et les modifications esthétiques du formalisme pour poste de travail (partie II §5.3.4 Modification esthétique d'un formalisme). Il faut remarquer que Macao associe utilisateur et poste de travail (notion d'ordinateur personnel). Nous n'avons pas implémenté la gestion des modifications esthétiques pour chaque utilisateur par le GUS-Serveur.

La palette est mise à jour en fonction des objets du formalisme. Cette palette contient l'ensemble des nœuds et des connecteurs du formalisme ainsi que des opérateurs de base comme l'opérateur de sélection et de manipulation de textes. Dans la figure ci-contre, la palette contient deux classes de nœuds (*place* et *transition*) et une seule classe de connecteur (*arc*). L'objet sélectionné est une *place* (indiqué en bas de la palette).

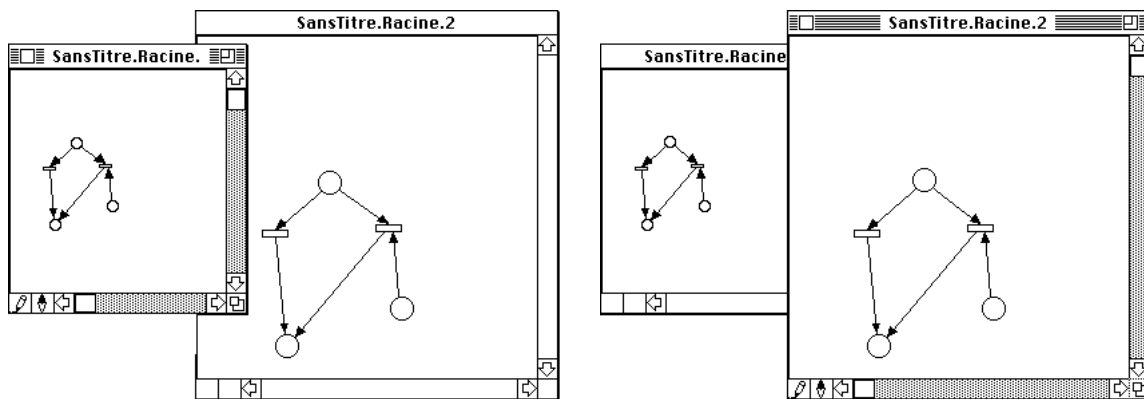


Une nouvelle fenêtre de nom "Sans titre" apparaît, c'est dans celle-ci que l'utilisateur va dessiner son énoncé. Pour le détail du dessin, voir dans le chapitre précédent (l'éditeur de graphes).

L'utilisateur peut changer l'échelle du dessin. Il peut aussi travailler sur le même document à deux échelles différentes ce qui lui permet d'avoir, à la fois une idée d'ensemble de son dessin et un détail du dessin.

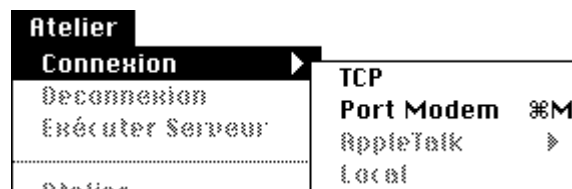


Le travail est ainsi possible à n'importe quelle échelle. L'utilisateur peut créer autant de fenêtres qu'il désire sur sa même feuille de travail. Toutes ces fenêtres ne sont pas connues de la structure d'accueil et les commutations de ces fenêtres n'entraînent pas de commutations de session.



7.3. Mode connecté de Macao

Deux cas de connexions sont prévus. Le premier intervient juste après l'ouverture de Macao et avant toute ouverture d'énoncé. Macao effectue uniquement une ouverture de communication puis lorsqu'un énoncé est ouvert, Macao ouvre une session. Dans le deuxième cas, l'utilisateur a ouvert un ou plusieurs énoncés puis se connecte à la structure d'accueil. Macao effectue l'ouverture de communication puis enchaîne les ouvertures de chacune des sessions. Nous décrivons le premier cas.



L'utilisateur ouvre Macao et demande la connexion dans le menu Atelier. La connexion correspond à l'ouverture du dialogue utilisateur-atelier. Cette **ouverture de communication** reconnaît l'utilisateur et met en communication les logiciels coté station de travail et coté structure d'accueil. Cette connexion a lieu soit par TCP/IP soit par ligne série (§5.4.2 Réalisation de la connexion physique).

Pour éviter l'utilisation de versions incohérentes de logiciels, Macao (GSV:couche session) envoie son numéro de version que la structure d'accueil vérifie afin d'autoriser la communication.

L'unicité de connexion d'un utilisateur (§6.2.1 Représentation des données personnelles) est effectuée par le test de présence du fichier '.Ami' qui se trouve dans le répertoire Ami de l'utilisateur et qui contient le numéro de port de connexion du communicateur. Si ce fichier n'existe pas, nous créons un processus communicateur et ce fichier, ensuite la connexion est réalisée entre la couche transport et la couche session. Le communicateur est ainsi le premier processus de la couche session chargé de gérer la multi-sessions. Il garde la liste des sessions en cours par utilisateur.

La structure d'accueil effectue une mise à jour du contexte dynamique de l'utilisateur pour indiquer qu'il a une communication en cours.

Après une ouverture de communication, l'utilisateur ouvre une session. Pour cela il choisit un formalisme compatible avec ceux connus de l'atelier. La structure d'accueil anticipe cette demande en envoyant la liste des formalismes que l'utilisateur peut choisir avec leur noms, numéros de version et messages d'aide.

Macao réceptionne la liste des formalismes utilisables et met à jour son GUS client. (voir la notion de GUS client, GUS serveur dans le chapitre §II.5.10 Architecture du GUS dans un environnement distribué)

7.4. Session - Présentation

Pour créer un nouvel énoncé, l'utilisateur choisit son formalisme comme dans l'utilisation en local. Pour cela, il faut que l'ensemble des fichiers de description formalismes sur le poste de travail se situent dans un répertoire.

Du fait qu'il y a une communication ouverte, l'ouverture de session est effectuée à la fin de lecture du fichier de description du formalisme. Cette ouverture n'est faite que si le formalisme choisi correspond à l'un des formalismes reçus précédemment de la structure d'accueil. En cas d'incohérence, ou d'inexistence sur la structure d'accueil, ou d'un formalisme pour lequel aucune application n'est actuellement disponible, l'utilisateur ne peut travailler qu'en mode autonome et la session n'est pas ouverte.

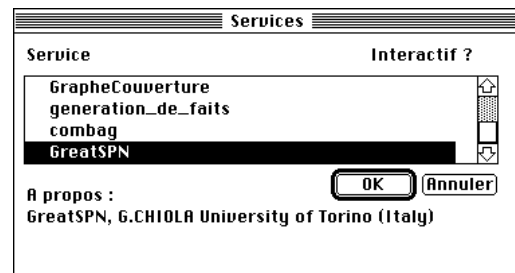
Remarque: dans la suite de ce paragraphe, le texte cadré sur la gauche correspond au travail de Macao et le texte cadré sur la droite correspond à celui de la structure d'accueil.

S'il y a correspondance au niveau des formalismes, Macao demande une ouverture de session en transmettant le nom du fichier, la date de dernière modification et le nom du formalisme utilisé: la session est alors ouverte.

L'ouverture de cette session permet à la structure d'accueil de créer d'une instance de couche présentation qui recherche la liste des services accessibles à l'utilisateur et l'envoie à Macao.


Le prototype AMI privilégie l'accès aux services par le nom de l'application., c'est à dire que l'utilisateur choisit une application. Dans la description de MARS et dans la version suivante d'AMI, l'utilisateur choisira directement des services.

Macao propose la liste des applications dans une fenêtre de dialogue et des informations d'aide. L'utilisateur choisit l'application désirée et ce choix est transmis à la structure d'accueil.



Le gérant des dialogues transmet la liste des questions que l'on peut poser à l'application, puis il demande d'afficher cette liste. En même temps, il lance l'application correspondante qui s'initialise.

La couche présentation de Macao réceptionne la liste des questions de l'application et affiche un nouveau menu correspondant à cette application. Ce menu apparaît grisé tant que l'application est en cours d'initialisation.

Atelier GreatSPN 

La phase d'initialisation de l'application consiste, en plus de phases d'initialisation classiques, à indiquer, pour chacune des questions possibles, si elles ont déjà été posées et si des résultats sont immédiatement disponibles. L'application peut aussi indiquer qu'une question est en cours d'exécution.

Macao réceptionne les mises à jour des questions et rend valide le menu de l'application. Cette fin d'initialisation correspond à une synchronisation forte utilisateur-application. A partir de là, l'utilisateur peut poser des questions à l'application.

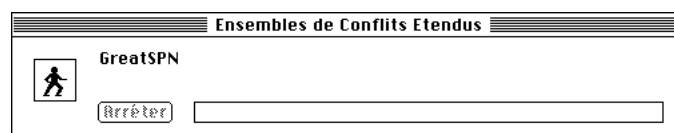


7.5. Application

Plusieurs sortes de questions peuvent être posées à une application. Des questions dites “texte terminal” dans lesquelles le simple nom de la question suffit à l'application pour effectuer son traitement. Des questions “texte” dans lesquelles un texte sert de paramètre à la question. Enfin des questions dites “objet terminal” dans lesquelles il faut sélectionner un ou plusieurs objets de l'énoncé qui seront les paramètres de la question. Toutes les questions peuvent être accompagnées d'options. Les options sont représentées dans Macao par des éléments de menu à cocher.

Quand l'utilisateur pose une question, la question et ses options sont transmises à la structure d'accueil. Sans attendre de confirmation de réception, Macao (partie interface utilisateur des services) affiche une **fenêtre d'état** indiquant à l'utilisateur que sa question a été prise en compte.

Remarque: si l'utilisateur n'invalide pas l'article “Résultat détaillé” du menu “Atelier”, les messages affichés dans la fenêtre d'état se retrouve aussi dans la fenêtre d'historique.



Dans cette fenêtre l'utilisateur obtient des informations concernant le déroulement de sa question. Il peut éventuellement arrêter la question avec le bouton “arrêter”.

Pour traiter une question, l'application a besoin de l'énoncé de l'utilisateur. C'est la structure d'accueil qui fournit cet énoncé car elle détermine, grâce à la date de dernière modification si l'énoncé de l'utilisateur a été modifié. S'il n'a pas été modifié, elle le transmet à l'application sans le demander à Macao car elle en a gardé un exemplaire. S'il y a eu modification, elle demande à Macao de lui transmettre l'énoncé. Macao transmet l'énoncé et affiche la progression de la transmission par une barre de défilement dans la fenêtre d'état ci-dessus.

Lorsqu'une question est en cours sur un énoncé, l'utilisateur ne doit pas modifier l'énoncé sinon le résultat n'aurait aucun sens. Plus précisément, il ne doit pas faire de modification syntaxique. Mais nous l'avons autorisé à faire des modifications esthétiques car l'application de calcul n'a pas connaissance de cette partie esthétique. Pour indiquer qu'il ne peut pas faire de modifications, l'icône en bas à gauche de sa fenêtre d'énoncé change (voir figure ci-contre).



Les résultats d'une question peuvent être de différentes formes:

La première et la plus simple de ces formes étant le texte. Les résultats textuels sont alors affichés dans la fenêtre d'historique. Lorsque des résultats sont affichés, un message dans la fenêtre d'historique indique le début de réponse à la question posée. La question posée est rappelée.

Une deuxième forme de résultats est la mise en évidence d'objets de l'énoncé associés ou non à des modifications d'attributs. Ces résultats sont vus du côté de l'application comme des ensembles de numéros d'objets. Pour Macao, chaque ensemble d'objets forme un groupe et Macao offre un moyen de les visualiser après la réception des résultats. Macao offre la possibilité de les visualiser, ensemble par ensemble, plusieurs ensembles à la fois. Il permet aussi de les cacher temporairement dans la fenêtre d'édition. Cette technique est utilisée pour la mise en évidence des flots dans les réseaux de Petri.

La troisième forme consiste à modifier des attributs de l'énoncé initial. Cette troisième forme est utilisée lors des simulations de réseaux de Petri ou le marquage des places évolue.

La quatrième forme de résultat consiste à modifier l'énoncé initial. Dans le cas d'une suppression, Macao procède de la même manière que si l'utilisateur effectuait la suppression. C'est à dire que le routage des arcs et les suppressions résultantes sont effectués. S'il faut créer de nouveaux objets du graphe énoncé l'application crée des objets en leur attribuant des numéros différents des numéros de l'énoncé initial. Macao dessine les objets mais comme l'application ne connaît pas l'esthétique, il se charge de positionner les objets automatiquement. Actuellement, les résultats sont corrects (esthétiquement) si les nœuds créés sont reliés à d'autres nœuds déjà existants dans l'énoncé initial. Le nouveau nœud est placé au barycentre des objets auxquels il est connecté.

Enfin la cinquième forme de résultat peut être un nouvel énoncé. Dans ce cas, Macao ouvre une fenêtre de résultat et les objets créés sont mis dans la nouvelle fenêtre. Ce résultats peuvent être dans un **autre formalisme**. Dans ce cas Macao recherche le fichier local de description esthétique du formalisme sur le disque de la station de travail.

Dans tous les cas de résultats de questions, l'utilisateur peut indiquer s'il veut des résultats détaillés. Cette demande est entièrement traitée localement. Dans ce cas, Macao affiche dans la fenêtre d'historique le détail de toutes les opérations effectuées: création d'objets, destructions, messages d'état des questions. Ces messages sont clairs dans la mesure où, par exemple, l'ordre de "suppression de l'objet n°45" sera traduit par "suppression place p3". Cela est d'une très grande utilité aussi bien pour les concepteurs de services lors de la mise au point, que pour le modélisateur qui obtient une synthèse textuelle des résultats du service.

En fin de question, la fenêtre d'état disparaît et l'utilisateur peut à nouveau modifier son énoncé.

7.6. Multi-sessions

L'atelier AMI prend en compte la notion de multi-sessions (Objectif 5), c'est à dire qu'à un instant donné, l'utilisateur peut ouvrir plusieurs énoncés ou peut travailler simultanément avec plusieurs applications.

L'utilisateur ouvre donc un fichier qu'il a créé précédemment. Macao propose à l'usager la liste de tous les fichiers disponibles sur son poste de travail. Ces fichiers sont constitués de commandes CAMI-LDF et LDE que Macao interprète comme si elles venaient de la structure d'accueil. La centralisation de réception de toutes les commandes CAMI permet aussi de simuler l'activité de communication avec la structure d'accueil par fichier. Un énoncé peut de manière inverse être transmis depuis la structure d'accueil simplifiant ainsi l'échange de modèles ou de parties de modèles.

Pour des raisons d'efficacité, un traitement spécifique est effectué lors de l'ouverture d'un énoncé par la couche de présentation. Ce traitement consiste à attendre la fin de lecture pour afficher le graphe et recalculer le routage des arcs (§5.6.1 L'éditeur de graphes). En effet, si ces calculs étaient faits au fur et à mesure, le calcul des points de connexion serait refait autant de fois qu'il arrive de connecteurs sur ce nœud. Nous avons constaté, un facteur de l'ordre de 20 gagné en temps de lecture.

A la fin de lecture du fichier, Macao suspend la session en cours et effectue une ouverture de session pour ce nouvel énoncé. Le travail de l'utilisateur est alors exactement le même que lorsqu'il n'y avait qu'une seule session.

Si l'utilisateur sélectionne une fenêtre n'appartenant pas à la session en cours, Macao va demander une suspension de la session et une reprise de la session correspondant à la fenêtre sélectionnée.

Ce problème survient durant une commutation; il est la conséquence de la nature multi-sessions des applications. En effet, effectuée à la demande de l'utilisateur, la commutation est instantanée pour Macao mais pas du côté distant en raison du délai de propagation et du temps de prise en compte (remontée des couches inter-processus et inter-machines) (§6.2.3 Répartition du fenêtrage du GSV en systèmes local et distant). Un problème analogue survient lors de la clôture avec poursuite des applications en arrière plan.

La prise en compte, au niveau de l'entité distante n'étant pas instantanée, il se peut que l'application ait déjà envoyé des informations qui deviennent incohérentes vis à vis de la session en cours à l'écran. Ce problème est un **problème de terminaison** dans un environnement tolérant aux pannes. Nous l'avons résolu en permettant à Macao de maintenir deux sortes de sessions: la **session courante** et la **session actuelle**. La session courante est la session en cours distante tandis que la session actuelle est la session active locale.

Dans figure 7.6, au temps t_1 , l'utilisateur clique sur une fenêtre de l'application B. Le changement de fenêtre est immédiat et la session actuelle devient B. Des messages de suspension de la session A et reprise de la session B sont transmis à la structure d'accueil. Ils sont reçus au temps t_2 et il y a commutation de session au niveau de la structure d'accueil. Un message d'accusé de commutation est transmis et Macao le reçoit au temps t_3 . La session courante devient B.

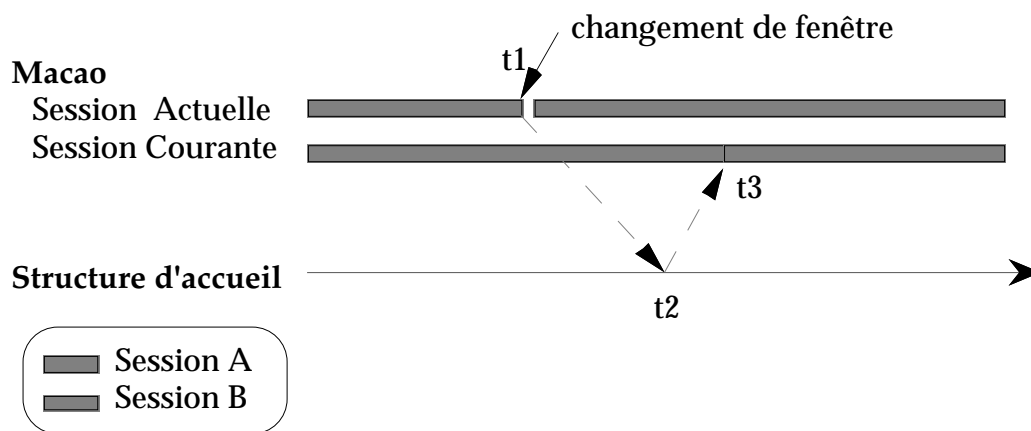


Figure 7.6: Commutation de session

Une autre solution aurait été d'estampiller chacun des messages par le nom de la session mais cela n'était pas nécessaire dans la mesure où, actuellement, les sessions ne se recouvrent pas dans le temps. Si on envisageait un multiplexage de sessions en parallèle, l'estampillage serait nécessaire au prix d'une augmentation de la taille de chaque message. Un tel estampillage devrait se faire avec le nom de la session et pas avec un numéro de session car il pourrait y avoir recouvrement des numéros en cas de déconnexion puis reconnexion.

En temps normal, les sessions courante et actuelle sont les mêmes et il n'y a qu'au moment de la commutation qu'elles sont temporairement différentes. Quand la demande de commutation est traitée par le GSV distant, il envoie un accusé de réception alors le GSV de Macao sait qu'il n'y aura plus de messages en provenance de la session précédente.

7.7. Déconnexion

L'utilisateur peut à tout moment arrêter Macao. Il faut donc effectuer une fermeture de chacune des sessions ouvertes. Un principe identique à la commutation de session est adopté pour obtenir un accusé de réception de terminaison.

Si une question était en cours sur un énoncé, Macao demande à l'utilisateur s'il désire arrêter la session distante ou s'il veut qu'elle se poursuive sans interface utilisateur. L'utilisateur pourra par la suite se reconnecter pour obtenir les résultats.

Lorsqu'une session se poursuit sans interface utilisateur, tous les résultats sont stockés par la structure d'accueil et les résultats déjà transmis peuvent donc être ignorés. Macao ignore donc ces résultats, en particulier, lorsque les résultats sont un nouvel énoncé, ils ne sont pas sauvegardés.

Du côté de la structure d'accueil, la couche transport se termine mais le communicateur reste actif tant que toutes les sessions en cours ne sont pas terminées. Ce communicateur garde le contexte dynamique de l'utilisateur.

7.8. Reprise de connexion

Des services demandés par l'utilisateur peuvent poursuivre leur exécution après déconnexion.

Pour se reconnecter, l'utilisateur ouvre un fichier déjà connu de la structure d'accueil à nouveau la session. Comme nous l'avons vu dans le paragraphe §6.3.1 La couche transport, le fichier ".ami" permet de savoir où se trouve le communicateur. Une enquête est faite auprès du communicateur pour savoir si la session en cours s'est bien terminée ou si elle est toujours en cours. L'utilisateur va être informé de l'état de sa session.

S'il choisit une session terminée ou en cours, une icône indiquant que le résultat est calculé ou en cours apparaît à côté de chacune des questions. Lorsqu'il choisit une de ces questions, le résultat arrive immédiatement. Pour cela, l'application, lors de son initialisation, met à jour l'état de chacune de ses questions. Ensuite, lorsque l'utilisateur pose une question, l'application envoie le fichier de résultats.

8. Conclusion

L'atelier AMI réalise effectivement l'ensemble de l'architecture que nous avons définie dans la partie II (Analyse Fonctionnelle). La figure 8.1 résume l'architecture de la version 1.1 de l'atelier AMI. L'application d'interaction utilisateur MACAO est réalisée sur Macintosh sous MacOS, tandis que la structure d'accueil fonctionne sur un réseau de machine UNIX.

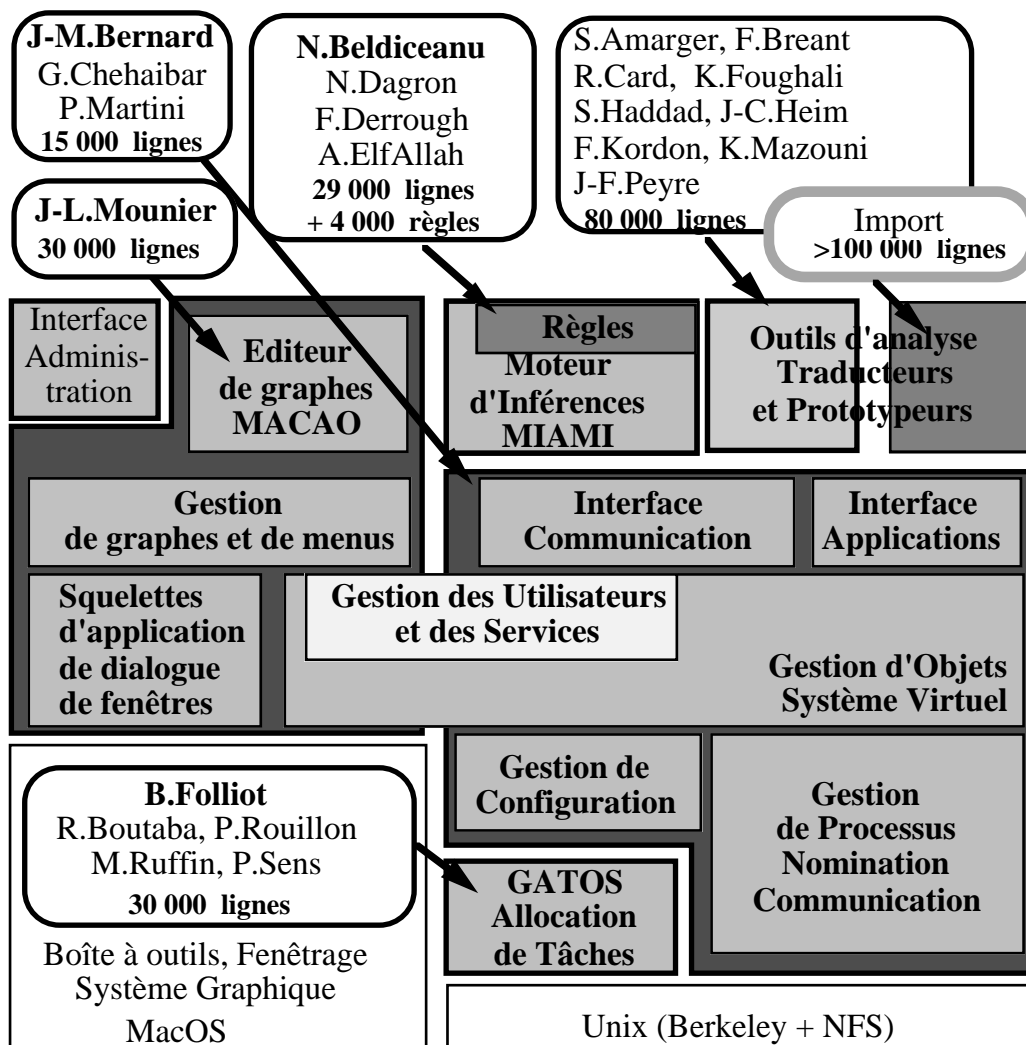


Figure 8.1: Réalisation de AMI

Pour montrer l'ampleur du travail réalisé et les collaborations impliquées, il nous fallait trouver une métrique pour cette réalisation logicielle. Nous avons donc indiqué sur la figure le nombre de lignes de différents codes source, complété par les noms des personnes (doctorants, stagiaires ou chercheurs extérieurs) ayant contribué à la réalisation de notre atelier ou des outils qui lui ont été intégrés.

Une propriété essentielle de la plate forme de l'atelier AMI est de savoir supporter de multiples formalismes. La figure 8.2 schématise l'existence de deux niveaux d'ouverture d'une part pour l'ajout de nouveaux formalismes et d'autre part pour l'ajout d'applications concernant un formalisme donné. Cette ouverture à de nouveaux formalismes et la facilité d'intégration d'applications basées sur les formalismes ainsi décrits garantissent les potentialités d'évolutions futures d'AMI vers de nouvelles théories.

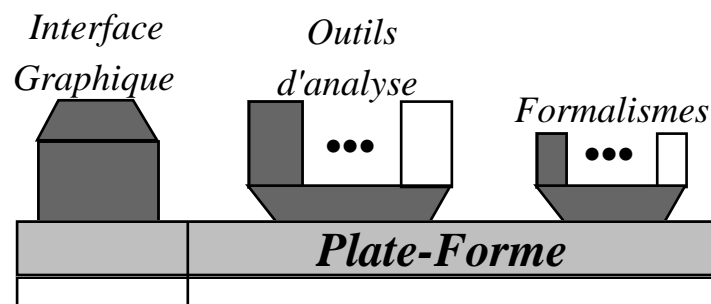


Figure 8.2: Possibilités d'extension de AMI

L'interface utilisateur de l'atelier est homogène et indépendante des programmes d'applications. Cette interface de Modélisation, Analyse et Conception Assistée par Ordinateur (MACAO) permet l'édition des énoncés, la mise en oeuvre des applications et la représentation de leurs résultats. Cette interface unique multi-formalismes et multi-applications facilite considérablement l'apprentissage et l'utilisation de l'atelier. La structure d'accueil peut ainsi exécuter, avec cette interface homogène, des applications compilées ou interprétées ou appliquer les règles d'un système expert.

Au point de vue support système AMI assure une gestion globale des données des utilisateurs et intègre la notion d'environnement multi-utilisateurs. L'atelier offre aux modélisateurs une puissance de calcul importante en assurant le parallélisme des services par l'utilisation du principe de multi-sessions et en supportant un ensemble de machines UNIX en réseau. Cet environnement distribué est rendu transparent aux modélisateurs et aux concepteurs d'outils. Par l'utilisation du système GATOS [B.Folliot 90], l'atelier assure la distribution des applications sur l'ensemble des machines du site avec des algorithmes de répartition de charge.

La répartition hétérogène des services sur des machines UNIX de l'interface sur toute la gamme des Macintoshs permet une utilisation particulièrement souple aussi bien isolée, que connectée par modem au réseau téléphonique, ou que connecté par divers réseaux informatiques (Figure 8.3). Ces possibilités se sont avérées efficaces au cours de nombreuses démonstrations dans des congrès.

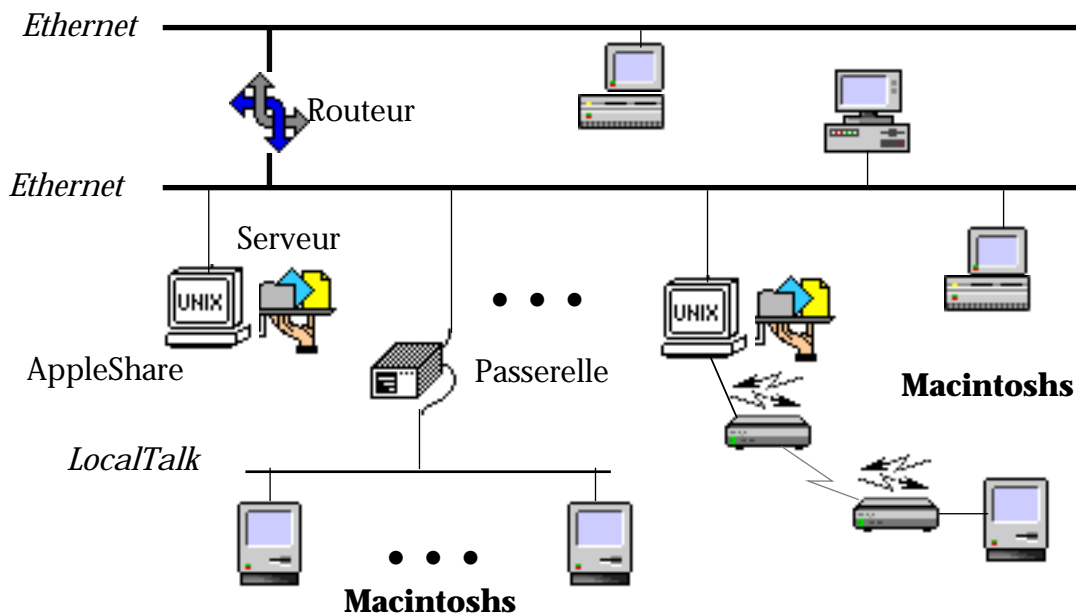


Figure 8.3: Possibilités d'exploitation de AMI

L'atelier AMI intègre l'ensemble de l'architecture fonctionnelle d'un atelier de spécification, il est une base opérationnelle pour le développement d'applications d'analyse et de prototypage. L'utilisation de cet atelier chez des industriels et universitaires, a montré ses capacités d'extensibilité et d'ouverture. Son achèvement et sa fiabilité lui permettent de servir de support à de nombreuses autres travaux de recherche.

Conclusion

Dans cette thèse, nous avons présenté les travaux de recherche relatifs à la conception et à la mise en œuvre de l'environnement système pour le projet MARS. Ce projet a pour but de concevoir un atelier intégré de Modélisation d'Analyse et de Réalisation de Systèmes afin d'accélérer la réalisation et l'intégration des logiciels.

Les domaines d'application du projet MARS sont les systèmes distribués, les protocoles de communication, la productique et le temps réel. Ces domaines d'application ont la particularité d'exprimer le parallélisme et l'indéterminisme des événements, ils sont combinatoires et donc difficilement appréhendables sans environnement logiciel. Ils impliquent des analyses à la fois qualitatives et quantitatives, l'étude de propriétés temporelles et la réalisation de prototypes pour des architectures dédiées.

Le projet MARS consiste donc à fournir un environnement muni d'outils coopérants pour assister l'ingénieur durant les étapes successives de spécification de systèmes, de validation de spécifications et de génération de prototypes. Il vise le travail coopératif par des équipes d'experts et d'ingénieurs travaillant dans le cadre d'un réseau, nécessairement hétérogène, pour profiter des évolutions des matériels et logiciels.

Nous avons donc défini l'architecture hiérarchique d'un atelier spécification de systèmes en la munissant a priori de possibilités d'ouverture puissantes afin de faciliter ses extensions. Cette architecture a été définie en analysant les besoins des utilisateurs aussi bien universitaires qu'industriels impliqués dans la conception de grands systèmes.

L'atelier repose sur une gestion multi-utilisateurs et multi-sessions dans un environnement distribué hétérogène. Un des principes fondamentaux de notre atelier est la séparation de l'interface utilisateur et des multiples outils d'analyse par l'utilisation d'une structure d'accueil. Contrairement à une approche commune où les interfaces utilisateur sont laissées à la charge des applications qui peuvent ainsi les particulariser, notre architecture localise la gestion du graphisme au niveau d'une application unique. Nous avons consacré un effort important de conception et de réalisation à cette interface graphique afin qu'elle soit de très grande qualité car elle joue un rôle crucial pour l'efficacité des ingénieurs. Cette approche nous a permis de réaliser une interface utilisateur puissante, de grande qualité et non spécialisée. L'unicité de l'interface utilisateur Macao assure pour les modélisateurs, une utilisation simple et homogène de l'ensemble des outils gérés dans l'atelier. L'avantage majeur est que l'interface graphique n'a plus à être connue des différents outils et dès lors, il est possible d'ajouter facilement de nouveaux outils.

Cette séparation de l'interface a exigé en contre partie de soigner particulièrement les communications avec les différents outils de l'atelier. Notre exigence, nous a amenés à définir deux langages originaux: d'une part un langage de description de menus (CAMI-LDQ), ceux-ci étant téléchargeables et gérés uniquement par l'interface utilisateur, et d'autre part un langage de description de formalismes et d'énoncés CAMI-LDF). Dans une telle architecture, la gestion des fenêtres graphiques pour l'édition des énoncés et des résultats reste exclusivement à la charge de l'interface utilisateur sans communication ni avec la structure d'accueil, ni avec les outils d'analyses.

La structure ouverte de l'atelier permet non seulement l'intégration d'applications mais la définition de nouveaux formalismes. Un méta-modèle basé sur les graphes typés à attributs simplifie la gestion du multi-formalismes en permettant de décrire les formalismes de manière externe aux applications. Notre langage CAMI-LDF de description des formalismes basé sur notre méta-modèle qui intègre la notion de représentation graphique du formalisme utilisable par l'interface utilisateur. L'utilisation d'une structure d'accueil et de différents langages de communication de données (formalismes, énoncés, menus et résultats) intégrés dans le langage CAMI permettent aux concepteurs d'outils de s'affranchir de la gestion graphique, non seulement de leurs énoncés, mais aussi de toutes leurs données. Une des conséquences de la souplesse de l'interface utilisateur et du multi-formalismes est la variété des modèles graphiques déjà utilisables dans l'atelier.

La prise en compte de la gestion du multi-formalismes dans notre atelier implique une gestion des données capable d'associer à chaque formalisme l'ensemble des services et applications accessibles. Nous avons défini la notion d'appel de service qui permet à une application de demander l'exécution d'un service dans un formalisme sur des données dans un environnement distribué. Cette notion nécessite aussi la gestion globale des données des modélisateurs dans l'environnement distribué. La structure d'accueil assure en particulier les liens énoncés-services-résultats et facilite les opérations de traduction de formalismes nécessaires pour passer de spécifications à des modèles ou à des prototypes. La neutralité de l'atelier vis à vis des formalismes permet d'ajouter dynamiquement de nouveaux formalismes avec appel des outils correspondants afin d'enrichir les connaissances de l'atelier accessibles à l'ensemble des utilisateurs.

La structure d'accueil s'exécute dans un l'environnement distribué hétérogène et offre ainsi une puissance de calcul nécessaire aux outils d'analyse. La structure d'accueil assure les communications entre un réseau de Macintosh et un réseau de machines hétérogènes sous UNIX. afin d'exécuter les applications choisies. La structure du réseau sous-jacent, l'hétérogénéité des matériels et des logiciels associés, la répartition de charge sont rendus transparents à l'interface utilisateur et aux concepteurs d'applications. L'architecture de l'atelier, ainsi obtenue, facilite les communications dans un environnement distribué entre les différents outils en fournissant un langage de communication inter-services rendant l'atelier matériellement extensible.

Nous avons réalisé un prototype d'atelier de spécification (AMI) comportant une structure d'accueil sous Unix, une application d'interface utilisateur MACAO et intégrant une quinzaine d'outils d'analyse.

L'application MACAO (Modélisation, Analyse et Conception Assistée par Ordinateur) s'exécute sur Macintosh. Elle ne se limite pas à la simple édition graphique de modèles, mais tient aussi à la qualité de visualisation des résultats. Sa standardisation, par rapport aux outils actuels ou futurs, facilite l'utilisation de différents formalismes et de différents outils d'analyse.

Nous avons éprouvé les possibilités d'ouverture de l'atelier AMI par l'intégration de nombreux outils d'analyse provenant des recherches de notre laboratoire et de celles de plusieurs autres laboratoires. Nous avons entre autres intégré, toujours avec la même interface graphique, le très important ensemble de logiciels GSPN (Great Stochastic Petri Nets) développé par l'équipe de G.Chiola à Turin. Dans la version actuelle de l'atelier, nous permettons d'introduire des applications programmées sous forme soit impérative, soit applicative, soit logique. En particulier nous offrons un outil orienté vers la conception de services par l'utilisation du système expert MIAMI (Moteur d'Inférences pour AMI, muni de nombreux paquets de règles) qui permet au concepteur d'écrire rapidement des applications sous forme de règles. Les différentes intégrations ont montré que notre objectif d'ouverture de l'atelier a été pleinement atteint.

L'étape d'intégration est facilitée par la notion de pilote d'application qui définit l'interface entre la structure d'accueil et les applications. Nous avons déduit de ces nombreuses intégrations des méthodes d'ajouts d'applications anciennes et de conception de nouvelles applications. La mise en place de l'atelier dans des laboratoires universitaires et chez des industriels nous a montré la nécessité de fournir des outils d'administration et d'installation dans un environnement distribué.

Nous voulons maintenant passer à une étape d'utilisation intensive dans le monde de l'industrie par l'ajout de nouveaux formalismes et de nouveaux outils. Pour des systèmes complexes, les besoins des utilisateurs universitaires et industriels ont évolué vers une modélisation hiérarchique et l'utilisation de bibliothèques de modèles. C'est pourquoi nous envisageons d'enrichir notre description des formalismes pour y ajouter la notion de hiérarchie et augmenter sa partie sémantique. De plus, pour améliorer la présentation des résultats des outils d'analyse, nous avons commencé, en concertation avec d'autres laboratoires, à normaliser les types usuels de modèles et de résultats. L'atelier prend déjà en compte le formalisme des Réseaux de Petri Bien Formés (Well Formed Petri Nets) avec au fur et à mesure les outils d'analyse et d'évaluation de performances qui sont en cours de réalisation. au laboratoire.

L'axe essentiel de nos travaux est l'extension d'AMI pour une conception collective de systèmes impliquant une forte coopération entre les utilisateurs d'un même projet. Nous envisageons en particulier le couplage de notre atelier de spécification avec un atelier de génie logiciel spécialisé pour assurer la gestion des projets, des versions, des programmes sources des prototypes générés et le partage de bases de connaissances. La version actuelle de l'atelier AMI constitue déjà une très bonne base de départ opérationnelle, pour concevoir la prochaine version de l'atelier intégrant les notions de formalismes hiérarchiques, de conception multi-utilisateurs, de tolérance aux pannes, d'administration des différents type d'utilisateurs (modélisateurs, concepteurs d'outils, administrateurs de formalismes et de bases de modèles).

Bibliographie

- Amarger S:** "Calcul du Graphe d'Etat au Plus Tôt d'un Réseau de Petri T-Temporisé: Intégration à l'atelier AMI", technical report (DEA) 1988.
- APPLE:** "*Inside Macintosh*", Ed. A. C. Inc., Addison Wesley, Reading, Mass., 1985.
- APPLE:** "*Human Interface Guidelines: the Apple® Desktop Interface*", Ed. A. C. Inc., Addison Wesley, Waltham, Mass., 1987.
- Arora A, Chan D, Ferrans J, Gordon R:** "*An overview of the visual software development environment*", Proceedings of the Computer Software & Applications Conference, the IEEE Computer Society's Ninth International, Chicago, 1985.
- Arthaud R, Roan A:** "*AGE: Maitriser la construction des systèmes temps réels*", Proceedings of the Second International Workshop on software engineering and its applications, Toulouse, France, 4-8 décembre 1989.
- Aubry C, Bougro H:** "*Implémentation industrielle et utilisation du langage LDS à TRT*", Journées du FIRTECH Systèmes et télématique, CNET Issy-les-Moulineaux, pp 61-76, 29-31 Janvier 1990.
- Bach MJ:** "*The design of the Unix® Operating system*", Ed. Prentice Hall, 1986.
- Barth PS:** "*An Object Oriented Approach to Graphical Interfaces*", ACM Transactions on Graphics, vol 5 (2), April 1986.
- Beaudouin-Lafon M:** "*User interface Support for the Intregation of software Tools: an Iconic Model of Interaction*", ACM SIGSOFT/SIGPLAN Software Engineering Symposium on Practical Software Development, vol 24 no 2, pp 143-152, 28-30 November 1988.
- Beaudouin-Lafon M:** "*Petripote, a graphic system for Petri net design and simulation*", Proceedings of the 4th european workshop on applications and theory of Petri nets, Toulouse France, pp 20-30, 1983.
- Behm P:** "*RAFAEL: A tool for analysing parallel systems in the L environment*", Proceedings of the Sixth european workshop on applications and theory of Petri nets, Espoo - Finland, 26-28 June 1985.
- Beldiceanu N:** "*Un langage de règles de production basé sur des contraintes et des actions*", 7^{èmes} journées internationales sur les systèmes experts et leurs applications, Avignon, 1987.

- Beldiceanu N:** "*Langage de règles et moteur d'inférence basés sur des contraintes et des actions. Application aux réseaux de Petri*", Thèse de l'université de Paris VI, 1988a.
- Beldiceanu N, Souissi Y:** "*Deterministic systems of sequential processes: theory and tools*", Proceedings of the International Conference on Super Computers, Hamburg, RFA, October 1988b.
- Belkhatir N, Estublier J:** "*Coopération et protection dans un environnement de génie logiciel*", troisième Colloque de Génie logiciel AFCET, Versailles, 1986.
- Benkiram MA:** "*Communication dans les systèmes distribués*", Thèse de doctorat d'état, Université Pierre et Marie Curie Paris VI, 1988.
- Benzekri A, Marquie D, Raynaud Y:** "*Administration de systèmes de communication en milieu hétérogène*", Séminaire Franco-Brésilien sur les systèmes informatiques répartis, Florianopolis, Brésil, pp 172-179, 14 Septembre 1989.
- Berger JL, Picciolto J, Woodward JPL, Cummings PT:** "*Compartmented Mode Workstation: Prototype Highlights*", IEEE Transaction on software Engineering, vol 16 nb11, pp 608-618, June 1990.
- Bernard J-M, Guilmet M, Smirnow C:** "*ARP : Analyse de Réseaux Paramétrés*", Technical report (DEA) 1985.
- Bernard J-M, Mounier J-L:** "*L'atelier logiciel AMI, un environnement multi-utilisateurs, multi-sessions pour une architecture distribuée*", Séminaire Franco-Brésilien sur les systèmes informatiques répartis, Florianopolis, Brésil, pp 200-206, 14 Septembre 1989.
- Bernard J-M, Mounier J-L, Beldiceanu N, Haddad S:** "*AMI: An extensible Petri net Interactive Workshop*", Proceedings of the 9th European Workshop on Application and Theory of Petri nets, Venice, Italy, pp 101-117, June 1988.
- Bernard J-M, Mounier J-L, Martini P, El Fallah A, Derrough F, Allain P, Amarger S, Madiou M:** "*Intégration de MIAMI dans l'atelier AMI*", technical report 1989.
- Bernas P:** "*L'environnement PEGASE: Aspect pratiques et fondements théoriques*", Proceedings of the Second International Workshop on software engineering and its applications, Toulouse, France, 4-8 décembre 1989.
- Berthelot G, Johnen C, Petrucci L:** "*PAPETRI: un Poste d'Analyse des réseaux de PETRI*", Journées du FIRTECH Systèmes et télématique, CNET Issy-les-Moulineaux, pp 109-126, 29-31 Janvier 1990.
- Boehm BW:** "*Software engineering economics*", Ed. R. T.Yeh, Prentice Hall, Englewood Cliffs, New Jersey 07632, 1981.
- Booch G:** "*Ingénierie du logiciel avec ADA*", Ed. interEditions, 1988.

- Bourgeois J, Gindre C, Ryckebusch P:** "*Une architecture d'atelier de génie logiciel réparti*", Revue Génie logiciel et Système Expert, vol 13, pp 12-15, Décembre 1988.
- Brams GW:** "*Réseaux de Petri, théorie et pratique*", Ed. Masson, Paris, 1983.
- Bréant F:** "*Tapioca: OCCAM Rapid Prototyping from Petri Nets*", Proceedings of the Vth Jerusalem Conference on Information Technology, Jerusalem, Israel, October 1990.
- Brettnacher J-M, Chabrier J-J, Champagne R, Derniame J-C, Jamard P, Legait A, Oldfield D, Psonis S, Sanchez J, Weber H:** "*Accueil Logiciel Futur (ALF) Overview of the Project (1520)*", ESPRIT'88 Putting the Technology to Use, North Holland, vol 1, pp 508-518, 1988.
- Broussard J-C, Michel C, Partouche J-P, Rousseau R, Rueher M:** "*Une évaluation du langage Eiffel*", Techniques et Science Informatique, vol 9-4, pp 331-373, 1990.
- Budkowski S, Dembinski P:** "*An Introduction to Estelle: A Specification Language for Distributed Systems*", Computer Networks and ISDN systems, vol 14, pp 3-23, 1988.
- Bustard DW, Harmer TJ:** "*An implementation of PEEP : A software animation tool for concurrent systems*", Proceedings of the Second International Workshop on software engineering and its applications, Toulouse, France, 4-8 décembre 1989.
- Bütler B, Esser R, Mattmann R:** "*A distributed Simulator for High Order Petri Nets*", Advances in Petri Nets 90, 1990.
- CAIS:** "*CAIS: Common APSE Interface Set*", n°DOD-STD-1838, 1986.
- Camacho J, Langlois C, Paul E:** "*ELVIS, an Integrated Environment*", 4th SLD Forum, 1989.
- Campbell I, Boudier G, Brémeau C, Grosselin J-C:** "*The PCTE proposed standard tool support interface*", Proceedings of the US Federal CASE Conference 1989 on Integrated Data Management for Software Engineering, Gaithersburg, Maryland, USA, november 1989.
- Caseau Y:** "*Etude et Réalisation d'un Langage Objet: SPOKE*", Thèse de l'Université Paris-Sud, Orsay, Novembre 1987.
- Chiola G:** "*A Software Package for the Analysis of Generalized Stochastic Petri Net Models*", Proceedings of the International Workshop on Timed Petri Nets, Torino, Italy, pp 136-143, July 1985.
- Chiola G:** "*GreatSPN User's Manual*", Rapport technique Université de TURIN, 1986.
- Chiola G:** "*A Graphical Petri Net Tool for Performance Analysis*", Proceedings of the 3rd International workshop on modelling techniques and performance evaluation, PARIS, pp 297-307, March 1987.

- Chiola G, Franceschinis G:** "*Colored GSPN Models and Automatic Symmetry Detection*", Proceedings of the 3rd International Workshop on Petri Nets and Performance Models, Kyoto, Japan, pp 50-60, 1989.
- Collet P:** "*Automatic Aesthetic lay-out of nodes in a graph*", technical report 1990.
- Collongues A, Hugues J, Laroche B:** "*Merise*", Ed. Dunod, 1987.
- Colom JM, Martinez J, Silva M:** "*Packages for validating discrete production systems modelled with Petri nets*", Proceedings of the IMACS-IFAC Symposium, 1986.
- Comer D:** "*Internet working with TCP/IP: Principles, protocols and architecture*", Ed. E. cliffs, Prentice Hall, 1988.
- Commando:** "*Creating a Commando Interface for Tools*", Macintosh Programming Workshop 3.0 Apple Computer, Inc (APDA), 1989.
- Coulouris G-F, Dollimore J:** "*Distributed systems concepts and design*", Ed. Addison Wesley, 1989.
- Courtiait J-P:** "*SEDOS results: exploitation & new perspectives for estelle & lotos*", Proceedings of the Second International Workshop on software engineering and its applications, Toulouse, France, pp 111-116, 4-8 December 1989.
- Cousin B, Estrailier P:** "*PLASMA : A multi-medium protocol for the LINK Layer*", Proceedings of the 12th IMACS World Congress 88 on scientific Computation, Paris, July 1988.
- Coutaz J:** "*Abstractions for User Interface Design*", IEEE, IEEE Computer, vol 18, pp 21-34, September 1985.
- Coutaz J:** "*Architectures et outils pour la programmation des systèmes interactifs*", Bulletin de liaison de la recherche en informatique et en automatique, BL-INRIA, INRIA BP 105 78153 Le Chesnay Cedex, vol 127, pp 2-11, 1990.
- Couvreur J-M, Haddad S, Peyre J-F:** "*Résolution Paramétrée d'une famille de systèmes d'équations linéaires a solutions positives*", Rapport de recherche IBP-MASI, n°90.38, 1990.
- CXP:** "*Les ateliers de génie logiciel sur grands systèmes*", CXP, 1987.
- CXP:** "*Les Ateliers de Génie Logiciel: Analyse et Comparaison*", CXP-125, 1989.
- Dagron N:** "*Spécification textuelle des réseaux RPAS*", 1989.
- Dagron N, Bonnaire X:** "*Syntaxe du langage LPAS*", Rapport de contrat industriel MASI (SLIGOS), n°R5/V2, 1990.
- Dagron N, Derrough F, El Fallah A:** "*Ingénierie à l'aide du système expert MIAMI*", 1^{ère} Conférence internationale Maghrébine d'Intelligence Artificielle et de Génie Logiciel, Constantine, Algérie, Septembre 1989a.

- Dagron N, Estrailier P, Haddad S:** "*Syntaxe et sémantique des réseaux RPAS*", Rapport hors contrat industriel MASI, n°R0+/V1, 1989b.
- Dagron N, Mounier J-L:** "*Correspondance entre un réseau RPAS et le réseau de Petri engendré par traduction*", 1989c.
- De Postel C, Massié H, A. M:** "*OASIS: Un atelier de gestion des objets pour les applications évolutives*", Proceedings of the Second International Workshop on software engineering and its applications, Toulouse, France, 4-8 décembre 1989.
- Deguine O:** "*Conception et spécification de l'interface homme-machine d'un poste de travail de génie logiciel*", Thèse de l'université de Paris VI, 1988.
- Deuin Mea:** "*Aïda: environnement de developpement d'application*", ILOG Paris, 1987.
- Diaz M:** "*Environnement logiciels pour la conception des protocoles dans les systèmes distribués*", Séminaire Franco-Brésilien sur les systèmes informatiques répartis, Florianopolis, Brésil, pp 18-27, 14 Septembre 1989.
- Diaz M, Visser C, Budkowski S:** "*Estelle and Lotos Software for the Design of Open distributed Systems (410)*", ESPRIT'87 Achievements and Impact, North Holland, vol 1, pp 543-558, 1987.
- Drouas E, Nerson J-M:** "*Les Ateliers logiciels intégrés : développements français actuels*", TSI, vol 1 n°3, 1982.
- Eager DL, Lazowska ED, Zahorjan J:** "*Adaptative Load Sharing in Homogeneous Distributed Systems*", IEEE Transaction on Software Engineering, IEEE, vol SE-12 n°5, pp 662-675, May 1986.
- El Fallah A:** "*Base de connaissances pour l'analyse et la validation des réseaux de Petri*", Proceedings of the 5ième congrès international d'intelligence artificielle et de robotique, Bratislava, Novembre 1989.
- Emrich J:** "*Remote File Systems, Streams, and Transport Level Interface*", UNIX Papers for UNIX Developers and Power Users, Indianapolis, pp 261-306, 1987.
- Endres R, Schneider M:** "*The GRASPIN Software Engineering Environment (125)*", ESPRIT'88 Putting the Technology to Use, North Holland, vol 1, pp 349-364, 1988.
- Estrailier P:** "*Conception de protocoles d'interconnexion robustes, application à la gestion d'un anneau virtuel*", doctorat de l'Université, Pierre et Marie Curie, 1986.
- Feldbrugge F:** "*Petri net tool overview 1989*", Advances in Petri nets 1989, vol 424, pp 151-178, 1989.
- Feldbrugge F, Fraisse P, Johnen C, Trèves N:** "*SERPE, an extensible structure for analysis of Petri nets*", Research Report LRI, Université d'Orsay, n°302, 1986.

- Feldbrugge F, Jensen K:** "*Petri net tool overview 1986*", Petri Nets: Applications and relationships to other models of concurrency, vol 255, pp 20-62, 1987.
- Finkel A:** "*Effective analysis of large Petri Nets with the minimal coverability graph*", Journées du FIRTECH Systèmes et télématique, CNET Issy-les-Moulineaux, pp 245-258, 29-31 Janvier 1990.
- Florin G, Natkin S:** "*RDPS: a software package for the validation and the evaluation of dependable computer systems*", Proceedings of the IFAC SAFECOMP workshop, Sarlat, France, October 1986.
- Foliot B, Ruffin M:** "*GATOS: Gérant de Tâches dans un système distribué*", Convention Unix 89, Paris, 1989.
- Foughali K, Masouni K:** "*Vers une méthode d'intégration d'applications dans l'atelier AML: mise en œuvre pour Combag et GreatSPN*", technical report (DEA) 1990.
- Galinier M:** "*L'assurance Qualité Logicielle: L'approche industrielle*", Revue Genie Logiciel : la qualité du logiciel, pp 12-29, Janvier 1988.
- Gardarin G:** "*Bases de données avancées*", Ed. Masson, 1989.
- Garlick L, Lyon R, Delzompo L, Gallagher B:** "*ONC: The Open Network Computing Environment*", Sun Technology, pp 42-46, Spring 1988.
- Giavitto J-L, Devarenne A, Rosuel G, Holvoet Y:** "*ADAGE: Utilisation de la généricité pour construire des environnements adaptables*", Proceedings of the Second International Workshop on software engineering and its applications, Toulouse, France, 4-8 décembre 1989.
- GKS:** "*Computer graphics - Graphical Kernel System (GKS)*", Functional Description ISO: Information Processing Systems, n°ISO 7942, 1985.
- Goldberg A:** "*Smalltalk-80: the interactive programming environment*", Ed. Addison Wesley, Reading, MA, 1984.
- Goodman D:** "*The Complete HyperCard Handbook*", Ed. Bantam Computer Books, Toronto, 1987.
- Gosling J:** "*Partitioning of function in window systems*", Eurographic Seminars, Methodology of window management, Abingdon UK, pp 101-106, April 1985.
- Guillemont M:** "*Etude comparative de quelques systèmes répartis*", TSI, pp 5-21, 1984.
- Haddad S, Bernard J-M:** "*ARP un outil de spécification et de validation des systèmes répartis*", troisième Colloque de Génie logiciel AFCET, Versailles, 1986.

- Hartson HR, Hix D:** "*Human Computer Interface Development: Concepts and Systems for its management*", ACM Computing Survey, vol 21 nb 1, pp 5-92, March 1989.
- Hayselden B:** "*Implementation of the PCTE user Interface on Sun Workstation (1277)*", ESPRIT'87 Achievements and Impact, North Holland, vol 1, pp 325-333, 1987.
- Heimbigner D, Krane S:** "*A graph Transformation Model for Configuration Management Environment*", ACM SIGSOFT/SIGPLAN Software Engineering Symposium on Practical Software Development, vol 24 no 2, pp 216-225, 28-30 November 1988.
- Hein J-C:** "*Intégration du logiciel ARP dans l'atelier AMI*", technical report (DEA) 1988.
- Helary J-M, Raynal M:** "*Vers une problématique de l'algorithmique répartie*", Séminaire Franco-Brésilien sur les systèmes informatiques répartis, Florianopolis, Brésil, pp 18-27, 14 Septembre 1989.
- Henderson-Sellers B, Edwards JM:** "*The Object oriented systems life cycle*", Communication of the ACM, vol 33 nb 9, pp 143-159, 1990.
- Hubert L, Perdreau G:** "*Software factory: using process modeling for integration purposes*", Proceedings of the Second International Workshop on software engineering and its applications, Toulouse, France, pp 237-248, 4-8 décembre 1989.
- Hughes AC, Puncello P, Pietri F:** "*Intelligent Systems Development in the ASPIS Environment (401)*", ESPRIT'88 Putting the Technology to Use, North Holland, vol 1, pp 380-391, 1988.
- Jaulent P:** "*Génie logiciel les méthodes*", Ed. A. Colin, 1990.
- Jensen K:** "*Computer tools for construction, modification and analysis of Petri nets*", Petri Nets: Applications and relationships to other models of concurrency, pp 4-19, 1987.
- Karsenty S:** "*Graffiti, un outil interactif et graphique pour la construction d'interfaces hommes-machine adaptables*", Thèse de 3ème cycle, LRI Orsay, 1987.
- Karsenty S:** "*La spécification d'interfaces*", Bulletin de liaison de la recherche en informatique et en automatique, BL-INRIA, INRIA BP 105 78153 Le Chesnay Cedex, vol 127, pp 12-17, 1990.
- Kordon F, Estrailier P, Card R:** "*Rapid ADA Prototyping: principles and example of a complex application*", Proceedings of the International Phoenix Conference on Computer and Communications, April 1990.
- Laforge P, Paire E:** "*Le réseau hétérogène multi-médias de l'IMAG*", Convention Unix 88, Paris, 1988.

- Laprie J-C:** "*Dependable computing and fault tolerance: basic concepts and terminology*", IFIP WG 10.4, Kissimee, Floride, Summer 1984.
- Leach P-Jaa:** "*The architecture of an integrated local network*", IEEE Journal on selected areas in communications, vol 1, pp 842-857, November 1983.
- Leffer SJ, McKusick MK, Karels MJ, Quarterman JS:** "*The Design and implementation of the 4.3BSD Unix® Operating System*", Ed. Addison Wesley, 1989.
- Legatheaux Martins J, berbers Y:** "*La désignation dans les systèmes d'exploitation répartis*", Techniques et Sciences de l'Ingénieur, TSI, vol 7 (4), pp 359-372, 1988.
- Lehman M, Turski WM:** "*Essential Properties of IPSEs*", Software Engineering Notes, vol 12 (1), pp 52-55, January 1987.
- Lieberman H:** "*There's More to Menu Systems the Meets the Screen*", Computer Graphics, SIGGRAPH'85, vol 19 (3), pp 181-189, 1985.
- Lyon B:** "*Sun eXternal Data Representation specification*", article Sun Sun Microsystems, Inc., 1984.
- Marca D, Gowan CM:** "*SADT Structured Analysis and Design Techniques*", Ed. Mc Graw Hill, 1987.
- Masai:** "*Masai 1.0, l'éditeur interactif d'interfaces graphiques.*", ILOG, 1989.
- Mc Call J:** "*Factors in software quality*", Ed. M. Call, general electric, 1977.
- Memmi G, Finkel A:** "*An introduction to FIFO-nets, monogeneous nets: a subclass of FIFO-nets*", Theoretical Computer Science, vol 35, pp 191-214, 1985.
- Meyer B:** "*Object-Oriented Software construction*", Ed. Prentice-Hall, 1988.
- Milner R:** "*CCS, a calculus for communication systems*", Lecture Notes in Computer Science, vol 92, 1980.
- Molich R, Nielson J:** "*Improving a Human Computer Dialogue*", Communication of the ACM, vol 33 nb 3, March 1990.
- Muenier M:** "*Atelier de Génie logiciel: place à l'ouverture*", Genie logiciel et systèmes experts, vol 17, décembre 1989.
- Murphy SC, Gunningberg P, Kelly JPJ:** "*Implementing Protocols with Multiple Specifications: Experiences with Estelle, LOTOS and SDL*", Proceedings of the Ninth IFIP WG 6.1 International Symposium on Protocol Specification, Testing, and Verification, Enschede, The Netherlands, 6-9 June 1989.
- Nelson BJ:** "*Remote Procedure Call*", PhD Thesis, Carnegie-Mellon, 1981.
- NeWS:** "*NeWS Preliminary Technical Overview*", Technical report 1986.

- NextStep:** "*Interface Builder Le générateur d'interface de la station Next*", Documentation technique NEXT Computer, Inc., 1989.
- NFS, Sandberg R, Goldberg D, Kleiman S, Walsh D, Lyon B:** "*Design and implementation of the Sun Network File System*", Proceedings of the USENIX Conference, Portland, pp 119-130, June 1985.
- NSE:** "*NSE network software environment*", Sun Microsystems Inc., 1990.
- Olsen D, Mike R:** "*the Menu Interaction Kontrol Environnement*", ACM Transactions on Graphics, pp 318-344, October 1986.
- OPEN LOOK:** "*OPEN LOOK™ Graphical User Interface Application Style Guidelines*", Addison-Wesley Publishing Compagny, Inc. Sun Microsystems, Inc. and AT&T, n°First printing, 1989a.
- OPEN LOOK:** "*OPEN LOOK™ Graphical User Interface Functional Specification Release 1.0.1*", Sun Microsystems, Inc. and AT&T, 1989b.
- Orange Book:** "*Trusted computer system evaluation criteria*", Ed. NCSC, 1985.
- OSF/Motif:** "*HP OSF/Motif Style Guide Edition 1*", Hewlett-Packard Compagny, 1000 N.E. Circle Blvd, Corvallis, OR 97330., 1989a.
- OSF/Motif:** "*OSF/Motif Programmer's Guide*", Open Software Foundation, 11 Cambridge Center, Cambridge, MA 02142, 1989b.
- PCTE:** "*Introduction to PCTE+, Independant European Programme Group*", PCTE Newsletter n°2, 1989.
- PHIGS:** "*Computer graphics - Programmer's Hierarchical Interface to Graphic (PHIGS)*", Functional Description ISO: Information Processing Systems, n°ISO DP 9592, 1986.
- PostScript:** "*PostScript language reference Manual*", Ed. Addison-Wesley, 1986.
- Presotto DL, Ritchie DM:** "*Interprocess Communication in the Ninth Edition Unix System*", Software Practice and Experience, vol 20(S1), June 1990.
- Prosser C:** "*Introducing Windows to Unix: User Expectations*", Eurographic Seminars, Methodology of window management, Abingdon, UK, pp 9-13, April 1985.
- Purtillo J:** "*On specifying an environment*", Proceedings of the the IEE Computer Society's 9 international Computer Software & applications conferences (COMPSAC 85), chicago,USA, 9 -11 october 1985.
- Ramamoorthy CV, Prakash A, Garg V, Yamaura T, Bhide A:** "*Issues in the development of large, Distributed, and Reliable Software*", Advances in computer, vol 26, pp 393-443, 1987.
- Rifflet J-M:** "*La programmation sous UNIX*", Ed. McGraw-Hill, 1986.

- Rifflet J-M:** "*La Communication sous UNIX*", Ed. McGraw-Hill, 1990.
- Rose MT:** "*The Open Book: A practical perspective on OST*", Ed. T. W. G. Inc., Prentice Hall, 1990.
- Roubine O:** "*Présentation d'AGL actuels*", Les ateliers de génie logiciel, Groupe des Utilisateurs de Sun, 6 Juin 1989.
- Rozier M, Legatheaux Martins J:** "*The CHORUS distributed operating system: some design issues*", Distributed Operating Systems, Theory and Practice, NATO ASI Series, vol F28, pp 261-287, 1987.
- Salmon R, Slater M:** "*Computer graphics: systems & concepts*", Ed. Addison Wesley Publishing Company, 1987.
- Sandberg R:** "*The Sun Network File System: Design, Implementation and Experience*", article Sun Sun Microsystems, Inc., 1987.
- Sanden B:** "*An Entity-Life modeling approach to the design of concurrent software*", Computer Practices, Communications of the ACM, vol 32 nb 3, pp 330-343, March 1989.
- Scapin DL:** "*Guide ergonomique de conception des interfaces Homme-Machine*", rapport technique INRIA, n°77, 1986.
- Schmucker K:** "*MacApp: An Application Framework*", BYTE, vol 11(8), pp 189-193, August 1986.
- Shneiderman B:** "*Response Time and display rate in Human Performance with Computer*", Computing Survey, vol 16 nb 3, September 1984.
- Shneiderman B:** "*Designing the user interface: Strategies for effective human-computer interaction*", Ed. Addison Wesley, 1987.
- Smith DCea:** "*Designing the Star User Interface*", Integrated interactive Computer systems, Amsterdam, pp 297-313, 1983.
- Sommerville I:** "*Le génie logiciel et ses application*", Ed. interédition, 1988.
- Stark PH:** "*Petri netz Maschine - a software tool for analysis and validation of Petri Nets*", Proceedings of the 2nd International Symposium in Systems Analysis and Simulation, Berlin, pp 474-475, 1985.
- Stern:** "*Comparison of Window Systems*", BYTE, vol 12(11), pp 265-272, November 1987.
- Stroustrup B:** "*What is object Oriented Programming*", Proceedings of the European Conference on O-O Programming, Paris, France, June 1987.
- SUN:** "*Networking on the Sun workstation: Inter-Process*", 1986.

- SUN:** "*Writing Application for Sun Systems: A guide for the Macintosh Programmer*", Ed. to be published by Addison Wesley, 1990.
- Tanenbaum:** "*Computer Networks*", Ed. P. Hall, 1981.
- Taylor RN:** "*Foundation of the ARCADIA Environment Architecture*", ACM SIGSOFT/SIGPLAN Software Engineering Symposium on Practical Software Development, vol 24 no 2, pp 1-13, 28-30 November 1988.
- TCP/IP:** "*Internet Protocole - DARPA Internet Program Protocol Specification*", USC Information Sciences Institute, 1981.
- Tedd M:** "*the SAPPHIRE project: Building Confidence in PCTE 1229 (1277)*", ESPRIT'87 Achievements and Impact, North Holland, vol 1, pp 325-333, 1987.
- Tomiyama T, Kiriyaama T, Takeda H, Xue D, Yoshikawa H:** "*Metamodel: A key to Intelligent CAD systems*", Research in Engineering Design, Communications of the ACM, New-York, vol 1,1, pp 19-34, 1989.
- Trèves N:** "*COMBAG: a tool for the computation of a basis and a set of generators of semi-flows for Pr/t systems*", Proceedings of the International conference on Parallel Processing and Applications, L'Aquila, Italy, pp 181-192, 1987.
- Trèves N, Bages C:** "*S-CORT une méthode de conception des systèmes de télécommunication*", Proceedings of the International workshop, software engineering & its applications, Toulouse, France, pp 87-104, 1988.
- Van Eijk P, Vissers C, Diaz M:** "*The formal description technique LOTOS*", Ed. North Holland, 1989.
- Vandome G:** "*Etude comparative de NFS, RFS et Andrew*", Convention AFUU, 1986.
- Vautherin J:** "*Parallel systems specification with coloured Petri nets and algebraic abstract data type*", Proceedings of the 7th European workshop on Application and Theory of Petri Nets, Oxford, 1986.
- Wasserman T, Pircher AI:** "*A Graphical, Extensible Integrated Environment for Software Development*", ACM SIGSOFT/SIGPLAN Software Engineering Symposium on Practical Software Development, vol 22, pp 131-142, January 1987.
- Wheeler GR, Wilbur-Ham MC, Billington J, Gilmour JA:** "*Protocol analysis using numerical Petri nets*", Proceedings of the Ffth international workshop on Protocol Specification, Testing and Verification, Toulouse-Moissac,, 1985.
- X-Window:** "*The definitive guide to the X Window System*", Xlib programming manual for version 11, vol 1, pp 635, 1989.
- X-Window:** "*The definitive guide to the X Window System*", X Protocol Reference Manual for X version 11, vol 0, pp 483, 1990.
- XICH TH:** "*Une architecture pour les Atelier de génie logiciel: les couches de boîtes à outils*", Revue Génie logiciel et Système Expert, vol 13, Décembre 1988.

B i b l i o g r a p h i e

Zimmerman: "*OSI reference model-The ISO model of architecture for Open Systems Interconnection*", IEEE Transactions on communication, IEEE, vol COM-28-4, pp 425-432, April 1980.

Index

A

Administration

- des applications 64
- des formalismes 38; 48; 64; 95
- outils 38; 63-65

AGL Atelier de Génie Logiciel; 14; 15; **16**; 18; 19; 20; 29; 41; 43; 44; 118

AIU Application d'Interaction Utilisateur

AMI Atelier de Modélisation Interactif

Animation 13; 28; **29**; 46; 53; 58; 63; 259

Application **118**

Arbre de Question (voir Question)

Atelier de spécification 14; **19-22**

B

Bézier 227

C

CAMI Langage de Communication dans AMI; 199; **205**; 215; 218; 233; 245; 248; 249; 251; 252; 262; 270

Catégories d'application **192**

Client-Serveur 137; **138**; 142; 143; 144; 148; 217; 234; **242**; 244

Comportement d'application **174**

Composabilité **147**

Configuration (voir Gestion de configuration)

Continuité **147**

D

DAA Protocole de Dialogue Application
Atelier

Protocole **118**

Décomposabilité **147**

Dictionnaire (voir Formalisme)

Domaine (Ancien nom de Formalisme)

Données administratives **95**; 98; 110; 131

Données personnelles **95**; 110; 114; 131; 236; 237

E

Efficacité **148**

Efficacité d'un atelier **44**

Encapsulation **147**

Énoncé **92**

langage de description **206**

Entité distante **148**

Entité locale **148**

Esthétique (voir Formalisme)

Extensibilité **42**

F

fichier énoncé **156**

Figure 6.2.

Arborescence dans AMI de l'historique d'un énoncé 238

Figure 1.1

Architecture fonctionnelle générale de l'atelier 71

Figure 1.1.a

Méthodologie de MARS 12

Figure 1.1.b

Utilisation du projet MARS 14

Figure 1.2.a

Cycle de vie du logiciel (AFCIQ) 17

Figure 1.2.b

Atelier de spécification 20

Figure 2.0

Les quatre axes d'un atelier de spécification 24

Figure 3

Séparation physique de l'architecture fonctionnelle 202

Figure 3.0

Le niveau extension système de l'architecture fonctionnelle 74

Figure 3.1

Extensions système pour l'application d'interaction utilisateur 75

Figure 4.0

Les objectifs regroupés en thèmes. 46

Figure 4.1

Le niveau plate-forme de l'architecture fonctionnelle. 83

Figure 4.2

- Le niveau plate-forme squelette 85
- Figure 4.4.a
 - Diminution de fenêtre sous .i.X-Window
OPEN LOOK 86
- Figure 4.4.b
 - Diminution de fenêtre sur .i.Macintosh 87
- Figure 4.5.a
 - Exemples de menus sur Macintosh 88
- Figure 4.5.b
 - Exemple de menus sous .i.X-Window
OPEN LOOK 89
- Figure 5.1
 - Adéquation des besoins aux composantes
50
 - Architecture fonctionnelle simplifiée 93
- Figure 5.10
 - Architecture fonctionnelle simplifiée 132
- Figure 5.2
 - Exemples d'implémentation de
l'architecture de MARS 211
- Figure 5.3
 - Séparation interface graphique -
application de calcul 54
- Figure 5.3.1.a
 - Architecture du GUS-serveur 243
- Figure 5.3.a
 - Exemple du formalisme réseau de Petri
102
- Figure 5.3.b
 - Les classes du formalisme réseaux de Petri
103
- Figure 5.3.c
 - Un réseau local 106
- Figure 5.3.d
 - Description du formalisme de nom interne
"Rdp" 108
- Figure 5.4.a
 - Administration du GUS 111
- Figure 5.4.b
 - Gestion d'un historique sur un énoncé 115
- Figure 5.4.c
 - Gestion complète d'un historique sur un
énoncé 116
- Figure 5.4.d
 - Appel de service 116
- Figure 5.6.a
 - Les types de programmes d'application
121
- Figure 5.6.b
 - Communication système expert à système
expert 122
- Figure 5.6.c
 - Communication entre système expert et
programme d'application 123
- Figure 5.6.d
 - Communication entre programme
d'application et système expert 123
- Figure 5.8
 - Gestion d'un historique sur un énoncé 130
- Figure 5.9
 - Les liens entre formalismes, services et
applications 130
- Figure 6.1
 - Architecture fonctionnelle simplifiée 134
- Figure 6.2.a
 - Architecture des systèmes de fenêtrage 137
- Figure 6.2.b
 - Architecture client/serveur X11 139
- Figure 6.2.c
 - Architecture de notre système de fenêtrage
réparti 144
- Figure 6.3.a
 - Réalisation de l'architecture en couches du
GSV par processus 246
- Figure 6.3.b
 - Réalisation de l'architecture en couches du
GSV par processus 248
- Figure 6.4.a
 - Architecture en couche de la plate-forme
GSV 149
- Figure 6.4.b
 - La couche transport 151
- Figure 6.4.c
 - Ouverture de communication 152
- Figure 6.4.d
 - La couche session 154
- Figure 6.4.e
 - Anticipation de l'ouverture de session 155

Figure 6.4.f
 Choix d'un formalisme 156

Figure 6.4.g
 Réception du formalisme 158

Figure 6.4.h
 Enquête d'ouverture de session 159

Figure 6.4.i
 Les entités activées lors de la multi-sessions 160

Figure 6.4.j
 Déplacement de fenêtres 161

Figure 6.4.k
 Zone de clip 161

Figure 6.4.l
 Commutation de session graphique 162

Figure 6.4.m
 La couche présentation 167

Figure 6.4.n
 Choix d'un service 169

Figure 6.4.o
 l'accès aux applications 172

Figure 6.4.p Les différentes représentations d'un arbre de questions 176

Figure 6.4.q
 La couche application 180

Figure 7
 Le niveau interface 181

Figure 7.1.a
 Pilote interne 184

Figure 7.1.b
 Pilote externe 185

Figure 7.3
 Communication entre applications 187

Figure 7.4
 Le niveau interface d'application d'interaction utilisateur 188

Figure 7.6
 Commutation de session 271

Figure 9
 Architecture fonctionnelle complète 197

Fonctionnalités d'une application **126**

Formalisme 1; 11; 13; 23; 25; **92**

administration 38; 94; 106; 108; 110; 111; 118; 125; 132; 212; 228; 255
formalisme **112**; 196

catégories de 12; 61; 106

classe
connecteur **99**; 157; 220; 221; 262
forme graphique **100**; 133; 157
héritage **103**
information **100**; 221
nœud **99**; 157; 220; 221; 262

de haut niveau 27; 37; 45; 52; 58; 198; 199

description graphique 35; 57; 58; 60; 72; 99; 132; 156; 212; 228

dictionnaire **110**; 179

esthétique 222; 262

exemple
CCS 26
CSP 26
ESTELLE 26; 28; 30; 99
FDT 26
HOOD 15
LDS 26; 28; 30
LOTOS 26; 28
réseaux de Petri 15; 26; 30; 92; 99; **102**; 107; 200; 220; 257
réseaux locaux 39; **104**
RPAS 99; 199; 257; 259
SADT 15; 26; 30; 92; 99
STP 15; 99

fichier de description 213; 239; 262; 264

gérant des 153

gestion 48; 52; 59-61; 63; 98; 106-109; 211; 212-214
GUS 83; 92; **93**-133; 211-212; 236-245

gestion externe 5; 32; 33; 45; 60; 93; 106

hiérarchie 61; 108; 163; 195

langage de description 70; 92; 98; **100**; 106; 156; 177; 178; 199; **205**; 239

méta-modèle 48; 60; 92; 98; 99; 199

multi-formalismes 5; 13; **26**; 33; 38; 45; 52; 58; 60; 98; 153; 198

multi-linguismes 107; 128; 133; 178; 239

palette 157; **220**; 262

transformation 27; 33; 58; **113**; 128; 200; 241; 269

version **101**; 132; 157; 212

- Gestion de configuration **38**; 74; 81; 119; 124; 196; 235; 239; 245
- GKS 90
- GSV Gestionnaire de Station de travail Virtuelle
- GUS Gestionnaire des Utilisateurs et des Services (voir gestion des Formalismes)
- H
- Hiérarchie (voir Formalisme)
- I
- Interface utilisateur **21**
- J
- Jean-xxx (Auteurs de cette thèse)
- L
- Langage interne de communication **33**
- LAQ Langage d'Arbre de Questions (voir Question)
- LDE Langage de Description des Énoncés (voir Énoncé)
- LDF Langage de Description des Formalismes (voir Formalisme)
- LDR Langage de Description des Résultats (voir Résultat)
- M
- MACAO Modélisation Analyse et Conception Assistées par Ordinateur
- Macintosh 42; 61; 75; 77; 78; 199; 202; 203; 209; 210; 212; 215; 216; 225; 227; 247; 261; 274
- MARS Modélisation Analyse et Réalisation de Systèmes
- Méta-modèle (voir Formalisme)
- MIAMI Moteur d'Inférence pour AMI (voir Système expert)
- Mode autonome 5; 56; **72**; 95; 108; 132; 154; 156; 159; 166; 209; 261-263; 265
- Mode d'envoi d'un énoncé **174**
- Modularité **147**
- Multi-formalisme (voir Formalisme)
- N
- NeWS 142
- O
- Objectif 1 **25**; 36; 46
- Objectif 10 **30**; 45; 114; 194
- Objectif 11 **31**; 46; 227
- Objectif 12 **31**; 46; 96
- Objectif 13 **32**; 46
- Objectif 14 **32**; 46; 113; 116; 186
- Objectif 15 **33**; 45; 93; 98; 112; 220
- Objectif 16 **35**; 46; 109; 167; 178; 181; 256
- Objectif 17 **35**; 46
- Objectif 18 **36**; 46; 132
- Objectif 19 **36**; 45
- Objectif 2 **26**; 45; 98; 106
- Objectif 20 **36**; 46; 118
- Objectif 21 **37**; 46; 120; 196
- Objectif 22 **37**; 46; 116
- Objectif 23 **41**; 46; 126; 178
- Objectif 24 **43**; 46; 126
- Objectif 3 **26**; 45
- Objectif 4 **27**; 45; 115
- Objectif 5 **27**; 45; 95; 270
- Objectif 6 **27**; 45; 115
- Objectif 7 **28**; 46; 115
- Objectif 8 **29**; 45; 107; 170; 195
- Objectif 9 **30**; 46; 164; 189; 194; 195
- Objectifs
 tableau Synthétique 45-46
- OPEN LOOK (voir X-Window)
- Orienté objet
 attributs **99**
 classe **99**
 cliente **99**
 héritage **99**
 instances **99**
 méthodes **99**
 type **99**
- OSF/Motif (voir X-Window)
- Ouverture **41**
- P
- Palette (voir Formalisme)
- PHIGS 90
- Postscript 90; 142; 227
- Processus (voir Unix)
- Programme d'application **118**
- Q

Question

Arbre de **127**; 130; 167; 170; 175; 190; 241;
245; 246; 250; 252

CAMI-LAQ 207

Etat des 175

langage de description **207**

LAQ **127**; 205; 207

QuickDraw 77; 90; 227

R

Renseignements **126**

Réseaux de Petri 1; **102**; 178; 182; 194; 199;
200

Résultat

langage de description **207**

S

Service **92**; 109; **114**

d'administration **112**

de calcul **112**; 128

de transformation **113**; 128

interactif **113**; 115; 128; 171; 175

Options **109**; 127; 131; 168; 171; 175; 176;
250; 267

Session **94**; **159**

Squelette d'application 78; 83; 84-91; 188; 189;
195; 198; 209; 210; 214

Structure d'accueil **20**

Système expert

But **121**

MIAMI 200; 254; 257; 260

Paquet de règles **121**

T

TCP/IP (voir Unix)

Transformation (voir Formalisme)

U

Unix 18; 42; 76; 80; 104; 114; 199; 203; 215;
216; 227; 229; 274

AU/X 209

Berkeley 202; 231

Berkeley que nous utilisons nous permet
d'intégrer les communications dans le
réseau. 202

Berkeley) qu'un processus interagisse
directement sur les données d'un autre
processus. Nous avons utilisé cette
particularité pour regrouper dans un
processus toutes les primitives qui
effectuent des opérations sur des objets
(encapsulation). Cet 229

fork 229

login 215; 247

NIS 234

processus 229

sockets streams 231

System V 233

TCP/IP 203; 215; 217; 231; 234; 247; 249;
264

V

Version (voir Formalisme)

X

X-Window 63; 75; 76; 90; 91; 136; **138-143**;
144; 161; 223; 275

OPEN LOOK 55; **77**; 140; 210

OSF/Motif 55; **77**; 140; 210

X11R4 77

Xlib 77; **139**

Y

Yellow Pages (voir Unix NIS)