HAL
open science

Un outil d'aide à l'installation d'UNIX fondé sur les connaissances

Philippe Vial

To cite this version:

Philippe Vial. Un outil d'aide à l'installation d'UNIX fondé sur les connaissances. Génie logiciel [cs.SE]. Ecole Nationale Supérieure des Mines de Saint-Etienne; Université Jean Monnet - Saint-Etienne, 1988. Français. NNT : 1988STET4010 . tel-00813379

HAL Id: tel-00813379

https://theses.hal.science/tel-00813379

Submitted on 15 Apr 2013

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

ECOLE NATIONALE SUPERIEURE
DES MINES DE SAINT-ETIENNE

UNIVERSITE DE SAINT-ETIENNE
N° d'ordre : 22 ID

THESE

présentée par

Philippe VIAL

pour obtenir le titre de

DOCTEUR

DE L'UNIVERSITE DE SAINT-ETIENNE

ET DE L'ECOLE NATIONALE SUPERIEURE DES MINES DE SAINT-ETIENNE

(Spécialité : Informatique, Image, Intelligence Artificielle et Algorithmique)

UN OUTIL D'AIDE A L'INSTALLATION D'UNIX FONDE SUR LES CONNAISSANCES

Soutenue à Saint-Etienne le 22 Décembre 1988

COMPOSITION DU JURY

Monsieur M. HABIB	Président et rapporteur
Monsieur J. KOULOUMDJIAN	Rapporteur
Messieurs C. BERTIN M. JOURLIN F. MIREAUX B. PEROCHE	Examinateurs

ECOLE NATIONALE SUPERIEURE
DES MINES DE SAINT-ETIENNE

UNIVERSITE DE SAINT-ETIENNE
N° d'ordre : 22 ID

THESE

présentée par

Philippe VIAL

pour obtenir le titre de

DOCTEUR

DE L'UNIVERSITE DE SAINT-ETIENNE

ET DE L'ECOLE NATIONALE SUPERIEURE DES MINES DE SAINT-ETIENNE

(Spécialité : Informatique, Image, Intelligence Artificielle et Algorithmique)

UN OUTIL D'AIDE A L'INSTALLATION D'UNIX FONDE SUR LES CONNAISSANCES

Soutenue à Saint-Etienne le 22 Décembre 1988

COMPOSITION DU JURY

Monsieur M. HABIB	Président et rapporteur
Monsieur J. KOULOUMDJIAN	Rapporteur
Messieurs C. BERTIN M. JOURLIN F. MIREAUX B. PEROCHE	Examinateurs

REMERCIEMENTS

Je tiens à exprimer ma reconnaissance aux personnes qui ont accepté de juger ce travail ainsi qu'à toutes celles qui m'ont aidé au cours de ces années de thèse.

- Monsieur le Professeur Michel Habib du Centre de Recherche en Informatique de Montpellier qui me fait l'honneur de présider le jury de cette thèse. Alors qu'il était directeur du Département Informatique de l'Ecole des Mines de Saint-Etienne, il m'a permis de débuter ce travail.

- Monsieur Christian Bertin, Ingénieur à l'Ecole des Mines de Saint-Etienne, pour sa compétence, sa rigueur, sa disponibilité, la formation et l'aide qu'il m'a apportées pendant ces années de thèse.

- Monsieur François Mireaux, gérant de la société SYNC, pour le financement de cette thèse et la liberté dont j'ai pu disposer.

- Monsieur le Professeur Jacques Kouloumdjian de l'INSA de Lyon pour la lecture de ce travail et pour sa présence aujourd'hui.

- Monsieur le professeur Bernard Peroche, directeur du Département Informatique de l'Ecole des Mines de Saint-Etienne, pour l'intérêt porté à cette thèse.

- Monsieur Michel Jourlin, Maître de Conférences au laboratoire TSI (UA 842) de l'Université de Saint-Etienne pour sa participation à ce jury.

- Evelyne Arcuri, Jamal Bencheikh, Rodela Rubio, Jean-Sébastien Salis pour leur aide.

- Tous les membres du Département Informatique pour leur amitié et leur aide.

- Monsieur Jean-Marc Piégay de la société SYNC pour ses conseils dans le maniement du formateur de texte GROFF employé pour éditer ce document.

- Le personnel du service de reprographie pour avoir assuré la réalisation de cet ouvrage.

- Ma famille pour son affection et son soutien.

- Prince mon gros loulou et Timothée ma ti'puce.

15/12/1988

TABLE DES MATIERES

INTRODUCTION 1

1 LE CADRE DU PROJET 1

2 LE BUT DU PROJET 2

3 LE PLAN DE LA THESE 4

PREMIERE PARTIE : L'ADMINISTRATION D'UNIX SYSTEM V 7

1 PRESENTATION D'UNIX 9

1 GENERALITES 9

2 LES FICHIERS 10

2.1 La structure d'un inode 11

2.2 Les fichiers classiques 12

2.3 Les répertoires 14

2.4 Les fichiers spéciaux 14

2.5 L'arborescence UNIX 21

3 LES SYSTEMES DE FICHIERS 23

3.1 Les avantages du partitionnement d'un disque 23

3.2 La structure d'un système de fichiers 24

3.3 Le montage et le démontage d'un système de fichiers 29

3.4 Contrôle et sauvegarde de systèmes de fichiers 30

4 LES UTILISATEURS 33

5 LES FICHIERS D'ADMINISTRATION D'UNIX 35

5.1 Le contrôle du lancement d'UNIX 36

5.2 Le contrôle de la connexion des utilisateurs 37

5.3 Divers 37

5.4 Les réseaux 38

2 L'INSTALLATION D'UNIX SYSTEM V 41

---

1 INTRODUCTION 41

2 LES DIFFICULTES DE L'INSTALLATION D'UNIX 42

3 LES ETAPES DE L'INSTALLATION D'UNIX 43

3.1 L'inventaire de la configuration matérielle 43

3.2 L'inventaire de la configuration logicielle 44

3.3 La définition des partitions à créer sur les disques 45

3.4 Le premier boot depuis le support de distribution 47

3.5 La configuration des disques 47

3.6 L'installation de l'environnement logiciel 49

3.7 La génération d'un nouveau noyau 51

3.8 Le passage en multi-utilisateurs 52

4 LES SOLUTIONS POUR AUTOMATISER L'INSTALLATION D'UNIX 52

DEUXIEME PARTIE : LES SYSTEMES EXPERTS 55

3 LES SYSTEMES EXPERTS 57

1 INTRODUCTION 57
1.1 Les objectifs d'un système expert 57
1.2 L'architecture d'un système expert 58

2 LA MODELISATION DES CONNAISSANCES 62
2.1 La programmation orientée vers les procédures 62
2.2 La programmation orientée vers les règles 62
2.3 La programmation orientée vers les objets 66
2.4 Les représentations hybrides 75
2.5 Comment choisir une représentation des connaissances ? 77

3 LE FONCTIONNEMENT D'UN MOTEUR D'INFERENCES 78
3.1 Le cycle de base d'un moteur d'inférence 78
3.2 Les modes fondamentaux d'invocation des règles 82
3.3 Le choix d'un moteur d'inférences 84

TROISIEME PARTIE : LA REALISATION 85

4 LE DEROULEMENT DE L'ETUDE 87

1 INTRODUCTION 87

2 L'ACQUISITION DES CONNAISSANCES POUR INSTALLER UNIX 87

3 LE DEVELOPPEMENT D'UN SYSTEME EXPERT DE CONSEIL EN EPARGNE 89

4 LA RECHERCHE D'UN ENVIRONNEMENT DE DEVELOPPEMENT DE SYSTEMES EXPERTS 90

5 LA REALISATION DU SYSTEME EXPERT POUR INSTALLER UNIX 91

5 UN SYSTEME EXPERT DE CONSEIL EN PLACEMENT 93

---

1 PRESENTATION DU PROJET 93

1.1 Les objectifs 93

1.2 Le déroulement de l'étude 94

2 LES CARACTERISTIQUES DU PROGICIEL DE DEVELOPPEMENT DE SYSTEMES EXPERTS 98

2.1 Modélisation des connaissances 98

2.2 Stratégie de résolution de conflits 99

2.3 Monotonie 99

2.4 Appel de fonctions externes 99

2.5 Interface homme-machine 100

3 LES PROBLEMES RENCONTRES 100

3.1 Représentation des connaissances 100

3.2 Stratégie de résolution de conflits 100

3.3 Monotonie 102

3.4 Appel de fonctions externes 102

3.5 Interface homme-machine 102

4 CONCLUSION 103

6 REPRESENTATION DES CONNAISSANCES NECESSAIRES POUR INSTALLER UNIX 105

1 INTRODUCTION 105

2 LES CONNAISSANCES DE L'INSTALLATION D'UNIX 106

2.1 Les connaissances factuelles 106

2.2 Les connaissances opératoires 109

3 MODELISATION DES CONNAISSANCES 110

3.1 Les connaissances factuelles 111

3.2 Les connaissances opératoires 115

3.3 Conclusion 119

7 LA REALISATION 121

1 QUELQUES CARACTERISTIQUES DE KOOL 121

1.1 La structure d'une base de connaissances KOOL 121

1.2 Les règles 122

1.3 L'héritage dans KOOL 122

1.4 La stratégie de KOOL pour déterminer la valeur d'un attribut 123

1.5 La fonction koolStart 124

1.6 Les fonctions prédéfinies de KOOL 124

1.7 La fonction getOBjectValue 125

2 L'ARCHITECTURE GENERALE 126

2.1 Le déroulement d'une installation d'UNIX avec notre outil 127

2.2 Les fichiers de l'application 130

3 L'ETAT ACTUEL 130

4 LES DIFFICULTES 132

4.1 Les difficultés liées au domaine d'expertise 132

4.2 Les difficultés liées au générateur de systèmes experts 132

5 EXTENSIONS 134

5.1 L'enrichissement de la base de connaissances 134

5.2 L'amélioration des dialogues 135

5.3 Les nouveaux services 135

5.4 L'implantation 137

5.5 Le générateur 138

CONCLUSION 139

BIBLIOGRAPHIE 143

ANNEXES 153

A : LES CLASSES 153

B : LES REGLES EN CHAINAGE-AVANT 169

C : LES REGLES EN CHAINAGE-ARRIERE 173

D : UN EXEMPLE D'INSTALLATION

INTRODUCTION

1 LE CADRE DU PROJET

La thèse présentée dans ce rapport a été entreprise dans le cadre de la convention CIFRE (Convention Industrielle de Formation par la REcherche) 264/85 établie entre la société SYNC et l'Ecole Nationale Supérieure des Mines de Saint-Etienne.

La réalisation de cette thèse a comporté quatre activités très différentes, à savoir :

- l'acquisition des connaissances nécessaires à l'administration et à l'installation d'UNIX[1] .

- l'acquisition des compétences nécessaires au développement d'un système expert. J'ai pu acquérir ces compétences en participant à la réalisation d'une maquette de système expert pour l'aide aux particuliers dans le choix des produits d'épargne dans le cadre d'un contrat établi entre la Société Lyonnaise de Banque et l'Association pour la recherche et le développement des méthodes et processus industriels (ARMINES).

- le choix d'un environnement de développement de systèmes experts adapté au domaine d'expertise. Nous avons retenu le générateur de systèmes experts KOOL[2] de BULL dans le cadre des conventions de collaborations, pour la diffusion de KOOL, mises en place entre BULL et les établissements de recherche et d'enseignement.

- la réalisation du système expert pour installer UNIX.

La rédaction de ce rapport est donc l'aboutissement d'un effort important, en effet j'ai dû, avant de débuter la réalisation du système expert, devenir un ingénieur système UNIX car j'étais à la fois l'expert et le cogniticien.

---

1 UNIX est une marque déposée des laboratoires A.T.&T.

2 KOOL est une marque déposée de BULL

2 LE BUT DU PROJET

Le génie logiciel s'est, jusqu'à maintenant, principalement intéressé à l'amélioration, d'une part, de la productivité des développeurs, et d'autre part, de la qualité et de la fiabilité des programmes, et en revanche a quelque peu négligé l'administration des systèmes informatiques. En effet, il n'existe pas encore de logiciels d'"administration assistée par ordinateur" pour aider un ingénieur système dans son travail. Les quelques outils disponibles sont des logiciels qui surveillent l'activité des machines, mais la plupart du temps ils se contentent de fournir une trace brute que l'administrateur doit interpréter.

La complexité de l'administration d'un système dépend en fait du type d'équipement informatique. Sur les micro-ordinateurs personnels, elle est suffisamment simple pour être prise en charge par les utilisateurs. Par contre, sur les mini-ordinateurs et les "mainframes" elle est encore réservée à des spécialistes, les ingénieurs systèmes. Entre ces deux extrêmes, il existe une gamme de machines, les stations de travail et les ordinateurs départementaux, dont l'administration est presque aussi complexe que celles des "mainframes", mais vue l'utilisation de ces ordinateurs elle est le plus souvent à la charge d'utilisateurs mal préparés à cette tâche.

Un exemple typique de cette situation est le système UNIX. UNIX est un bon outil de génie logiciel bien adapté au développement d'applications. En revanche peu d'outils existent pour faciliter l'installation et la maintenance du système UNIX lui-même, en fonction des caractéristiques matérielles de chaque configuration et des besoins logiciels de chaque utilisateur. Or l'installation d'un système UNIX sur un ordinateur peut être considérée comme la génération d'un logiciel complexe à partir de composants logiciels simples. Nous pouvons donc appliquer les techniques du génie logiciel qui à partir d'un ensemble de programmes de base construisent un logiciel personnalisé en les assemblant de manière adéquate.

Faute d'outils d'aide à son installation, l'installation d'UNIX représente donc une charge importante. Cette charge est vivement ressentie au département informatique de l'Ecole des Mines de Saint-Etienne, car nous gérons un important parc de machines UNIX (SM90[3] , SPS9[4] , HP9000[5] , stations de travail MATRA) connectées entre elles par un réseau local ETHERNET[6] . Pour ces raisons, nous avons décidé de développer un logiciel d'aide à l'installation d'UNIX avec les objectifs suivants :

- rendre l'installation d'UNIX possible par un non-spécialiste qui simplement décrit la machine et son utilisation. A partir de cette description notre logiciel d'installation

---

3 SM90 est une marque déposée du CNET

4 SPS9 est une marque déposée de BULL

5 HP9000 est une marque déposée de HEWLET-PACKARD

6 ETHERNET est une marque déposée de RANK XEROX

génère automatiquement les procédures d'installation et de sauvegardes, et permet à l'utilisateur d'assurer l'administration de son système sans avoir à se transformer en ingénieur système. Nous ne voulons pas que notre logiciel apparaisse à l'utilisateur comme une boîte noire. A tout moment, il doit expliquer ses décisions et les commandes qu'il exécute pour que l'utilisateur puisse suivre sa démarche et acquérir en douceur la maîtrise de son système.

- affecter les ressources de la configuration en fonction de l'utilisation envisagée de manière à optimiser les performances

- générer un tableau synoptique représentant la machine qui pourra être consulté avant toute opération de maintenance.

- assister un spécialiste, en effet l'installation d'UNIX est un processus long au cours duquel il convient de respecter parfaitement l'enchaînement des différentes étapes, et il est rare que même un expert ne commette pas au moins une erreur lors d'une installation.

Si les fournisseurs de systèmes UNIX proposent des procédures d'installation, ces dernières se contentent d'installer un système prédéfini en fonction des caractéristiques des principaux composants matériels de la configuration. En revanche, si la personne chargée de l'installation désire définir un système adapté aux besoins des futurs utilisateurs, elle dispose au mieux d'outils qui assurent la génération de commandes syntaxiquement correctes. Comme il n'existe pas d'outil d'aide pour déterminer la configuration d'un système en fonction de son utilisation, l'installateur doit pour prendre ses décisions parfaitement connaître, d'une part, les caractéristiques des matériels et des logiciels de sa machine et, d'autre part, l'administration d'UNIX.

Nous pensons que nous pouvons expliquer cette absence d'outils d'aide à l'installation par le fait que l'installation demande des connaissances qui viennent de deux domaines différents : le matériel et le logiciel. En effet, d'un côté, les constructeurs de matériels informatiques ont principalement développé des outils d'aide pour déterminer les configurations matérielles des ordinateurs, comme R1 [McDermott82] ou NOEMIE [Brignon87], et de l'autre côté, les concepteurs de logiciels se sont surtout attachés à construire des outils pour faciliter la génération, la gestion de configuration, la maintenance des logiciels, comme XCM [Kemppainen86] ou REMAP [Dhar86].

De plus l'automatisation de l'installation d'UNIX est une tâche complexe car il est difficile d'exécuter le logiciel d'installation sur une machine avant la mise en place de son système d'exploitation. Ainsi, Harris a développé COGITO [Harris86] un système expert qui conseille un administrateur système pour installer UNIX 4.2BSD sur un VAX, sur un ordinateur personnel. Du fait de ce choix COGITO ne peut qu'établir un plan d'action pour installer UNIX. A partir de la description des caractéristiques de la configuration à installer, COGITO fournit à l'utilisateur des conseils pour définir un système UNIX adapté à sa configuration.

Même si COGITO est extrêment précis dans les conseils qu'il donne (jusqu'aux commandes de l'éditeur de texte pour modifier un fichier), l'installation reste entièrement à la charge de l'utilisateur et les risques d'erreurs sont importants.

Comme les procédures rudimentaires d'installation qui accompagnent les systèmes UNIX se présentent le plus souvent sous la forme de fichiers de commandes, nous avons débuté notre travail par le développement d'une maquette en employant comme langage de programmation le shell. Mais, à cause de sa complexité, cette première maquette est rapidement devenue difficilement maintenable. Ceci nous a conduits à rechercher des modes de représentation des connaissances autres que la programmation traditionnelle par fonctions et procédures. L'existence de COGITO, de systèmes experts pour configurer matériellement des ordinateurs ou pour générer intelligemment des logiciels ayant montré la validité de l'approche système expert dans des domaines voisins de celui que nous envisageons, nous avons donc décidé de développer un système expert pour installer UNIX.

Dès à présent, nous tenons à souligner l'originalité de notre travail par rapport à celui de Harris. En effet, notre outil d'aide à l'installation d'UNIX, non seulement, détermine une configuration qui essaie de satisfaire au mieux les besoins des utilisateurs, mais aussi génère et exécute les commandes nécessaires à sa mise en place. Pour atteindre cet objectif, nous avons été amenés à définir une nouvelle méthode pour réaliser l'installation d'UNIX.

La diversité et la complexité des connaissances (caractéristiques des matériels et des logiciels, principes d'installation, commandes UNIX...) [Vial88-2] de ce domaine d'expertise nous ont conduits à employer conjointement les trois paradigmes de programmation, procédures, objets et règles, pour les représenter. Parmi les différents environnements de développement de systèmes experts qui disposent de ces trois paradigmes, nous avons retenu KOOL de BULL [Vial88-1].

3 LE PLAN DE LA THESE

Cette thèse comporte trois parties. Dans la première, nous présentons UNIX. Le chapitre 1 est consacré à la présentation des systèmes de fichiers, des fichiers spéciaux et des fichiers d'administration alors que dans le chapitre 2 nous décrivons les difficultés rencontrées pendant l'installation d'UNIX, une installation idéale et les solutions possibles pour automatiser cette installation.

La deuxième partie composée du chapitre 3 commence par rappeler les objectifs et l'architecture d'un système expert, puis expose les trois paradigmes de programmation utilisables pour modéliser les connaissances et enfin le fonctionnement d'un moteur d'inférences.

La troisième partie présente la réalisation effectuée pendant cette thèse. Le chapitre 4 définit les différentes étapes de ce travail. Puis le chapitre 5 expose le système expert de conseil en placement que nous avons développé pour la Société Lyonnaise de Banque et les enseignements que nous avons pu tirer de cette expérience. Dans le chapitre 6, nous rappelons,

tout d'abord, les connaissances nécessaires à l'installation d'UNIX, et ensuite, nous montrons comment nous les avons représentées dans notre système expert. Pour terminer le chapitre 7 présente l'implémentation de notre système expert en insistant sur :

- les caractéristiques de l'environnement de développement de systèmes experts employé
- l'architecture générale du système expert
- l'état actuel de notre système expert
- les difficultés rencontrées
- les extensions ultérieures.

PREMIERE PARTIE

L'ADMINISTRATION D'UNIX SYSTEM V

Chapitre 1

PRESENTATION D'UNIX

1 GENERALITES

UNIX a été développé à partir de 1969 principalement par Ken Thompson et Dennis Ritchie [Ritchie78] dans les Bells Laboratories dans le but de fournir un environnement confortable de production de logiciels et de production de documents. C'est un système d'exploitation multi-utilisateurs constitué :

- d'un noyau temps partagé qui offre les services de base pour :
    - la gestion des processus
    - la gestion des fichiers
    - la gestion des entrées-sorties
- d'une large palette d'utilitaires qui fournit les services de haut niveau et forme un environnement de développement complet.

Ses principales caractéristiques sont :

- une grande portabilité, car il est lui-même écrit en langage C, langage disponible sur presque tous les ordinateurs
- la disponibilité de plusieurs langages de commande interprétés très puissants
- la possibilité de créer des processus asynchrones
- un système de fichiers hiérarchisé qui peut être découpé en plusieurs sous-systèmes démontables
- des interfaces d'échange identiques pour :
    - les fichiers
    - les périphériques

- les programmes

Cette dernière propriété, qui a fait dire que "tout dans UNIX n'est que fichier" [Kernighan84], explique l'importance primordiale de la définition et de la mise en place du système de fichiers lors de l'installation d'UNIX. Aussi débuterons-nous la présentation des connaissances que l'ingénieur système doit posséder pour être capable d'installer UNIX par la description des fichiers et des systèmes de fichiers. Ensuite nous définirons les utilisateurs et les groupes d'utilisateurs. Nous terminerons ce chapitre par la présentation des fichiers d'administration.

2 LES FICHIERS

Le système de fichiers UNIX est organisé en arborescence avec une racine unique. Dans cette arborescence, il existe trois types de fichiers :

- les répertoires ou catalogues : les noeuds de l'arborescence
- les fichiers classiques : les feuilles de l'arborescence
- les fichiers spéciaux qui représentent les périphériques (ce sont aussi des feuilles de l'arborescence)

Un fichier UNIX constitue l'association des trois entités suivantes :

- ses désignations externes, c'est-à-dire les noms que lui ont donnés les utilisateurs
- ses caractéristiques : son propriétaire, ses droits d'accès, sa localisation sur le disque, ...
- son contenu.

Le contenu d'un fichier est une suite d'octets qui ne possède aucune structure imposée par le système et aucune signification. Sa structuration et sa signification dépendent seulement des programmes qui l'utilisent. Ceci est valable aussi bien pour les fichiers classiques que pour les fichiers spéciaux. Par exemple, le fichier spécial représentant un disque permet de voir ce dernier comme n octets consécutifs (utilitaire */bin/dd* de copie physique) ou comme un système de fichiers (utilitaire */etc/fsck* de vérification de la cohérence des systèmes de fichiers). Chaque fichier possède une description de ses caractéristiques séparée de son contenu : l'inode qui contient entre autres toutes les informations nécessaires à UNIX pour accéder au contenu du fichier sur le disque. Nous poursuivrons cet exposé par la description de la structure d'un inode, puis par la présentation de chaque type de fichiers et enfin par la présentation de l'arborescence UNIX. Cependant, nous allons tout d'abord définir les deux notions suivantes :

- le bloc est l'unité de mesure de la taille des disques et des systèmes de fichiers. Pour des raisons historiques, c'était la taille des secteurs des disques des machines sur lesquelles UNIX a été développé, la taille d'un bloc est de 512 octets. Ainsi toutes les

tailles (systèmes de fichiers, place libre sur un disque, espaces occupés, ...) sont exprimées en nombre de blocs par les commandes UNIX que ce soient des résultats ou des paramètres.

- le bloc logique est l'unité d'allocation du système de fichiers, mais aussi l'unité d'échange entre le disque et le cache-disque. La taille des blocs logiques varie avec les systèmes, si UNIX version 7 utilisait des blocs logiques de 512 octets (il y avait alors égalité entre les blocs et les blocs logiques), la plupart des UNIX system V utilisent des blocs de 1024 octets, mais certaines versions définissent des blocs de 2048 et même 4096 octets. Par la suite nous supposerons qu'un bloc logique a une taille de 1024 octets.

2.1 LA STRUCTURE D'UN INODE

Le mot inode est le résultat de la contraction des deux mots anglais "index" et "node", aussi quelques auteurs français utilisent le terme "noeud d'index". Nous avons préféré conserver le terme anglais d'origine car il est le seul véritablement employé. Un inode permet d'accéder à l'implantation unique sur disque du contenu du fichier. Toutefois un fichier peut posséder plusieurs noms externes, ces différents noms constituent les liens du fichier, et sont associés au même inode, donc aux mêmes caractéristiques et au même contenu.

La taille d'un inode est de 64 octets, nous y trouvons les informations suivantes :

- le mode et le type du fichier (2 octets)
  le mode définit les droits d'accès des utilisateurs à ce fichier et certaines particularités de gestion des fichiers contenant des exécutables très spécifiques à UNIX. Pour déterminer les droits des utilisateurs sur un fichier, UNIX distingue trois catégories d'utilisateurs :

  - le propriétaire du fichier
  - le groupe d'utilisateurs auquel appartient le propiétaire
  - les autres utilisateurs

  Pour les exécutables UNIX rajoute trois informations :

  - le "sticky" bit qui indique qu'après son chargement dans la zone de swap le programme doit y rester en permanence
  - le "setuid" bit qui précise sous quelle identité effective le programme doit s'exécuter
  - le "setgid" bit qui définit à quel groupe effectif le programme va appartenir pendant son exécution

- le nombre de liens existant sur le fichier (2 octets)
- l'identification de l'utilisateur (2 octets)
- l'identification du groupe de l'utilisateur (2 octets)
- la taille en octets du fichier (4 octets)
- une table de 13 numéros de blocs (chaque numéro est codé sur 3 octets)
    - 10 de données brutes
    - 1 de simple indirection
    - 1 de double indirection
    - 1 de triple indirection
- un octet inutilisé
- la date du dernier accès au fichier (4 octets)
- la date de la dernière modification du fichier (4 octets)
- la date de la dernière modification de l'inode (4 octets)

2.2 LES FICHIERS CLASSIQUES

L'utilisateur voit un fichier comme une suite d'octets logiquement contigus, mais ces octets ne sont pas nécessairement alloués sur le disque de manière contiguë. UNIX utilise les numéros de blocs logiques conservés dans l'inode pour accéder au contenu du fichier de la manière décrite dans la figure 2.1. L'usager, par exemple, pour lire des données doit indiquer leur position par rapport au début du fichier, c'est-à-dire le déplacement relatif en octets depuis le début du fichier, le noyau UNIX se charge alors de convertir cette adresse en numéro du bloc logique contenant ces données et en déplacement depuis le début du bloc.

L'utilisation d'indirections permet de conserver une structure d'inode compacte et de taille constante, tout en permettant de gérer des fichiers de taille importante. En pratique, la taille d'un fichier ne peut pas dépasser 4 giga-octets (2**32) car elle est stockée sur 4 octets dans l'inode.

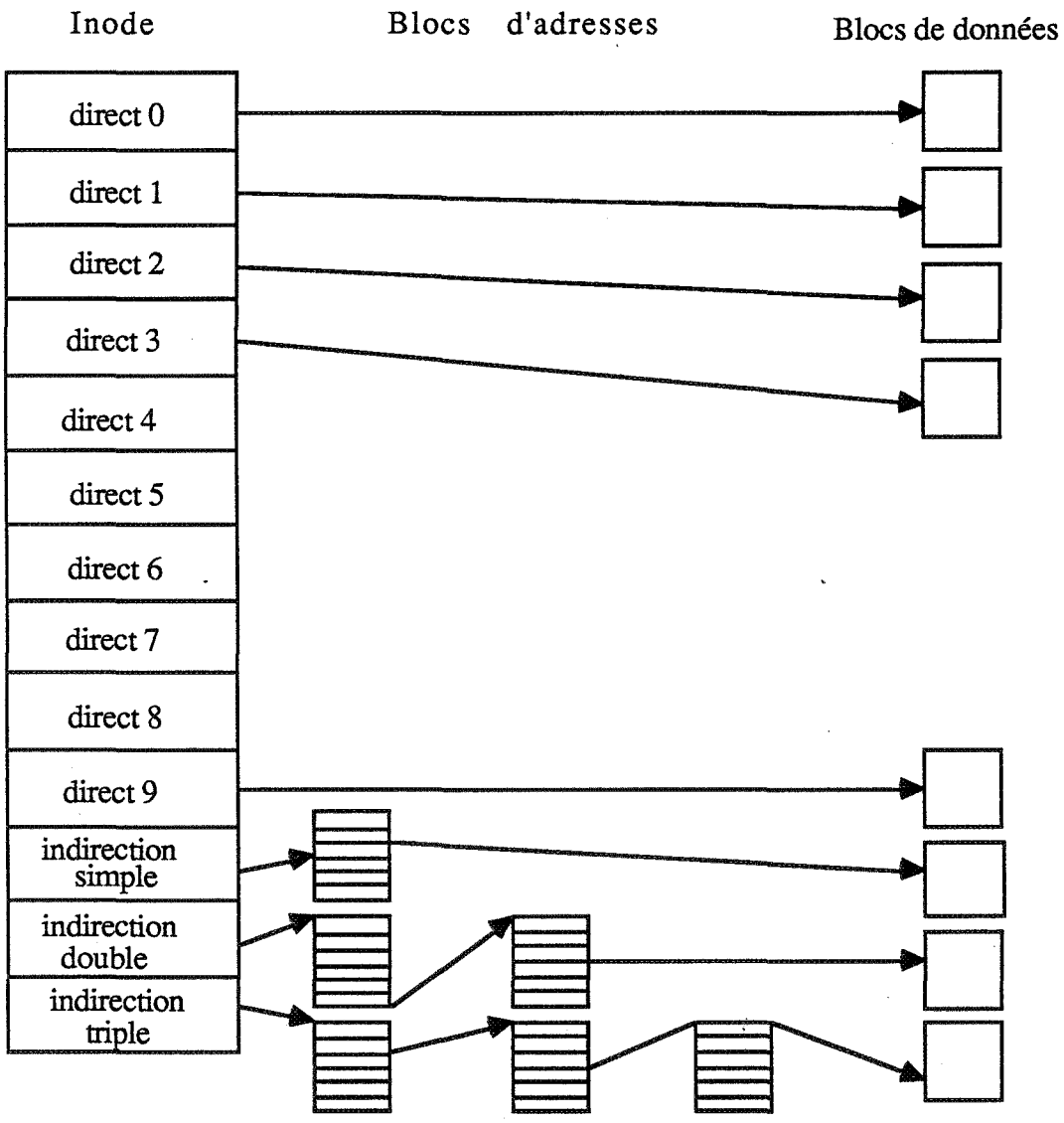

Inode
Blocs d'adresses
Blocs de données
direct 0
direct 1
direct 2
direct 3
direct 4
direct 5
direct 6
direct 7
direct 8
direct 9
indirection simple
indirection double
indirection triple

Figure 2.1

2.3 LES REPERTOIRES

Un répertoire est un fichier dont les données sont interprétées par le système de gestion de fichiers comme une suite d'entrées. Chaque entrée représente l'association d'un nom externe avec un inode qui peut aussi bien décrire un fichier, un sous-répertoire, ou un fichier spécial. Une entrée se compose :

- du numéro d'inode codé sur 2 octets
- du nom externe du fichier limité à 14 octets

UNIX stocke les données du répertoire de la même façon que celle d'un fichier classique en utilisant le système d'indirection. La première entrée d'un répertoire correspond au lien . sur lui-même, et la seconde au lien .. sur son répertoire père.

2.4 LES FICHIERS SPECIAUX

Une des caractéristiques les plus remarquables d'UNIX est la manière dont il traite les périphériques (mémoire, disques, dérouleurs de bandes, les voies de communication, etc). Plutôt que de disposer de fonctions particulières pour accéder aux périphériques, comme c'est le cas dans la plupart des autres systèmes d'exploitation, à chaque périphérique est associé un fichier spécial et un pilote. Quand le noyau UNIX rencontre une référence à un fichier spécial, il transmet la demande au pilote correspondant qui génère les ordres de gestion physique de ce périphérique. Par exemple, nous pouvons parfaitement copier le contenu d'une cartouche magnétique dans un fichier comme s'il s'agissait d'un fichier classique :

```
cp /dev/ct/0 copie
```

Le programme *cp* ne possède pas de renseignement sur la particularité du fichier */dev/ct/0*, pour lui ce n'est qu'une suite d'octets.

Contrairement à un fichier classique ou à un répertoire, un fichier spécial ne possède pas de zone de données, il n'existe que par son inode qui est légèrement différent des inodes standard. En effet l'inode d'un fichier spécial ne contient ni les numéros de blocs de données, ni la taille du fichier qui sont remplacés par des informations identifiant le pilote qui gère le périphérique : le majeur et le mineur du fichier spécial. Le majeur (M) identifie la classe de tous les périphériques gérés par un même pilote (sous-programme contrôlant les entrées-sorties). Et le mineur (m) précise auquel des périphériques de cette classe est destinée la requête. En général, il existe une "formule magique" permettant de passer de l'adresse hardware du périphérique et de son type aux majeurs et mineurs. Cette formule est spécifique à chaque machine et à chaque périphérique. Là s'arrête le niveau de standardisation d'UNIX.

Enfin, il existe deux types de fichiers spéciaux :

- les fichiers de type bloc (b) qui accèdent aux périphériques (disques, disquettes, dérouleurs de bandes magnétiques, ...) à travers un cache-mémoire. L'interface de type bloc

est la seule qui autorise l'accès à la structuration en système de fichiers d'un disque.

- Les fichiers de type caractère (c) qui accèdent aux périphériques (terminaux, imprimantes, réseaux, ...) directement sans passer par un cache-mémoire.

Un périphérique de type bloc peut aussi avoir une interface de type caractère. Dans ce cas, il est représenté par deux fichiers spéciaux : un fichier de type bloc et un fichier de type caractère.

Un fichier spécial est créé par la commande */etc/mknod*, par exemple pour créer le fichier spécial qui représente la mémoire physique du système :

```
/etc/mknod /dev/mem c 3 0
```

Traditionnellement tous les fichiers spéciaux sont regroupés dans le répertoire */dev*, et ses sous-répertoires. Sans ces fichiers UNIX est incapable de fonctionner, aussi l'ingénieur système doit apporter le plus grand soin à leur mise en place pendant l'installation, malheureusement les renseignements nécessaires aux calculs des majeurs et des mineurs sont parfois difficiles à trouver dans la documentation.

Nous allons maintenant examiner les principaux fichiers spéciaux d'un système UNIX. Les valeurs des majeurs et des mineurs que nous donneront sont celles utilisées par SMX V.2, pour d'autres systèmes elles peuvent être légèrement différentes.

2.4.1 Les pseudo-périphériques

Dans tous les systèmes UNIX, nous trouvons dans le répertoire */dev* des fichiers spéciaux qui ne représentent pas des périphériques existant réellement. Ces fichiers servent de point d'accès à certaines informations, ou correspondent à des facilités de programmation. Ils sont qualifiés de pseudo-périphériques, les principaux sont :

/dev/mem	type = c	majeur = 3	mineur = 0
/dev/kmem	type = c	majeur = 3	mineur = 1
/dev/null	type = c	majeur = 3	mineur = 2
/dev/error	type = c	majeur = 6	mineur = 0
/dev/prf	type = c	majeur = 30	mineur = 0
/dev/tty	type = c	majeur = 2	mineur = 0

Grâce aux fichiers */dev/mem* et */dev/kmem*, il est possible d'accéder respectivement à la mémoire physique du système et à la zone mémoire contenant le noyau UNIX.

Le fichier */dev/null* sert de "décharge publique" aux données que les utilisateurs souhaitent abandonner.

Le fichier */dev/error* sert d'interface entre les processus et les fonctions de traitement des erreurs.

Lorsque la surveillance de l'activité du système est activée, les informations nécessaires à ce suivi sont transmises par des écritures dans le fichier */dev/prf*.

Le fichier */dev/tty* constitue, pour chaque processus, un équivalent du terminal de contrôle, et permet donc des échanges avec ce dernier sans avoir à connaître son nom exact.

2.4.2 Les disques

Les disques physiques et virtuels possèdent à la fois une interface de type bloc et une interface de type caractère. Aussi nous trouvons dans le répertoire */dev/dsk* les fichiers de type bloc des disques, et dans le répertoire */dev/rdsk* les fichiers de type caractère. UNIX connaît les disques par un numéro, par exemple le disque 312 est le deuxième disque du contrôleur 1 du module d'échange 3. Un disque physique est représenté par les fichiers */dev/dsk/cXdY* et */dev/rdsk/cXdY*, et un disque virtuel par les fichiers */dev/dsk/cXdYsZ* et */dev/rdsk/cXdYsZ* avec :

X = numéro du module d'echange
Y = numéro du disque
Z = numéro de la partition

Ainsi, le disque 312 est représenté par */dev/dsk/c3d12* et */dev/rdsk/c3d12* et la partition numéro 4 du disque 300 par */dev/dsk/c3d0s4* et */dev/rdsk/c3d0s4*. Comme ces noms ne donnent que peu de renseignements sur l'utilisation des disques et des partitions, des liens sont créés sur ces fichiers pour pouvoir utiliser des désignations plus imagées, par exemple :

/dev/pd300 sur */dev/dsk/c3d0* pour le disque physique 300 (pd pour "physical disk")
/dev/rvduser1 sur */dev/rdsk/c3d0s2* pour la partition numéro 2 du disque 300 qui contient les fichiers d'un groupe d'utilisateurs (vd pour "virtual disk")

Pour les disques le majeur du fichier spécial est 4 en mode caractère et 0 en mode bloc.

Le mineur d'un disque physique est codé sur un octet de la manière suivante :

- le bit numéro 7 est toujours à 0
- les bits 6 et 5 contiennent le numéro du module d'échange
- les bits 4 et 3 le numéro du contrôleur
- les bits 2,1 et 0 le numéro de l'unité du contrôleur

Par conséquent, le mineur du disque 312 est 106, pour le calculer il suffit d'appliquer la formule :

mineur = (numéro du module d'échange * 32) + (numéro du contrôleur * 8) + numéro de l'unité

d'où 106 = (3 * 32) + (1 * 8) + 2.

Les disques virtuels ont des mineurs compris entre 128 et 255. Le mineur 128 correspond à la partition système et le mineur 129 à la zone de swap. Pour chaque autre partition, le mineur est obtenu en ajoutant 1 au mineur de la partition qui la précède sur un disque. Cependant si la configuration possède plusieurs disques, le mineur de la première partition d'un disque doit être un multiple de 8. Par exemple, dans une configuration disposant de deux disques (300 et 301), le disque 300 contenant la partition système, la zone de swap et une partition utilisateur user1, le disque301 contenant une seule partition utilisateur user2, nous avons :

disque 300 :

c3d0s0	root	mineur = 128
c3d0s1	swap	mineur = 129
c3d0s2	user1	mineur = 130

disque 301 :

c3d1s0	user2	mineur = 136

2.4.3 Les lecteurs de cartouches magnétiques

Comme les disques, les lecteurs de cartouches magnétiques (streamer) possèdent à la fois une interface de type bloc et une interface de type caractère. Les fichiers spéciaux associés se trouvent respectivement dans les répertoires */dev/ct* et */dev/rct*. Un lecteur est représenté par les fichiers :

/dev/ct/cXdY
/dev/ct/cXdYn
/dev/rct/cXdY
/dev/rct/cXdYn

où X = le numéro du module d'échange
Y = le numéro du lecteur
n indique que la cartouche ne doit pas être rembobinée automatiquement après l'exécution d'une commande d'écriture ou de lecture.
Comme la majorité des configurations n'ont qu'un seul lecteur, souvent les caractères "cXd" sont omis et Y vaut 0.

En mode bloc, le majeur d'un lecteur de cartouches est 2 et en mode caractère 7.

Le mineur d'un lecteur est codé de la manière suivante sur un octet :

- le bit 7 vaut 1 en mode bloc et 0 en mode caractère

- le bit 6 vaut 1 sans rembobinage automatique et 0 avec
- les bits 5 et 4 contiennent le numéro du module d'échange
- les bits 3 et 2 le numéro du contrôleur
- les bits 1 et 0 le numéro de l'unité, c'est toujours 0

Nous obtenons, de cette façon, pour le lecteur de cassettes du module d'échange 3 :

/dev/ct/0	majeur = 2	mineur = 176
/dev/ct/0n	majeur = 2	mineur = 240
/dev/rct/0	majeur = 7	mineur = 48
/dev/rct/0n	majeur = 7	mineur = 112

2.4.4 Les dérouleurs de bandes magnétiques

Les dérouleurs de bandes magnétiques possèdent aussi une interface de type bloc et une interface de type caractère. Les fichiers spéciaux associés se trouvent respectivement dans les répertoires */dev/mt* et */dev/rmt*. Un dérouleur est représenté par les fichiers :

/dev/mt/cXdYZ
/dev/mt/cXdYZn
/dev/rmt/cXdYZ
/dev/rmt/cXdYZn

où X = le numéro du module d'échange
Y = le numéro du dérouleur
Z précise la densité de la bande utilisée
m pour 1600 bpi
i pour 3200 bpi
h pour 6250 bpi
n a la même signification que pour un lecteur de cartouches

Comme pour le lecteur de cartouches, les caractères "cXd" sont souvent omis.

En mode bloc, le majeur d'un dérouleur de bandes est 1 et en mode caractère 9.

Pour déterminer le mineur d'un dérouleur de bandes, le codage suivant est employé :

- le bit 7 est toujours à 0
- les bits 6 et 5 représentent la densité de la bande
  1 pour 1600 bpi
  2 pour 3200 bpi
  3 pour 6250 bpi

- le bits 4 vaut 0 si rembobinage automatique et 1 sinon
- les bits 3 et 2 contiennent le numéro du module d'échange
- les bits 1 et 0 le numéro du dérouleur.

Par exemple, pour le dérouleur 0 du module d'échange 2 nous avons :

/dev/mt/0m	majeur = 1	mineur = 40
/dev/mt/0mn	majeur = 1	mineur = 56
/dev/rmt/0i	majeur = 9	mineur = 72
/dev/rmt/0in	majeur = 9	mineur = 88

2.4.5 Les lignes asynchrones

Les lignes asynchrones sont représentées par des fichiers spéciaux de type caractère qui sont regroupés dans le répertoire */dev/al*. A la ligne numéro Y du module d'échange numéro X correspond le fichier */dev/al/cXlY*. Pour des raisons historiques, le fichier */dev/al/cXlY* est aussi accessible par le lien */dev/ttyXY*, en effet dans les versions précédentes d'UNIX les lignes asynchrones étaient représentées par des fichiers nommés ainsi. De plus, si une voie sert à connecter une imprimante l'administrateur crée le lien supplémentaire */dev/<Nom de l'imprimante>*, ou si elle est utilisée pour établir une liaison avec un autre ordinateur il crée également le lien */dev/tty<Nom de l'autre machine>*.

Le majeur des fichiers spéciaux associés aux lignes asynchrones est 1, et le mineur est codé sur un octet de la manière suivante :

- les bits 7, 6, 5, 4 représentent le numéro du module d'échange
- les bits 3, 2, 1, 0 le numéro de la ligne.

En conséquence, le mineur d'une ligne asynchrone se calcule grâce à la formule :

mineur = (numéro du module d'échange * 16) + numéro de la ligne

Par exemple, si la voie 3 du module d'échange 1 est employée pour relier l'imprimante série pr1, elle sera représentée par les fichiers :

/dev/al/c1l3
/dev/tty13
/dev/pr1

Une ligne asynchrone joue un rôle particulier, c'est la ligne qui permet de démarrer le système grâce à la console système. Cette ligne est représentée par un fichier spécial qui possède les deux liens externes suivants : */dev/console* et */dev/systty*. Pour UNIX, cette ligne constitue la console système physique. Or UNIX ne dialogue pas directement avec cette ligne,

mais avec une ligne virtuelle : la console système virtuelle représentée par le fichier spécial */dev/syscon*. Lors du lancement d'UNIX, il faut donc établir un lien entre la console système virtuelle et la ligne asynchrone qui servira effectivement de console système, généralement la console système physique.

Par exemple sur une SM90, la console système physique est gérée par le module de traitement dont le numéro de module d'échange est 0, elle est donc représentée par les fichiers spéciaux :

/dev/console	majeur = 0	mineur = 0
/dev/systty	majeur = 0	mineur = 0

Si elle est effectivement utilisée comme console système, elle sera également accessible par le lien :

/dev/syscon	majeur = 0	mineur = 0

2.4.6 Le réseau local Ethernet

Si la configuration comprend un module d'échange Ethernet, les logiciels, qui gèrent les transferts de données sur ce support, accèdent au pilote grâce au fichier spécial de type caractère suivant :

/dev/ethernet qui identifie le module d'échange Ethernet. Le majeur dépend du fabricant du module d'échange, 20 pour un module d'échange BULL, et 26 pour un module d'échange HACKERS. Le mineur est le numéro du module d'échange Ethernet, sa valeur est comprise entre 0 et 15, habituellement le numéro utilisé est 15.

L'utilitaire rlogin, qui permet de se connecter sur une machine distante à travers un réseau Ethernet, utilise deux fichiers spéciaux pour établir la connexion. Ces fichiers sont appelés des pseudo-terminaux, il y a un pseudo-terminal maître et un pseudo-terminal esclave. Les pseudo-terminaux esclaves sont représentés par les fichiers spéciaux */dev/ttypX* dont le majeur est 11 et le mineur vaut X. Les pseudo-terminaux maîtres sont représentés par les fichiers spéciaux */dev/ptypX* dont le majeur est 12 et le mineur vaut X. Un module d'échange peut gérer au plus 16 connections, par conséquent la valeur de X est comprise entre 0 et 15.

Pour un module d'échange Ethernet HACKERS de numéro 15 gérant 5 connexions, nous aurons donc les fichiers spéciaux :

- */dev/ethernet*	majeur = 26	mineur = 15
- */dev/ttyp[0-4]*	majeur = 11	mineur = [0-4]
- */dev/ptyp[0-4]*	majeur = 12	mineur = [0-4]

2.4.7 Les liaisons synchrones X25

Les deux liaisons synchrones X25 du module d'échange X25 fabriqué par OST sont représentés par les fichiers spéciaux de type caractère:

/dev/ostN où N est le numéro de liaison (0 ou 1).

Le majeur de ces fichiers vaut 16 et le mineur est obtenu grâce à la formule de calcul :

mineur = numéro du module d'échange + (16 * numéro de liaison)

Par exemple, le module d'échange de numéro 13 gère les lignes :

- */dev/ost0* majeur = 16 mineur = 13
- */dev/ost1* majeur = 16 mineur = 29

La fonction PAD permet d'offrir aux utilisateurs, sur les liaisons synchrones mais aussi sur un réseau Ethernet, les mêmes services de connexion à distance, de transfert de fichiers, et de messagerie que ceux disponibles entre des machines reliées par des lignes asynchrones. Pour conserver la compatibilité avec les logiciels conçus pour des lignes asynchrones, la notion de terminaux virtuels ou de voies logiques PAD, et la notion de périphérique de contrôle ont été introduites. Ces voies logiques PAD sont représentées par les fichiers spéciaux de type caractère */dev/padX* dont le majeur est 28, et le mineur le numéro de la voie X. Le fichier */dev/pad0* correspond au périphérique de contrôle. Sur les SM90, nous pouvons installer jusqu'à 16 voies logiques.

Nous terminons ici cette présentation des fichiers spéciaux que nous pouvons rencontrer sur une machine UNIX, car ce sont les plus utilisés. Naturellement, en fonction des options matérielles de la configuration, il pourra être nécessaire de créer d'autres fichiers spéciaux.

2.5 L'ARBORESCENCE UNIX

Maintenant que nous connaissons les différents types de fichiers utilisés par UNIX, nous allons présenter l'arborescence UNIX, c'est-à-dire les principaux répertoires qui la composent et les fichiers qu'ils contiennent.

/ désigne la racine du système, nous y trouvons notamment le fichier */unix* qui est l'exécutable du noyau UNIX.

/bin regroupe les principales commandes UNIX comme *cp, ls, grep, cc, make, ...*

/dev comme nous l'avons vu précédemment contient les fichiers spéciaux.

/etc est le répertoire de l'administration, nous y trouvons les utilitaires (exécutables et fichiers de commandes) et les fichiers de données qui permettent d'assurer l'administration d'UNIX.

/lib regroupe les bibliothèques de base du système.

/mnt est utilisé pour monter, le temps d'une copie ou d'une sauvegarde, un système de fichiers d'un disques mobile ou d'une disquette.

/tmp reçoit les fichiers temporaires créés par certains utilitaires *vi, cc...*

/usr est le point de départ d'une sous-arborescence de structure semblable à celle de la racine.

/usr/adm contient les fichiers du suivi de l'activité du système.

/usr/bin des commandes UNIX comme *vi, troff, cxreff, mail, ...*

/usr/catman le manuel UNIX.

/usr/include les fichiers en-têtes des programmes C.

/usr/include/sys les fichiers en-têtes système pour C.

/usr/lib les bibliothèques pour C, PASCAL... et les utilitaires de gestion du spoule d'impression.

/usr/lib/terminfo la base de données qui décrit les fonctionnalités d'un grand nombre de modèles de terminaux.

/usr/lib/uucp les fichiers d'administration et les utilitaires d' *uucp* (uucp est un ensemble d'utilitaires qui permettent de se connecter sur une autre machine UNIX, de tranférer des fichiers entre deux machines UNIX, et d'exécuter à distance une commande sur une autre machine UNIX).

/usr/lib/x400 les fichiers d'administration et les utilitaires de la messagerie normalisée X400.

/usr/local les applications propres au site.

/usr/local/bin les exécutables de ces applications.

/usr/local/lib les bibliothèques de ces applications.

/usr/local/src les sources de ces applications.

/usr/spool accueille, dans ses répertoires, les fichiers temporaires et les fichiers d'administration de certains utilitaires, ainsi :

/usr/spool/lp les fichiers temporaires du spoule d'impression, ainsi que les fichiers d'administration indispensables à son fonctionnement.

/usr/spool/uucp les fichiers d' *uucp*.

/usr/spool/x400 les boîtes aux lettres de la messagerie.

/usr/src les codes sources des commandes, des bibliothèques si vous possédez une licence source d'UNIX.

/usr/src/terminfo les sources des descriptions des modèles de terminaux.

/usr/tmp d'autres fichiers temporaires.

En plus de ces répertoires dont les noms sont à peu près constants d'une version d'UNIX à une autre, il existe aussi un répertoire qui contient les fichiers nécessaires à la génération du noyau UNIX, mais son nom dépend du fournisseur, pour SMX V.2 c'est */sys*.

Enfin il faut rajouter les répertoires qui accueillent les fichiers des utilisateurs, et là il n'y a pas de standard, les ingénieurs systèmes laissant libre cours à leur imagination. Toutefois il semble que le répertoire */users* soit très utilisé comme racine des répertoires des utilisateurs.

3 LES SYSTEMES DE FICHIERS

Nous avons vu, pendant l'étude des fichiers spéciaux des disques, qu'UNIX permet de découper un disque physique en plusieurs disques virtuels. Sur chaque disque virtuel, il est possible de générer un système de fichiers indépendant. Nous allons exposer les avantages apportés par cette démarche, puis la structure d'un système de fichiers, et comment les différents systèmes de fichiers sont réunis pour construire l'arborescence UNIX.

3.1 LES AVANTAGES DU PARTITIONNEMENT D'UN DISQUE

Le découpage des disques physiques en disques virtuels ou partitions présente les avantages suivants :

- accélérer les accès disques, en effet vu l'organisation des systèmes de fichiers plus ceux-ci sont petits plus les temps d'accès aux données sont courts. Car le taux de dispersion des blocs logiques d'un fichier augmente avec la taille du système de fichiers.

- isoler les zones vitales comme la racine, le répertoire */usr* des zones potentiellement perturbatrices comme les répertoires */tmp* et */usr/spool* où de nombreuses commandes créent des fichiers temporaires.

- continuer à travailler sur les autres disques virtuels quand une partition est désorganisée ou défectueuse.

- structurer les disques en regroupant dans des partitions soit des données soit des fichiers possédant des points communs comme par exemple les fichiers d'un groupe d'utilisateurs, ou l'ensemble d'une base de données.

- affecter des droits d'accès différents à chaque partition.

- répartir l'espace disque disponible entre les utilisateurs en affectant si possible à chaque groupe une partition. Comme UNIX alloue à un utilisateur de la place disque tant qu'il trouve de la place libre, nous empêchons ainsi qu'un consommateur fou bloque complètement le système. Il saturera seulement le disque virtuel où il peut écrire, et seuls les utilisateurs du même groupe seront pénalisés.

- définir des disques dont la taille est inférieure à la capacité maximale des supports magnétiques de sauvegarde, et donc éviter l'utilisation de sauvegardes sur des volumes multiples toujours délicate.

- découper l'espace disque en différents domaines [Gould86] indépendants pour améliorer le contrôle d'accès et de ce fait la sécurité d'UNIX.

3.2 LA STRUCTURE D'UN SYSTEME DE FICHIERS

Lors de la création d'un système de fichiers sur un disque, UNIX ajoute au-dessus de l'organisation physique du disque une organisation logicielle qui utilise comme unité de manipulation des données le bloc logique. Nous présenterons seulement la structure statique d'un système de fichiers, nous nous permettons de renvoyer le lecteur intéressé par l'allocation dynamique des blocs aux fichiers par UNIX à l'excellent ouvrage de Maurice J. Bach "The design of the UNIX operating system" [Bach86], l'objectif de cet exposé n'étant pas l'étude du noyau UNIX.

Tout système de fichiers est divisé en quatre parties :

- le bloc d'initialisation

- le superbloc

- la table des inodes

- la zone des blocs

La structure d'un système de fichiers peut être schématisée de la façon suivante :

ORGANISATION D'UN SYSTEME DE FICHIERS

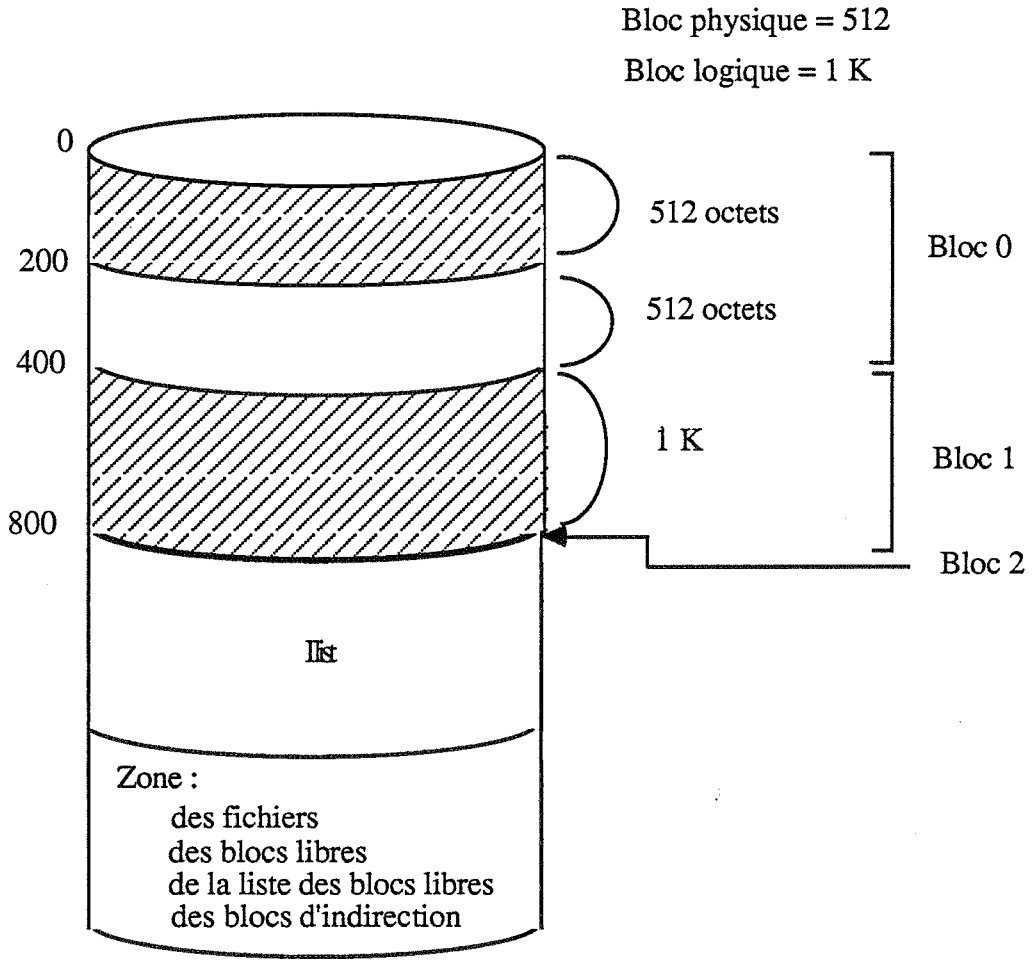


Figure 3.1

3.2.1 Le bloc d'initialisation

Le bloc d'initialisation n'est utilisable que dans le premier système de fichiers de chaque disque physique. Il contient dans les tout premiers octets du disque des informations pour le contrôleur de ce disque. Sur la partition système qui doit impérativement se trouver au début d'un disque physique, ces octets sont alors suivis du lanceur initial (boot) d'UNIX. De toute manière, les 512 premiers octets du bloc 0 de chaque système de fichiers sont réservés au bloc d'initialisation qu'il soit présent ou non, utilisé ou non.

3.2.2 Le superbloc

Le superbloc est la zone d'un système de fichiers où UNIX conserve les informations permettant d'accéder aux blocs et aux inodes libres de ce système de fichiers. Il se trouve juste derrière le bloc d'initialisation et occupe les 512 derniers octets du bloc 0. Pour des raisons de compatibilité, le superbloc commence toujours à l'adresse 512 en octets depuis le début du disque quelque soit la taille du bloc logique. De cette manière, tout système UNIX est capable de lire le superbloc d'un système de fichiers même s'il a été créé avec une autre version d'UNIX. Sa structure est la suivante :

- le numéro du premier bloc logique allouable pour des données
- la taille du système de fichiers en blocs logiques
- le pointeur sur le premier numéro de bloc logique libre de la table des blocs logiques libres
- le numéro du prochain bloc logique de la liste chaînée des blocs logiques libres
- la table des numéros des blocs logiques libres
- le pointeur sur le premier inode libre de la table des inodes libres
- la table des inodes libres
- le sémaphore sur la table des blocs logiques libres
- le sémaphore sur la table des inodes libres
- l'indicateur de modification du superbloc
- l'indicateur de protection d'écriture du superbloc (si cet indicateur est positionné le système de fichiers sera accessible en lecture seulement)
- la date de dernière mise à jour du superbloc
- le nombre total de blocs logiques libres

- le nombre total d'inodes libres
- le nom du système de fichiers
- le nom du volume
- le type du système de fichiers, en fait ce type correspond à la taille du bloc logique exprimée en nombre de blocs de 512 octets (le dernier octet du superbloc).

L'initialisation du superbloc est réalisée par le programme */etc/mkfs* qui assure la création des systèmes de fichiers.

3.2.3 La table des inodes

Comme le superbloc, la table des inodes est créée par */etc/mkfs*. L'ingénieur système peut indiquer le nombre d'inodes à construire pour le système de fichiers, ou accepter la valeur par défaut qui vaut le quart de la taille du système de fichiers en nombre de blocs logiques. Cette table débute au bloc 2 du système de fichiers et occupe des blocs contigus. Le premier inode est inutilisé et l'inode 2 est l'inode de la racine du système de fichiers. Les inodes suivants sont repérés par leur numéro d'indice dans la table des inodes.

3.2.4 La zone des blocs

Cette zone contient :

- les blocs des données des fichiers (cf 2.1 et 2.2)
- les blocs d'indirection pour les fichiers dont la taille dépasse 10240 octets ou 10 fois la taille du bloc logique (cf 2.1 et 2.2)
- les blocs libres, ces blocs sont organisés sous la forme d'une liste chaînée. Chaque bloc de la liste contient un tableau de numéros de blocs libres et le premier élément du tableau donne en fait le prochain élément de la liste.

La figure 3.2 fournit un exemple de cette liste des blocs libres où le premier bloc correspond à la table des blocs libres du superbloc.

LISTE DES BLOCS LIBRES DU SUPER-BLOC

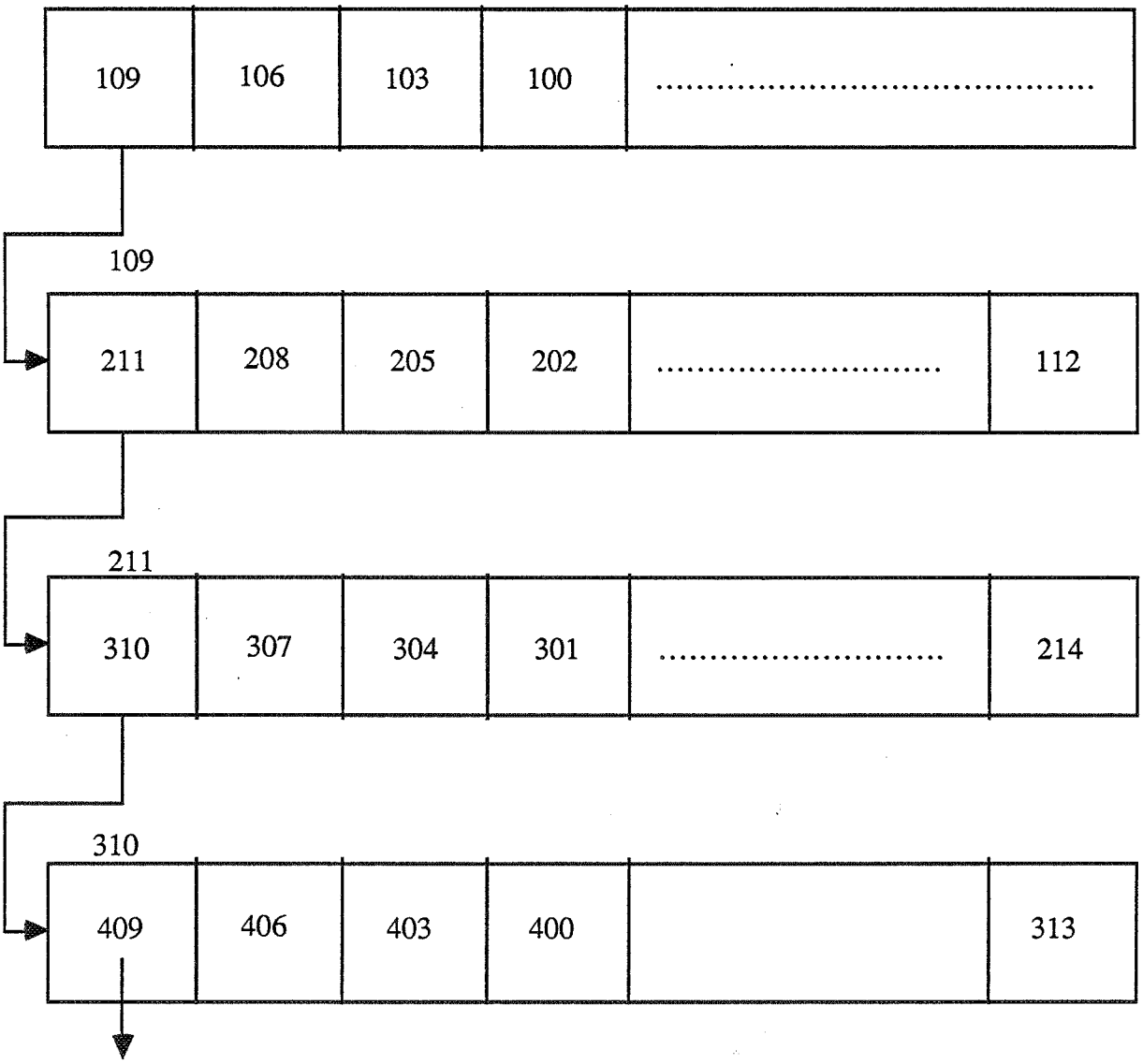


Figure 3.2

3.3 LE MONTAGE ET LE DEMONTAGE D'UN SYSTEME DE FICHIERS

Immédiatement après le lancement d'UNIX, seule la partition système est accessible au sens de l'arborescence logique des fichiers et des répertoires. Par contre les autres partitions, à ce moment-là, sont seulement accessibles par leurs fichiers spéciaux et UNIX ignore leur structure de systèmes de fichiers. Pour les rendre accessibles en tant que systèmes de fichiers, dans l'arborescence UNIX, l'ingénieur système doit réaliser le montage de ces disques virtuels. L'opération inverse est le démontage.

3.3.1 Le montage

L'opération de montage d'un disque consiste à associer la racine du système de fichiers d'une partition du disque à un répertoire de la hiérarchie existante. Pour contrôler le montage des partitions, le noyau gère une table des systèmes de fichiers montés qui est sauvegardée sur disque dans le fichier */etc/mnttab*. Chaque entrée de la table contient :

- le numéro du disque virtuel ou physique du système de fichiers monté,
- un pointeur sur le superbloc du système de fichiers,
- un pointeur sur l'inode de la racine
- un pointeur sur l'inode du répertoire d'accrochage.

Lors du montage, le système installe un drapeau dans la table des systèmes de fichiers montés, afin d'éviter de monter deux fois le même système de fichiers ou de monter deux systèmes de fichiers sur le même répertoire. Après exécution du montage, la racine du système de fichiers est accessible par le répertoire d'accrochage. Les processus peuvent accéder aux fichiers montés et ignorent qu'ils sont détachables. L'association entre le répertoire d'accrochage et la racine du système de fichiers monté permet au noyau de traverser la hiérarchie du système de fichiers, sans avoir recours au savoir de l'utilisateur.

La commande pour monter un système de fichiers est */etc/mount*, par exemple pour monter la partition */dev/vdusers* en lecture et écriture sur le répertoire */users* par :

```
/etc/mount /dev/vdusers /users
```

L' option -r permet de monter un système de fichiers en lecture seule.

3.3.2 Le démontage

Avant de démonter un système de fichiers, le noyau vérifie qu'aucun fichier de ce système de fichiers n'est encore actif. Si un fichier de ce système de fichiers est ouvert, le noyau refuse le démontage. Après démontage, le système de fichiers n'est plus accessible par aucun processus.

La commande pour démonter un système de fichiers est */etc/umount*, par exemple pour démonter la partition */dev/vdusers* on exécute la commande :

```
/etc/umount /dev/vdusers
```

La structure du système de fichiers UNIX, que nous venons d'étudier, est sans redondance et elle est réputée fragile. En plus des problèmes qui peuvent survenir à la suite d'une défaillance de l'alimentation électrique, une source de problèmes graves est une mauvaise définition des droits d'accès des fichiers spéciaux. En effet si un autre utilisateur que le super-utilisateur peut écrire sur le fichier spécial de type caractère, cet utilisateur peut écraser le superbloc, la table des inodes, des blocs libres ou de données et complètement désorganiser le système de fichiers. L'ingénieur système doit donc vérifier régulièrement la cohérence des systèmes de fichiers et réaliser des sauvegardes. Nous allons examiner les utilitaires dont il dispose.

3.4 CONTROLE ET SAUVEGARDE DE SYSTEMES DE FICHIERS

3.4.1 Contrôle des systèmes de fichiers

Deux utilitaires permettent de vérifier la cohérence d'un système de fichiers, ce sont */etc/fsck* et */etc/dfsck*.

/etc/dfsck est intéressant lorsque la configuration possède plusieurs disques physiques, car il permet de vérifier deux systèmes de fichiers sur deux disques différents en parallèle, mais il ne doit pas être utilisé pour contrôler la partition système.

/etc/fsck doit toujours travailler sur des volumes inactifs, il faut donc que UNIX fonctionne en mode mono-utilisateur et que le système de fichiers soit démonté et que */etc/fsck* accède au système de fichiers par son fichier spécial de type caractère. Si le volume est actif, */etc/fsck* risque de trouver une incohérence seulement parce qu'à cet instant des modifications réalisées dans le cache-disque n'ont pas encore été écrites sur le disque. Cette règle doit être impérativement respectée si l'on ne veut pas risquer de détériorer un système de fichiers. Naturellement elle ne s'applique pas à la partition système qui est toujours active et montée automatiquement lors du lancement d'UNIX, on doit utiliser alors le fichier spécial de type bloc.

/etc/fsck détecte les dix défauts suivants :

- des blocs qui appartiennent à plusieurs inodes ou à un inode et à la liste des blocs libres
- des blocs qui sont attribués à un inode ou à la liste des blocs libres, mais qui se trouvent en dehors du système de fichiers, c'est-à-dire des blocs dont le numéro est supérieur à la taille du système de fichiers

- un nombre de liens incorrect, le nombre de liens indiqué dans l'inode est différent du nombre de fois où l'inode est référencé dans l'ensemble de l'arboresence
- des tailles incorrectes, ce peut être
    - la taille calculée du système de fichiers est différente de celle conservée dans le superbloc
    - la taille d'un répertoire n'est pas un multiple de 16
    - la non concordance entre la taille d'un fichier et le nombre de blocs occupés par ce fichier
- un format incorrect d'un inode
- des blocs qui n'appartiennent ni à un inode, ni à un bloc d'indirection, ni à la liste des blocs libres
- des répertoires incorrects
    - un répertoire qui pointe sur un inode non affecté
    - l'association d'un nom externe et d'un inode non affecté
    - un inode qui n'est pas associé à un nom externe
    - un numéro d'inode supérieur au nombre d'inodes du système de fichiers
- un superbloc erroné
    - un nombre total d'inodes supérieur à 65536
    - une table des inodes supérieure à la taille du système de fichiers
- un format incorrect d'un bloc de la liste des blocs libres
- une non-égalité entre :
    - le nombre calculé de blocs libres et celui du superbloc
    - le nombre calculé d'inodes libres et celui du superbloc

Quand */etc/fsck* trouve un défaut, il n'offre malheureusement que deux possibilités : soit supprimer le fichier qui pose problème, soit ne rien faire. Si l'administrateur désire réparer un système de fichiers sans perdre trop d'informations, il doit employer un des deux utilitaires suivants :

- */bin/wi* qui permet de clicher ("to dump" en anglais ou "dumper" en franglais) un système de fichier ou un disque physique

- */etc/fsdb* qui permet de rechercher, de visualiser, et de modifier aisément un inode, un bloc de données dans un système de fichiers
  Par exemple :

```
/bin/wi          /dev/rvdusers
/etc/fsdb /dev/rroot
```

3.4.2 Les sauvegardes

Il existe deux catégories d'utilitaires de sauvegarde, ceux qui réalisent la sauvegarde globale d'un volume, et ceux qui permettent d'archiver un sous-arbre de l'arborescence UNIX, d'où la qualification de copie physique pour les premiers et celle de copie logique pour les derniers. La réalisation d'une sauvegarde cohérente, sous UNIX, ne peut être menée à bien que si la sauvegarde est effectuée sur un ensemble qui n'évolue pas. Lors de la copie physique d'une partition, une manière efficace de s'assurer que l'état de la partition ne sera pas modifié pendant la sauvegarde consiste à démonter cette partition. En effet, le démontage d'une partition force l'écriture sur disque du cache et la rend inaccessible à tous les utilisateurs. La copie physique se fait alors par l'interface de type caractère de la partition, ainsi l'administrateur système évite le passage des données par le cache-disque et peut définir des tampons de bufférisation dont la taille est mieux ajustée aux structures de la partition et du support de sauvegarde.

Les trois utilitaires principaux de copie physique sont :

- *dd* (disk-dump) qui copie un fichier source sur un fichier cible. *dd* admet plusieurs paramètres qui précisent :

  - la partie du fichier à lire

  - la taille du tampon de bufférisation à utiliser pour la copie

  - la position où installer la copie sur le fichier cible

  dd accepte de copier tous les types de fichiers et permet donc de copier des volumes, des partitions ou des sous-ensembles contigus de ceux-ci qui ne sont pas forcément des systèmes de fichiers, par exemple, un disque physique entier ou un volume géré directement par une base de données, car il conserve la structure du volume (système de fichier ou autre). *dd* est aussi très employé pour les sauvegardes de la partition système, par exemple la sauvegarde sur un streamer d'une partition de 80640 blocs avec un cache de 4096 octets se fera par la commande :

```
dd if=/dev/rroot of=/dev/rct/0n bs=4096 count=10080
```

- *volcopy* qui sauvegarde un système de fichiers sur un autre système de fichiers ou sur une bande

- *dcopy* qui sauvegarde un système de fichiers sur un autre en le réorganisant pour optimiser l'accès aux fichiers. Cet utilitaire doit être utilisé régulièrement pour éviter une dégradation des performances du système, en effet lorsque les blocs des fichiers sont éparpillés sur tout un système de fichiers, UNIX perd beaucoup de temps pour les trouver.

Comme les copies logiques travaillent sur des sous-arborescences de l'arborescence UNIX, il est impossible de les réaliser sur des partitions démontées. Pour garantir leur cohérence, nous recommandons de réaliser les sauvegardes logiques de la façon suivante :

- démonter la partition qui contient la sous-arborescence à sauvegarder, ainsi elle devient inaccessible

- monter cette partition en lecture seule, la sous-arborescence est de nouveau accessible mais ne peut pas être modifiée.

UNIX offre deux utilitaires de copie logique :

- *tar* (tape-archive) qui a été conçu au départ pour la sauvegarde sur une bande d'un sous-arbre, mais qui en fait est aussi utilisé pour des sauvegardes sur disques.

- *cpio* qui permet soit de déplacer un sous-arbre dans l'arborescence UNIX, soit de le copier dans un fichier. Utilisé de pair avec l'utilitaire de recherche de fichiers *find*, il autorise la sauvegarde sélective de fichiers selon des critères très fins. Par exemple on peut copier le sous-arbre */ia* sur une cassette avec *cpio* par la commande suivante :

```
cd /ia
find . -print | cpio -ocBv /dev/rct/0n
```

Pour les copies logiques, il ne faut sutout pas désigner les fichiers par leur cheminom absolu pour pouvoir les restaurer correctement dans un sous-arbre différent de celui de départ.

4 LES UTILISATEURS

Pour qu'un utilisateur puisse travailler sur un système UNIX, il doit être connu de celui-ci. UNIX conserve la liste des utilisateurs autorisés à se connecter dans le fichier */etc/passwd*. Chaque ligne de ce fichier décrit un utilisateur, une ligne comporte les renseignements suivants :

- le nom de connexion de l'utilisateur (au plus huit caractères)
- le mot de passe codé de l'utilisateur (un mot de passe est formé de six à huit caractères alphanumériques dont au moins un n'est pas une lettre)
- le numéro de l'utilisateur. Naturellement, sur une machine, un numéro doit être affecté à un seul utilisateur. De plus, lors de la définition d'un utilisateur sur un réseau local, nous conseillons que cet utilisateur ait le même numéro sur toutes les machines auxquelles il aura accès pour éviter des problèmes lors de l'utilisation des logiciels réseaux.
- le numéro du groupe auquel appartient l'utilisateur. Pour le numéro du groupe, nous pouvons faire les mêmes remarques que pour le numéro de l'utilisateur.
- un champ disponible où on trouve souvent le nom complet de l'utilisateur
- le répertoire d'accueil de l'utilisateur. Ce répertoire doit exister et appartenir à une partition montée. L'utilisateur doit être propriétaire de son répertoire d'accueil et pouvoir y accéder.
- le programme d'accueil de l'utilisateur, c'est-à-dire le programme à exécuter lors de la connexion. En général, le programme d'accueil est un shell (*sh*, *csh*, *ksh*) mais on peut limiter les privilèges de l'utilisateur en indiquant ici un interprête de commandes moins puissants (*rsh*) ou une application spécialisée.

Dans le fichier */etc/passwd*, nous trouvons tout d'abord *root*, un utilisateur privilégié : le super-utilisateur qui possède tous les droits. L'administrateur système utilise ce compte lorsqu'il doit réaliser une opération de maintenance. Puis sont définis d'autres utilisateurs systèmes comme *bin, sys, adm, daemon* qui sont les propriétaires des fichiers du système. Ensuite viennent quelques utilisateurs associés à des commandes très utiles comme *date, who, sync, shutdown,* il est ainsi possible de les exécuter sans se connecter. Et d'autres utilisateurs qui assurent des services particuliers comme *lp* qui gère le spoule d'impression ou *uucp* qui gère les transferts de fichiers entre machine UNIX. Pour finir, nous trouvons les utilisateurs du système. Généralement, les utilisateurs systèmes ont un numéro compris entre 0 et 100 (0 pour root), et les autres un numéro supérieur à 100. Voici un extrait du fichier */etc/passwd* d'une de nos machine :

```
root:/Ex062mwOZuRo:0:3:0000-Admin(0000):/:/bin/sh
bin:**NO LOGIN**:2:2:0000-Admin(0000):/bin:/bin/sh
sys:**NO LOGIN**:3:3:0000-Admin(0000):/usr:/bin/sh
adm:**NO LOGIN**:4:4:0000-Admin(0000):/users/adm:/bin/sh
uucp:RE3SnRF/mNdeQ:5:5:0000-Admin(0000):/usr/lib/uucp:/bin/sh
who::7:0::/bin:who
date::8:0::/bin:date
```

```
shutdown:RE3SnRF/mNdeQ:21:1:0000-Admin(0000):/:/etc/shutdown
lp:**NO LOGIN**:71:2:0000-lp(0000):/usr/spool/lp:/bin/sh
barneoud:ht72sH2xnYTBc:101:100:Marie-Line:/users/barneoud:/bin/sh
bertin:dfAY65.t5P/oc:301:300:Christian Bertin:/ia/bertin:/bin/sh
vial:xh1WPxDXr9IjQ:303:300:Philippe Vial:/ia/vial:/bin/sh
```

UNIX offre la possibilité de rassembler les utilisateurs dans des groupes, par exemple un groupe peut être constitué des utilisateurs d'un même service, ou des utilisateurs qui travaillent sur un même projet. Cela permet de définir des droits d'accès identiques pour les fichiers communs à tous les utilisateurs d'un même groupe. Les numéros des groupes d'utilisateurs systèmes sont compris entre 0 et 100, et les numéros des autres groupes sont habituellement des multiples de 100 et les numéros des utilisateurs de ce groupe sont pris dans cette centaine. Comme les utilisateurs, les groupes sont définis dans un fichier : le fichier */etc/group*. Chaque ligne correspond à la description d'un groupe, elle est structurée de la manière suivante :

- le nom du groupe
- le mot de passe codé du groupe
- le numéro du groupe
- la liste des utilisateurs qui forment le groupe

Il est à noter qu'un utilisateur peut appartenir à plusieurs groupes, comme le montre cet exemple :

```
root::0:root
bin::2:root,bin,daemon,lp,lw
sys::3:root,bin,sys,adm
adm::4:root,adm,daemon
infa::100:barneoud
ia::300:bertin,elbaz,vial,debarbie
```

5 LES FICHIERS D'ADMINISTRATION D'UNIX

Pour terminer cette présentation des connaissances nécessaires pour pouvoir comprendre la manière dont UNIX fonctionne et assurer l'administration d'un système, nous allons maintenant étudier les principaux fichiers qui permettent de définir le mode de fonctionnement de ce système. Nous commencerons cet examen par les fichiers communs à tous les sites UNIX : les fichiers qui contrôlent son lancement, les connexions des utilisateurs, et quelques autres fichiers divers. A ces fichiers de base, viennent s'ajouter les fichiers d'administration propres aux applications installées sur la machine. Comme de plus en plus fréquemment, une configuration comprend plusieurs ordinateurs reliés par un réseau local, nous présenterons aussi les

fichiers d'administration des logiciels de communication sur un réseau Ethernet et sur un réseau X25.

5.1 LE CONTROLE DU LANCEMENT D'UNIX

Le fichier qui contrôle le lancement d'UNIX est le fichier */etc/inittab*, en effet il sert de base de données au processus *init* qui est le premier processus UNIX créé lors du démarrage de la machine. Ce processus lit le fichier */etc/inittab* et lance l'exécution des commandes contenues dans ce fichier. Chaque commande est accompagnée de deux paramètres. Comme UNIX peut posséder jusqu'à sept niveaux de fonctionnement différents numérotés de 0 à 6 en mode multi-utilisateur, le premier de ces paramètres précise pour quels niveaux la commande doit être exécutée. Le second paramètre définit, lui, le comportement de *init* vis à vis de la commande, c'est-à-dire comment et quand elle doit être exécutée.

Une des tâches les plus importantes de l'administrateur système est de définir les niveaux de fonctionnement (run levels). Traditionnellement on rencontre une solution tout ou rien :

- soit UNIX travaille en mode mono-utilisateur, alors les partitions de sont pas montées, le spoule est inactif...

- soit UNIX travaille en mode multi-utilisateur et tous les services sont disponibles. Le niveau 2 correspond habituellement à cette exploitation normale d'UNIX.

Mais nous pensons qu'il est intéressant de définir d'autres niveaux de fonctionnement, par exemple :

- le niveau 0 pour réaliser les opérations d'administration et de sauvegardes

- le niveau 3 identique au niveau 2 sans les serveurs des logiciels réseau, par conséquent seuls les utilisateurs locaux peuvent se connecter.

Généralement, on trouve successivement dans */etc/inittab* les appels aux commandes :

- */etc/machconf* dans lequel sont initialisés le nom de la machine, la partition système et la zone de swap

- */etc/bcheckrc* qui se charge de définir la date, et de vérifier la cohérence de la partition système et des systèmes de fichiers indiqués dans le fichier */etc/checklist*

- */etc/brc* qui réinitialise la table des partitions montées

- */etc/rc* qui effectue le montage des disques locaux comme défini dans le fichier */etc/mountlist* ou le fichier */etc/fstab* suivant les versions d'UNIX, détruit les fichiers temporaires et de suivi de l'activité du système, traces de la précédente période d'utilisation, et lance le gestionnaire de tâches de fond */etc/cron*.

- */etc/initcons* qui réalise l'initialisation de la console système, en créant un lien entre la console système virtuelle et une ligne asynchrone, l'administrateur peut ainsi déporter où il le désire la console système

- */etc/initlp* qui lance le spoule d'impression

- */etc/powerfail* que *init* exécute s'il reçoit le signal "powerfail".

Puis */etc/inittab* contient autant d'entrées que de lignes de communication (lignes asynchrones physiques ou lignes logiques PAD). Chaque entrée définit l'utilisation d'une voie, si cette voie peut être employée pour se connecter sur la machine, *init* lui associe un processus qui traite les demandes de connexion.

Enfin, en fonction des options installées et des services offerts aux utilisateurs, l'administrateur peut être amené à créer d'autres entrées dans */etc/inittab*, comme par exemple le lancement des serveurs des logiciels réseaux, des serveurs graphiques ou du serveur de la messagerie X400.

5.2 LE CONTROLE DE LA CONNEXION DES UTILISATEURS

Nous venons de voir les fichiers d'administration utilisés par UNIX lors du démarrage de l'ordinateur et de son passage en mode de fonctionnement multi-utilisateurs, examinons maintenant les fichiers qu'UNIX utilise pour connaître la façon dont il doit traiter les demandes de connexion d'un utilisateur. En plus du fichier */etc/inittab* ces fichiers sont au nombre de quatre :

- */etc/gettydefs* qui contient les paramètres déterminant la gestion pour chaque type de voie, la vitesse de transmission, le message d'appel de connexion, etc

- */etc/issue* dont le contenu est affiché sur le terminal avant le message d'appel de connexion

- */etc/motd* qui contient un message qui est affiché juste après la connexion

- */etc/profile*, un fichier de commandes, où sont définis certaines variables de l'environnement système de l'utilisateur, les droits par défaut des fichiers que créera l'utilisateur, et où sont lancés des programmes à exécuter avant que l'utilisateur ne commence à travailler. L'utilisateur peut par exemple être prévenu de l'arrivée de courrier dans sa boîte aux lettres.

5.3 DIVERS

De même que le démarrage d'UNIX est piloté par le fichier */etc/inittab*, toutes les actions nécessaires pour l'arrêter proprement sont regroupées dans le fichier de commandes */etc/shutdown.*

Si tous les fichiers que nous venons d'étudier se trouvent obligatoirement dans tous les systèmes UNIX, le guide de l'administrateur UNIX recommande à l'ingénieur système de créer un fichier, par exemple */etc/local*, pour conserver les noms des fichiers d'administration qui ont été modifiés lors de l'installation. Ainsi il saura parfaitement, lors de la mise en place de nouvelles versions de système ou de logiciels, quels fichiers corriger; et il peut même développer des procédures de sauvegarde et de restauration automatiques de ces fichiers.

Enfin, suivant la solution retenue par le constructeur pour le formatage des disques, nous pouvons trouver, comme sous SMX V.2, dans le répertoire */etc/format* les fichiers qui décrivent les modèles de formatage des disques connus du système.

5.4 LES RESEAUX

Comme les logiciels disponibles sous UNIX sont très nombreux, nous avons choisi comme exemples d'options demandant l'installation de fichiers d'administration supplémentaires les logiciels de communication sur les réseaux Ethernet et X25 parce qu'ils sont très répandus et font presque partie maintenant de la distribution standard UNIX.

5.4.1 Ethernet

Sur un réseau Ethernet, un utilisateur dispose, pour se connecter sur une machine distante, des utilitaires *telnet*, *rlogin*, pour exécuter une commande sur une machine distante, de l'utilitaire *rshell*, et, pour transférer des fichiers, des utilitaires *rcp*, *ftp*. Les utilitaires *telnet* et *ftp* sont les protocoles de connexion à distance et de transfert de fichiers de l'architecture de réseau Internet de l'Advanced Research Projets Agency (ARPA); ils permettent de communiquer avec tout système supportant cette architecture. En revanche, les utilitaires *rlogin*, *rcp*, *rshell* autorisent seulement les communications avec des machines où sont aussi installés les logiciels réseaux d'UNIX 4.2 BSD. Tous ces services exigent la présence pour fonctionner sur chaque machine du réseau des serveurs correspondants. Le lancement de ces serveurs est réalisé par le fichier de commandes */etc/rc.inet* pour lequel l'administrateur crée une entrée dans */etc/inittab*. Le fichier */etc/rc.inet* assure aussi l'initialisation du module d'échange Ethernet.

Le fichier */etc/networks* définit les sous-réseaux constituant le réseau auquel appartient notre système UNIX. Le fichier */etc/hosts* contient la liste des machines formant le réseau. Une entrée de ce fichier associe le nom d'une machine avec l'adresse Internet de cette machine et les synonymes possibles pour la désigner. Le fichier */etc/hosts.equiv* précise la liste des machines du réseau dont les utilisateurs peuvent se connecter sur la machine locale sans avoir à fournir leur mot de passe. Enfin, les fichiers */etc/protocols* et */etc/services* permettent de définir les différents protocoles de communication et services Internet offerts sur le réseau.

5.4.2 X25

L'initialisation d'un serveur de communication X25 est assurée par les trois fichiers de commandes suivants :

- */etc/rc.x25* qui initialise le module d'échange X25
- */etc/rc.pad* qui initialise la fonction PAD
- */etc/padlist* qui définit l'emploi des voies logiques PAD, c'est-à-dire si une voie est entrante ou sortante, et vers quel ordinateur.

Le plus souvent une entrée dans le fichier */etc/inittab* est créée pour chacun des fichiers */etc/rc.x25* et */etc/rc.pad*, le fichier */etc/padlist* est lui appelé depuis */etc/rc.pad*. Le routage des communications est contrôlé par le fichier */etc/L_route* qui définit en fonction des destinations le module d'échange et la voie à utiliser pour établir la connexion.

Nous arrêtons là la présentation des fichiers d'administration car notre but n'est pas de construire une liste exhaustive de tous les fichiers qu'il est possible de rencontrer sur un système UNIX, mais de donner aux lecteurs les bases indispensables pour pouvoir débuter l'installation d'UNIX. En particulier, nous avons choisi de ne pas parler de l'installation de *uucp* (UNIX to UNIX copy) bien que ce soit un des logiciels les plus délicats à mettre en place, car nous pensons qu'avec le développement de logiciels réseaux performants son utilisation deviendra de plus en plus rare. Nous allons maintenant étudier plus précisément l'installation d'UNIX et voir comment l'ingénieur système met en place les fichiers d'administration.

Chapitre 2

L'INSTALLATION D'UNIX SYSTEM V

1 INTRODUCTION

Les administrateurs d'un système informatique assurent deux fonctions très différentes, d'une part l'installation des matériels et des logiciels constituant le système et d'autre part la gestion journalière du système. En fait, nous pouvons encore distinguer deux sortes d'installations : la construction d'un système et l'intégration d'extensions tant logicielles que matérielles. Par gestion journalière, nous entendons toutes les opérations que doit réaliser l'administrateur pour garantir le bon fonctionnement et la pérennité du système, les principales sont :

- le redémarrage du système après un arrêt
- l'arrêt du système
- la réalisation de sauvegardes
- la maintenance du système après une défaillance
- la surveillance de l'activité du système.

Nous allons nous intéresser aux problèmes rencontrés lors de la première installation d'UNIX system V sur une machine. En effet, celle-ci regroupe les opérations nécessaires à l'installation de tous les composants de la configuration, nous pouvons donc la considérer comme étant constituée de plusieurs sous-tâches : les installations d'extensions. D'autre part la gestion journalière dépend fortement des décisions prises pendant l'installation et une partie de cette dernière peut consister à définir les procédures à mettre en oeuvre lors de la gestion.

Installer UNIX sur une machine est une tâche complexe dont l'objectif est de permettre à plusieurs utilisateurs de se connecter et d'exécuter des programmes sur cette machine. Cette tâche est d'autant plus complexe qu'UNIX a été conçu pour être portable sur de nombreux ordinateurs et doit donc s'adapter à un grand nombre de composants. De plus les différentes étapes d'une installation doivent être exécutées dans un ordre précis qui dépend de choix faits par les concepteurs des matériels et logiciels pour construire un système prêt à fonctionner.

Nous allons présenter premièrement les difficultés de l'installation d'UNIX, deuxièmement ses différentes étapes, et troisièmement quelques solutions possibles pour essayer de rendre cette installation réalisable par un non spécialiste.

2 LES DIFFICULTES DE L'INSTALLATION D'UNIX

Quand une machine UNIX arrive dans un service ou un laboratoire, l'ingénieur, qui deviendra l'administrateur du système, reçoit avec l'ordinateur le système UNIX lui-même, les différents logiciels complémentaires sur des supports magnétiques (bandes, cassettes, disquettes), et la très volumineuse documentation qui concerne aussi bien le matériel que le logiciel (guide administrateur, manuels de référence,... [ATT84] [BULL86]).

Alors commence une tâche longue et pénible : l'installation d'UNIX. Le responsable du système doit tout d'abord lire très soigneusement la documentation fournie pour déterminer les caractéristiques des matériels et des logiciels présents. Malheureusement ces informations ne sont pas toujours faciles à trouver, surtout dans la documentation UNIX où il existe de nombreux renvois indiqués dans les rubriques "SEE ALSO" et "FILES" des manuels de référence. De plus la quasi-totalité de la documentation est disponible seulement en anglais, et manque parfois de précision et laisse une grande liberté d'interprétation au lecteur.

Une fois les informations nécessaires à l'installation trouvées, l'ingénieur système novice peut débuter l'installation proprement dite, il peut alors choisir entre deux solutions, soit installer une des configurations standard fournies par le constructeur, soit installer manuellement le système UNIX pour gérer exactement les composants matériels et logiciels de sa machine.

La première solution est la plus simple, en effet l'utilisateur n'a qu'à lancer quelques commandes, et un système UNIX sera disponible. Mais il est vraisemblable que la configuration installée ne réponde pas totalement à ses besoins. La deuxième solution suppose une excellente connaissance du système UNIX. Les constructeurs fournissent bien des procédures d'installation, sous forme de menus, d'interfaces graphiques pour les présentations les plus agréables, ou sous forme de fichiers de commandes, mais ce ne sont le plus souvent que des encapsulations de commandes système qui n'apportent aucune aide à l'ingénieur système dans la prise de décisions. Ces connaissances ne peuvent s'acquérir que par la pratique, et le débutant ne peut trouver de l'aide qu'auprès d'administrateurs expérimentés. Car la documentation UNIX présente seulement les grands principes d'installation et la syntaxe des commandes; par exemple, elle explique comment découper un disque, mais pas pourquoi et comment déterminer les tailles des partitions.

De toute manière, même s'il choisit d'installer une configuration standard, l'ingénieur système risque de rencontrer des problèmes. En effet, après l'installation d'un découpage prédéfini des disques, il devra assurer l'ajustement fin de la configuration logicielle, à savoir définir les utilisateurs, leurs répertoires d'accueil, les droits d'accès des fichiers, mettre en place les fichiers d'administration des logiciels de communication sur un réseau local, qui

nécessite les compétence d'un ingénieur système confirmé.

Si les problèmes rencontrés lors de l'installation d'UNIX sur une machine isolée sont déjà non triviaux, l'installation d'un réseau local avec plusieurs serveurs disques les multiplie. En effet il faut alors répartir les fichiers des différentes machines du réseau entre les serveurs afin de répartir la charge de la façon la plus équilibrée possible. Comme les constructeurs ne proposent que des installations types avec un seul serveur, l'administrateur se retrouve sans aucune aide.

Nous allons présenter maintenant les différentes étapes qu'il convient d'observer dans une installation d'UNIX pour que tout se déroule sans problèmes.

3 LES ETAPES DE L'INSTALLATION D'UNIX

Pour essayer d'aider les administrateurs d'un système UNIX à l'installer, nous allons décrire une installation idéale où tout se passerait sans problèmes, ce qui est rarement le cas dans la réalité. La pratique et l'analyse de nombreuses installations d'UNIX nous a conduits à décomposer cette installation en 8 phases :

- l'inventaire de la configuration matérielle
- l'inventaire de la configuration logicielle
- la définition des partitions
- le premier boot
- la configuration des disques
- l'installation de l'environnement logiciel
- la génération d'un nouveau noyau
- le passage en multi-utilisateurs

Examinons maintenant chacune de ces étapes dans le détail.

3.1 L'INVENTAIRE DE LA CONFIGURATION MATERIELLE

Dès la réception du système, il faut dresser l'inventaire des matériels livrés. Cela permet tout d'abord de vérifier que la livraison correspond bien à la commande et que le matériel est en bon état et de se familiariser avec ce nouvel équipement. Ensuite l'ingénieur système doit étudier très soigneusement la documentation technique qui accompagne chaque élément du système, ainsi il peut acquérir les informations nécessaires à l'installation. Si, par exemple, la livraison comprend des disques non formatés et si UNIX ne connaît pas leur modèle de formatage, l'ingénieur système doit avant toutes choses explorer la documentation de ces disques et

de leurs contrôleurs pour connaître leurs paramètres caractéristiques et pouvoir, par la suite, les formater. Ce faisant, il détermine les potentialités du système et il va pouvoir commencer à affecter à chacune des ressources une utilisation. Ainsi, il est recommandé d'installer la partition système et la zone de swap sur des disques rapides, de répartir les partitions sur les disques pour obtenir une charge équilibrée et les meilleures performances. L'administrateur peut également, dès à présent, définir en fonction de leurs caractéristiques l'utilisation des voies asynchrones, c'est-à-dire réaliser les connexions avec les terminaux, les imprimantes séries, et d'autres machines (micro-ordinateurs)...

Comme les échanges de données entre l'unité centrale et les périphériques se font par l'intermédiaire de fichiers spéciaux, établir la liste des matériels présents permet donc de déterminer les fichiers spéciaux à créer. Pour ce faire, il faut réussir à trouver pour chaque périphérique les "formules magiques" pour calculer les majeurs et les mineurs des fichiers spéciaux (cf chapitre précédent).

3.2 L'INVENTAIRE DE LA CONFIGURATION LOGICIELLE

Après l'inventaire du matériel constituant le système, il faut dresser l'inventaire des logiciels qui sont à installer sur ce système. Comme pour le matériel, l'ingénieur système doit acquérir, pour chaque logiciel, les renseignements nécessaires à leur installation, à savoir :

- l'espace occupé sur disque par ce logiciel
- la taille mémoire nécessaire à son exécution
- le répertoire où il doit être installé
- ses utilisateurs
- les équipements nécessaires à son fonctionnement. Cette dernière information permet de vérifier par confrontation avec la liste du matériel disponible qu'il n'existe pas d'impossibilité d'exécution à cause de l'absence d'un élément matériel ou de la pénurie d'une ressource (taille mémoire insuffisante par exemple).
- les logiciels nécessaires à son fonctionnement. En effet certains logiciels exigent la présence d'autres logiciels pour pouvoir être exécutés. Par exemple, si LE_LISP n'est pas installé, il est vain de chercher à installer le générateur de systèmes experts KOOL de BULL, il ne pourra pas fonctionner.
- les fichiers nécessaires à son fonctionnement. Ces fichiers peuvent être des fichiers de données, des fichiers d'administration ou des fichiers spéciaux comme ceux des pseudo-terminaux des logiciels réseaux.

Cet inventaire permet donc d'une part de vérifier qu'on dispose d'un ensemble matériel et logiciel cohérent, et d'autre part de déterminer la taille qu'occupera le système, la zone de swap, et par conséquent l'espace restant disponible pour les utilisateurs. Après cet inventaire, l'administrateur connaît également les logiciels que chaque utilisateur employera, il peut donc constituer les groupes d'utilisateurs en fonction des intérêts principaux de ceux-ci, et définir leurs privilèges (droits d'accès). Enfin, cet inventaire lui aura permis de déterminer l'ordre d'installation des logiciels en fonction des logiciels indispensables à leur exécution et les fichiers à mettre en place pour qu'ils puissent s'exécuter correctement.

A l'issue des inventaires des matériels et des logiciels, l'ingénieur système possède toutes les informations nécessaires pour déterminer la paramétrisation du système à installer.

3.3 LA DEFINITION DES PARTITIONS A CREER SUR LES DISQUES

La troisième étape de l'installation UNIX system V sur un ordinateur consiste à définir le découpage des disques durs ou fixes en partitions. Cette étape est inutile pour de certaines versions d'UNIX qui gèrent les disques comme des zones de stockage indivisibles.

Le découpage des disques est déterminé tout d'abord par le nombre de disques disponibles, leur taille, leurs performances, et ensuite la taille du système, le volume des applications et le nombre d'utilisateurs. Pour fonctionner un système UNIX nécessite au moins deux partitions, une partition contenant le système lui-même, le plus souvent appelée "root", et une partition pour la zone de swap.

3.3.1 La partition système

Déterminer le contenu et la taille de la partition système est une opération difficile. La partition système contient toujours l'exécutable du noyau UNIX, et les répertoires */bin*, */etc*, */dev*, */lib*, et dans un répertoire dont le nom varie d'un système à un autre les données néccessaires à la génération du noyau. Pour le répertoire */usr*, il existe deux écoles :

- la première l'installe dans la partition système
- la seconde lui affecte un système de fichiers particulier

Pour notre part, nous estimons que le répertoire */usr* doit aussi appartenir à cette partition car les commandes UNIX qui ne sont pas dans le répertoire */bin* se trouve dans le répertoire */usr/bin*. Par principe, nous pensons que toutes les commandes UNIX doivent être regroupées sur la partition système pour qu'elles puissent être accessibles dès le boot. Il est, par exemple, très agréable de pouvoir utiliser l'éditeur plein écran */usr/bin/vi* sans avoir à se soucier de monter une partition. De plus certains de ses sous-répertoires contiennent des utilitaires qui sont devenus des commandes standard comme *uucp*, les logiciels réseaux ou la messagerie X400, et leurs fichiers d'administration auxquels l'ingénieur système doit pouvoir accéder dès le lancement de la machine en mode mono-utilisateur.

Une fois la taille de la partition système déterminée, sa création peut parfois être délicate. En effet sur certaines machines, il est **impossible** de la définir avant de copier le système de distribution; nous sommes confrontés à ce problème lorsque nous installons SPIX sur un SPS7 ou un SPS9. Dans ce cas, il faut construire une partition système de la taille souhaitée sur un autre disque, la copier sur un support magnétique puis reprendre l'installation à partir de ce support. Si on dispose de plusieurs disques et de suffisamment d'espace, on peut prévoir la création d'un deuxième disque système pour permettre de redémarrer immédiatement en cas de problème sur la partition système usuelle.

3.3.2 Les autres partitions

Il est recommandé de créer des disques virtuels pour les répertoires */tmp*, */usr/tmp*, et */usr/spool*. En effet UNIX crée les fichiers temporaires dont il a besoin dans ces répertoires, aussi constituent-ils un point sensible de ce système et en leur associant une partition les risques de problèmes sont réduits. Il nous semble également judicieux de créer un disque virtuel pour le répertoire */usr/local* où l'on installera les différentes applications, ainsi elle seront nettement séparées des fonctions de base du système. Enfin la place restant disponible, après la définition de ces partitions, sera utilisée pour les partitions destinées à recevoir les données des utilisateurs, et si possible pour des partitions de manoeuvre ou de sauvegarde.

Quant au tailles des partitions, il est bon d'observer les règles suivantes :

- la taille de la partition correspondant à */tmp* peut être comprise entre 1500 ko et 2500 ko selon le nombre maximum d'utilisateurs actifs simultanément, et selon les programmes principalement utilisés.

- la taille de la zone de swap est habituellement de 2 à 3 fois la taille de la mémoire centrale. Toutefois, suivant le nombre d'utilisateurs et la taille des exécutables l'administrateur peut réduire ou au contraire augmenter cette taille.

- la taille des autres partitions doit être au minimum de 3000 ko. Il est en effet inutile de créer des partitions trop petites qui risquent d'être rapidement saturées car alors UNIX passera beaucoup de temps à chercher de la place libre dans ces petits systèmes de fichiers.

Ces tailles correspondent à une utilisation d'un système UNIX qualifiée de normale. Bien entendu, comme chaque installation est un cas particuler, l'administrateur déterminera, en s'inspirant des ces conseils, à chaque fois les partitions et les tailles qu'il estimera les mieux adaptées à cette configuration.

3.4 LE PREMIER BOOT DEPUIS LE SUPPORT DE DISTRIBUTION

Une fois les inventaires matériels et logiciels effectués et le découpage des disques défini, l'ingénieur système peut véritablement débuter l'installation d'UNIX. Pour ce faire, il doit démarrer sa machine avec le système qu'il a reçu sur un support magnétique mobile, et copier ce système de distribution sur le disque qu'il a choisi comme disque système. Cette copie terminée, l'ingénieur système redémarre la machine depuis le disque système qu'il vient de définir, et il peut maintenant entreprendre la configuration du système pour qu'il prenne en compte tous les composants présents. Il commence par la configuration des disques.

3.5 LA CONFIGURATION DES DISQUES

Le but de cette configuration est de rendre les disques et les partitions accessibles. Elle exige l'exécution dans un ordre très strict des opérations suivantes :

- la création des fichiers spéciaux de type caractère et bloc des disques physiques, de liens sur ces fichiers pour leur donner des noms plus expressifs que les noms prédéfinis, et la définition des droits d'accès de ces fichiers, par exemple pour le disque 302 :

```
/etc/mknod /dev/dsk/c3d2 b 0 98
/etc/mknod /dev/rdsk/c3d2 c 4 98
/bin/ln /dev/dsk/c3d2 /dev/pd302
/bin/ln /dev/rdsk/c3d2 /dev/rpd302
/bin/chmod 660 /dev/pd302
/bin/chmod 660 /dev/rpd302
```

- le formatage des disques si nécessaire. Nous avons supposé que le disque système était formaté, pour les autres il faut vérifier si le formatage a déjà été réalisé en usine par le constructeur et sinon l'effectuer maintenant pour pouvoir poursuivre l'installation. Le formatage est une opération qui dépend fortement des choix des constructeurs lors de la conception de la machine, aussi n'existe-t-il pas une commande standard de formatage UNIX.

- la création des fichiers spéciaux des partitions, et de liens sur ces fichiers spéciaux. Si l'on désire installer la partition système, la zone de swap et une partition utilisateur sur le disque 302, on exécutera les commandes :

```
/etc/mknod /dev/dsk/c3d2s0 b 0 128
/etc/mknod /dev/dsk/c3d2s1 b 0 129
/etc/mknod /dev/dsk/c3d2s2 b 0 130
/etc/mknod /dev/rdsk/c3d2s0 c 4 128
/etc/mknod /dev/rdsk/c3d2s1 c 4 129
/etc/mknod /dev/rdsk/c3d2s2 c 4 130
/bin/ln /dev/dsk/c3d2s0 /dev/root
```

```
/bin/ln /dev/rdsk/c3d2s0 /dev/rroot
/bin/ln /dev/dsk/c3d2s1 /dev/swap
/bin/ln /dev/rdsk/c3d2s1 /dev/rswap
/bin/ln /dev/dsk/c3d2s2 /dev/vdusers
/bin/ln /dev/rdsk/c3d2s2 /dev/rvdusers
```

et pour restreindre l'accès à ces fichiers à *root* et à son groupe :

```
/bin/chmod 660 /dev/root
/bin/chmod 660 /dev/rroot
/bin/chmod 660 /dev/swap
/bin/chmod 660 /dev/rswap
/bin/chmod 660 /dev/vdusers
/bin/chmod 660 /dev/rvdusers
```

- le découpage des disques en partitions. Comme pour le formatage des disques, chaque constructeur propose sa méthode de partitionnement des disques. Certains UNIX gèrent les disques comme des zones indivisibles, d'autres proposent des découpages prédéfinis et d'autres enfin laissent l'administrateur libre de fixer le découpage qui convient le mieux à l'utilisation prévue de la machine. Dans ce cas, le découpage est écrit sur une zone reservée du disque. Par exemple, sous SMX V.2 il se fait en deux temps. Tout d'abord il faut écrire le partitionnement d'un disque physique sur son dernier bloc, puis il faut associer les disques virtuels avec les partitions physiques qui viennent d'être créées. Cette denière opération doit aussi être effectuée lors de chaque démarrage de la machine, généralement elle est réalisée par le fichier */etc/machconf*. Si nous reprenons l'exemple précédent et si nous désirons une partition système de 81270 blocs, une zone de swap de 16000 blocs et une partition utilisateur de 21296 blocs, on écrira le découpage du disque par :

```
/etc/mkpar /dev/rpd302 0 81270 81270 16000 97270 21296
```

et on associera les disques virtuels aux partitions physiques par :

```
/etc/vdset /dev/rpd302 128
```

- la création des systèmes de fichiers sur les partitions. Dans notre exemple, seules les partitions système et utilisateur supportent des systèmes de fichiers. Si le système de fichiers de la partition système est automatiquement généré lors de la copie du système de distribution sur le disque, il restera à créer celui de la partition utilisateur par :

```
/etc/mkfs /dev/rdsk/c3d2s2 21296
```

- l'étiquetage des systèmes de fichiers. Chaque système de fichiers possède deux

étiquettes, la première identifie le système de fichiers lui-même, et la seconde le disque physique auquel il appartient. Ces étiquettes sont utilisées par certaines procédures de sauvegarde et de vérification de la cohérence des systèmes de fichiers. Pour notre exemple :

```
/etc/labelit /dev/rdsk/c3d2s0 system 302
/etc/labelit /dev/rdsk/c3d2s2 users 302
```

- la création des répertoires de montage des partitions. Si nous décidons de monter la partition utilisateur sur le répertoire */users*, il faudra le créer avec la commande :

```
/bin/mkdir /users
```

- le montage des partitions. Le montage de la partition utilisateur sera réalisé par :

```
/etc/mount /dev/vdusers /users
```

- la création dans chaque système de fichiers d'un répertoire lost+found qui accueille, lors de la vérification de la cohérence du système de fichiers par */etc/fsck*, les fichiers dont les inodes ne sont plus associés à un nom externe. Pour la partition utilisateur, cette création se fera par :

```
cd /users
/etc/mklost+found
```

- la mise à jour des fichiers d'administration */etc/machconf* où sont définis, entre autres choses, le disque système, la zone de swap et l'association des disques virtuels avec les disques physiques, */etc/mountlist* qui contient la liste des partitions à monter automatiquement lors du lancement de la machine, et /etc/checklist qui contient la liste des systèmes de fichiers dont on souhaite vérifier la cohérence lors du démarrage de la machine.

- et éventuellement l'écriture de procédures de sauvegarde pour les automatiser.

Ce travail accompli, les disques sont prêts à recevoir les logiciels d'application que l'administrateur désire installer.

3.6 L'INSTALLATION DE L'ENVIRONNEMENT LOGICIEL

L'installation d'UNIX se poursuit par la création des fichiers spéciaux nécessaires au fonctionnement du système autres que ceux des disques physiques et virtuels. Ces fichiers ont été déterminés lors de l'inventaire des matériels et des logiciels. Pour chaque fichier, l'ingénieur système calcule le majeur et le mineur, et définit les droits d'accès en fonction des futurs

utilisateurs ou des utilisations prévues (imprimante, connexion à un autre ordinateur). Par exemple pour la ligne asynchrone tty12 :

```
/etc/mknod /dev/al/cll2 c 1 18
/bin/ln /dev/al/cll2 /dev/tty12
/bin/chmod 622 /dev/tty12
```

Ensuite l'administrateur système peut commencer à personnaliser les fichiers d'administration. Généralement, les fichiers */etc/bcheckrc*, */etc/brc*, */etc/initcons* et */etc/powerfail* ne sont pas modifiés. Par contre, le fichier */etc/rc* est à mettre à jour en fonction des actions nécessaires pour monter les disques virtuels, pour effacer les fichiers temporaires créés pendant la période de fonctionnement précédente, pour lancer le serveur de traitements différés, ou le contrôle de l'activité du système. Le fichier */etc/rc* du système de distribution se présente comme un texte où la plupart des commandes à réaliser à ce moment du lancement du système sont en commentaires. Pour activer les commandes qui seront exécutées lors des démarrages de la machine, l'ingénieur système édite le fichier */etc/rc* et supprime simplement les commentaires devant ces commandes.

Puis, l'ingénieur système rajoute dans le fichier */etc/inittab* autant d'entrées que la machine comprend de lignes de communication. Les entrées qui correspondent à des voies où le système attend une connexion sont placées à *respawn* pour permettre au processus *init* de recréer après chaque fin de travail un processus */etc/getty* qui scrute la ligne dans l'attente de la prochaine connexion. Les autres entrées sont mises à *off*. Par exemple, si un terminal de type *tvi925* transmettant des données à 9600 bauds est connecté sur la ligne tty12, l'ingénieur système introduit la ligne suivante dans */etc/inittab* :

```
12:2:respawn:/etc/getty -Ttvi925 tty12 9600
```

Enfin, selon les imprimantes connectées au système, il modifie le fichier */etc/initlp* qui contrôle le lancement du spoule.

L'administrateur peut maintenant définir les utilisateurs qui auront accès au système, d'une part en mettant à jour le fichier */etc/passwd* et d'autre part le fichier */etc/group*. Pour pouvoir travailler, un utilisateur a besoin d'un répertoire d'accueil où il pourra installer ses fichiers. L'ingénieur système crée donc pour chaque utilisateur un répertoire, et surtout doit définir très précisément les droits d'accès sur ce répertoire. En effet ce sont ces droits d'accès qui assurent la protection des fichiers de l'utilisateur contre les autres utilisateurs. Si un utilisateur mal-intentionné possède le droit d'écrire dans un fichier et dans le répertoire de ce fichier il peut parfaitement modifier ou même détruire ce fichier. Les droits de chaque utilisateur doivent donc être fixés avec le plus grand soin pour éviter ce genre de problème.

Ainsi, définir strictement les droit d'accès d'un utilisateur sur les fichiers spéciaux, les fichiers d'administration et les fichiers des autres utilisateurs constitue la première étape indispensable à la protection du système contre les attaques extérieures. Ensuite, le lancement des utilitaires de surveillance de l'activité du système, même s'ils réduisent les performamces, peut permettre de détecter les intrusions puis de prendre les mesures nécessaires pour les rendre plus difficiles. Enfin, certains UNIX comme UTX/32S[1] de GOULD disposent de mécanismes supplémentaires pour renforcer le contrôle de l'accès au système.

Après cette première mise à jour des fichiers d'administration et la définition des utilisateurs, l'administrateur peut installer sur le système les différents logiciels optionnels prévus. L'installation d'un logiciel supplémentaire est le plus souvent contrôlé par un programme d'installation qui se charge de copier le logiciel depuis un support de distribution sur un disque, puis de lancer les programmes de configuration. L'ingénieur peut aussi, pour terminer cette installation, avoir à mettre à jour certains fichiers d'administration existants ou à en créer de nouveaux particuliers à ce logiciel. Par exemple, si la machine à installer s'intègre dans un réseau Ethernet et un réseau X25, il actualise les fichiers */etc/rc.inet* et */etc/rc.x25* en fonction des services qui seront offerts aux utilisateurs, il crée les entrées correspondantes dans le fichier */etc/inittab* et il crée les fichiers spéciaux des pseudo-terminaux (*/dev/ttypX* et */dev/ptypX*) et des lignes synchrones (*/dev/ost0* et */dev/ost1*). De plus si l'utilisation de voies logiques PAD sur ces réseaux est prévue, il met aussi à jour les fichiers */etc/rc.pad* et */etc/padlist*, il crée les fichiers spéciaux de ces voies logiques (*/dev/padX*) et définit aussi le fichier */etc/L_route* contenant la table de routage.

Pour finir l'installation de l'environnement logiciel du système, il reste à l'administrateur à adapter le fichier */etc/shutdown* à la configuration qui vient d'être installée et à créer un fichier */etc/local* où sera conservée la liste des fichiers d'administration propre au système.

3.7 LA GENERATION D'UN NOUVEAU NOYAU

Le noyau UNIX fourni dans la livraison par le constructeur est généralement un noyau qui inclut les pilotes indispensables seulement au fonctionnement d'une configuration de base et dont les paramètres du système reçoivent une valeur moyenne. Ce noyau correspond aux configurations les plus fréquentes. Par conséquent s'il répond aux besoins du système à installer, il est inutile de générer un nouveau noyau. Quand le système possède des périphériques autres que ceux dont les pilotes sont intégrés au noyau standard, ou quand les valeurs par défaut de certains paramètres ne conviennent pas à l'utilisation qui va être faite de la machine, pour obtenir un système opérationnel, l'administrateur doit générer un nouveau noyau UNIX.

---

1 UTX/32S est une marque déposée de GOULD INC.

Toutes les données nécessaires à la génération d'un noyau sont regroupées dans un seul fichier : le fichier */sys/cf/config.h*. Pour modifier la valeur d'un paramètre du système il suffit d'éditer ce fichier et de donner la nouvelle valeur. Pour ajouter un nouveau pilote, après son développement on doit seulement créer une nouvelle entrée dans le fichier */sys/cf/config.h*. Ensuite, il ne reste plus qu'à compiler le nouveau noyau et à remplacer l'ancien par celui que l'on vient d'obtenir. Mais pour que ce nouveau noyau soit pris en compte, il faut relancer la machine.

3.8 LE PASSAGE EN MULTI-UTILISATEURS

L'installation d'UNIX est maintenant terminée; pour que les utilisateurs puissent commencer à travailler, la dernière opération consiste à redémarrer le système et à passer en mode multi-utilisateurs. En effet UNIX dispose de deux modes de fonctionnement différents, le mode superviseur ou mono-utilisateur où seule la console système est active, toutes les opérations d'installation et de maintenance sont effectuées dans ce mode, et le mode multi-utilisateurs qui constitue le mode de fonctionnement normal d'UNIX.

4 LES SOLUTIONS POUR AUTOMATISER L'INSTALLATION D'UNIX

Comme nous venons de le voir, l'installation d'UNIX est une tâche longue et complexe qui exige des connaissances très vastes et une expérience certaine. La personne qui se charge de cette installation doit en effet posséder une connaissance approfondie des caractéristiques des matériels constituant la configuration et des caractéristiques du système UNIX. Mais plus encore que les connaissances brutes, l'installateur doit parfaitement connaître les principes d'installation, c'est-à-dire comment utiliser ces connaissances pour configurer le système, et il doit notamment savoir consulter la très volumineuse documentation d'UNIX. Or vu la grande diffusion que connaît actuellement UNIX en dehors des structures informatiques traditionnelles, de nombreux utilisateurs non informaticiens sont ou seront confrontés aux difficultés de l'installation d'UNIX. Un des objectifs de ce travail est de fournir à ces personnes un logiciel possédant une expertise suffisante pour leur éviter d'avoir à se transformer en ingénieurs système.

Le principal obstacle au développement d'un logiciel d'aide à l'installation est lié au fait qu'il est difficile d'exécuter un logiciel complexe sur un ordinateur avant la mise en place de son système d'exploitation. Voyons quelles sont les différentes solutions possibles pour aider un utilisateur à installer UNIX. Les solutions que nous allons proposer sont toutes des solutions dont le but est d'installer uniquement ce qui est nécessaire pour adapter UNIX à une configuration donnée. Cette approche nous distingue de certains fournisseurs qui livrent des systèmes où toutes les options possibles sont déjà intégrées. En procédant de cette manière, ces fournisseurs réduisent les problèmes d'installation, mais le système risque de comporter de nombreux éléments inutiles qui peuvent occuper une place disque importante et qui noyent les quelques informations significatives dans une masse sans intérêt. Pourquoi conserver l'ensemble des bibliothèques graphiques si les utilisateurs ne vont pas les employer, ou les

256 fichiers spéciaux qui décrivent les 16 partitions des 8 disques que le système est capable de gérer si la configuration dispose d'un seul disque découpé en 7 partitions.

L'assistance à l'installation d'UNIX peut s'envisager de trois façons :

- le logiciel est implanté sur une autre machine que la machine à installer, par exemple chez le constructeur ou le fournisseur, et à partir de la description de la configuration il génère le plan d'actions pour réaliser l'installation, et l'utilisateur en suivant ce plan configure son système [Harris86].

- l'installation est découpée en trois phases successives :

    1. comme dans la solution précédente, le logiciel est implanté sur une machine dédiée, mais cette fois il construit les fichiers d'installation et les fichiers d'administration destinés à la machine à installer

    2. puis sur cette première macchine, on crée la partition système destinée à la machine à installer et on la copie sur un support magnétique de distribution

    3. enfin, à partir du support de distribution on copie la partition système sur un disque de la machine à installer et l'installation se termine par l'exécution d' une commande unique qui met en place les options logicielles retenues. Une fois cette commande terminée, le système UNIX est prêt à être démarré en mode multi-utilisateur.

- le logiciel est lancé automatiquement lors du boot sur le support magnétique de distribution, et dialogue avec l'utilisateur pour obtenir les informations dont il a besoin pour définir la configuration. Puis il réalise l'installation en tenant compte des réactions du système lors de l'exécution des commandes, de cette façon, si une erreur se produit, il peut réagir pour supprimer les causes de l'erreur.

Quels sont les avantages et les inconvénients de chaque solutions? Les deux premières exigent la disponibilité d'une autre machine pour exécuter le logiciel d'installation. De ce fait la configuration souhaitée pour le système doit être prévue bien avant l'installation, or un utilisateur connaissant peu UNIX risque d'être incapable de la concevoir seul.

Le principal inconvénient de la première solution est que le bon déroulement de l'installation suppose que l'utilisateur ne commette aucune erreur. En effet, si ce dernier se trompe (fautes de frappe, commandes exécutées au mauvais moment), il risque fort de ne pas savoir comment réagir et de perdre beaucoup de temps en reprenant systématiquement l'installation depuis le début. Par contre, si un outil d'aide de ce type se contente de générer la liste des actions à entreprendre pour installer UNIX sans indiquer exactement les commandes à exécuter, il reste assez général pour s'appliquer à différents matériels et différents UNIX mais n'apporte, de ce fait, qu'une aide limitée à l'ingénieur système.

La deuxième solution en prenant totalement en charge l'installation du système élimine les erreurs précédentes de l'utilisateur. Mais l'utilisateur se trouve désarmé si la configuration a été improprement décrite, car le logiciel génère alors un système UNIX qui ne correspond pas exactement à la configuration et de ce fait ne peut pas fonctionner correctement. Et comme le logiciel de génération du système s'exécute sur une machine différente de la machine à installer, il ne peut pas détecter ce type d'erreurs.

En revanche, avec la troisième solution, ces problèmes sont évités en obtenant le plus possible d'informations en testant les composants présents dans l'ordinateur et en essayant de contrôler si possible l'exactitude des informations fournies par l'utilisateur. Atteindre cette troisième solution suppose d'exécuter, lors du lancement de la machine, le logiciel d'installation. C'est possible, mais pour ce faire il reste quelques problèmes techniques à résoudre. En effet, si ce logiciel d'installation est réalisé sous forme de système expert, il faudra être capable soit de charger lors du boot un programme très volumineux soit de "compiler" le système expert sous une forme exécutable plus compacte. Or pour exécuter les actions nécessaires à l'installation, le programme lancé lors du boot doit aussi inclure de nombreux utilitaires pour assurer le formatage, le découpage des disques, la création des systèmes de fichiers... ainsi sa taille risque de devenir énorme. Ensuite, à un moment qui reste à déterminer, il faudra copier UNIX sur disque, le lancer avant de pouvoir poursuivre l'installation par les actions qui ne peuvent s'exécuter que sous UNIX.

Nous avons retenu la seconde, car la première aurait laissé l'utilisateur impuissant devant la totalité des erreurs possibles, et la troisième aurait demandé un effort trop important de réalisation. De plus ce choix nous permet de nous concentrer sur les problèmes de l'installation et sur la représentation des connaissances d'un administrateur UNIX expert. En fait, pour nous rapprocher de la troisième solution, nous pouvons envisager d'utiliser un disque transportable qui contiendrait le logiciel d'installation et qui serait connecté à la machine à installer seulement pendant l'installation.

DEUXIEME PARTIE
LES SYSTEMES EXPERTS

Chapitre 3

LES SYSTEMES EXPERTS

1 INTRODUCTION

Comme nous avons présenté UNIX et les problèmes de son installation, nous allons préciser les objectifs et l'architecture d'un système expert ce qui nous conduira à examiner plus précisément les problèmes de représentation des connaissances et le fonctionnement d'un moteur d'inférences.

1.1 LES OBJECTIFS D'UN SYSTEME EXPERT

A l'opposé des programmes informatiques traditionnels (calcul scientifique, comptabilité, ...) qui assurent l'automatisation de calculs, de traitements répétitifs modélisables sous forme d'algorithmes, un système expert cherche à simuler des activités intellectuelles dans un domaine (diagnostic médical, diagnostic de pannes, décision financière, configuration d'un système, ...) où les connaissances sont volumineuses, éparses, évolutives et insuffisamment structurées pour qu'elles puissent être manipulées par des algorithmes sûrs et efficaces.

Un expert dans un de ces domaines est un homme qui possède un savoir-faire reconnu constitué d'un ensemble de granules de connaissances, de règles et d'heuristiques. Chaque règle s'applique à une classe particulière de situations et exploite seulement quelques granules ou fragments de la connaissance. Une règle apparaît donc comme une étape élémentaire du raisonnement de l'expert. Naturellement l'ensemble des règles évolue perpétuellement avec l'apparition de nouvelles connaissances et avec l'expérience de l'expert. Par conséquent, puisqu'un système expert tente de reproduire les facultés de décision, de jugement de l'expert humain à partir de connaissances obtenues de manière incrémentale et révisable, les principaux objectifs à atteindre lors du développement et la maintenance d'un système expert sont de :

- capturer les unités de connaissances et de raisonnement des experts de la façon la plus élégante, la plus efficace et la plus conforme à leur expression chez les experts

- exploiter convenablement ces connaissances pour résoudre un problème donné en déduisant de nouvelles connaissances et en expliquant comment elles sont obtenues

- supporter facilement la modification des connaissances.

1.2 L'ARCHITECTURE D'UN SYSTEME EXPERT

Pour réussir à atteindre ces trois objectifs, un système expert doit comporter les quatre composants suivants :

- un langage de représentation des connaissances
- une base de connaissances
- un moteur d'inférences
- un module de dialogue et d'explication

1.2.1 Le langage de représentation des connaissances

Ce langage doit être adapté au domaine étudié et compréhensible par les experts de ce domaine. Il doit aussi permettre d'exprimer les connaissances indépendamment les unes des autres, et indépendamment du moteur d'inférences pour garantir la possibilité de modifier aisément le système.

Souvent, ce langage cherche à se rapprocher le plus possible du langage naturel (langage cependant restreint aux aspects spécifiques du domaine d'expertise) car c'est ainsi qu'un expert humain formule habituellement ses connaissances. Mais le langage naturel n'est peut-être pas toujours la représentation la plus efficace car il apparaît alors des problèmes d'ambiguïté, de contextes, de sémantique qui peuvent rendre la modélisation des connaissances plus difficile. En effet, la signification de granules de connaissances exprimées en langage naturel comme *la température est trop élevée* dépend d'autres informations caractéristiques du domaine étudié (réacteur chimique, canon à neige, climatologie).

De même, l'obtention d'une représentation interne utilisable par la machine du fait *Jean attrape une balle* impose la levée de plusieurs ambiguïtés [Charniak85]. De fait, il faut pouvoir identifier de façon unique *Jean*, la *balle* et préciser dans quel sens le verbe *attraper* est utilisé. En effet, la représentation interne des connaissances ne sera exploitable que si toutes les ambiguïtés référentielles sont éliminées. Or la suppression de toutes les ambiguïtés ne peut être réalisée que par la modélisation de l'ensemble des connaissances qui forment l'environnement du problème à résoudre. Ainsi l'utilisation d'un langage à la fois plus pauvre et plus strict que le langage naturel, grâce à l'absence de toute référence implicite, peut favoriser la représentation des connaissances parce q'elle oblige l'expert à mieux formaliser l'expression ses connaissances. Par exemple, une représentation totalement non ambiguë du fait *Jean attrape une balle* peut être :

```
(inst jean_1 personne)
(inst balle_3 balle)
(attrape_objet jean_1 balle_3)
```

La première formule affirme que *jean_1* est une personne, la deuxième que *balle_3* est une

balle et la troisième que *jean_1 attrape la balle_3*.

Le modèle de représentation le plus employé est celui des règles de production. Une règle de production se présente sous la forme générale suivante :

```
SI condition_1
ET condition_2
ET ...
ALORS conclusion_1
ET conclusion_2
ET ...
```

La nature des conditions et des conclusions est très diverse suivant le type de connaissances à représenter. Ce peut être des variables propositionnelles, des prédicats, des fonctions LISP ou des accès à l'attribut d'un objet. Le succès de ce modèle de représentation s'explique, d'une part, par une bonne formalisation de certains raisonnements humains par des règles de type prémisse-conclusion, et d'autre part, par la modularité, la facilité de manipulation et de compréhension des règles [Vignard86].

1.2.2 La base de connaissances

A un instant donné, la base de connaissances représente l'ensemble des connaissances connues du système expert. Pour qu'elle soit aisément modifiable, la base de connaissances est isolée des autres composants du système et chaque connaissance de la base est représentée indépendamment des autres connaissances. On distingue deux types de connaissances :

- les connaissances factuelles qui décrivent :
    - les faits établis ou assertions
    - les faits à démontrer ou buts
- les connaissances opératoires qui représentent les connaissances obtenues auprès de l'expert exprimées dans le langage de représentation des connaissances. Autrement dit elles traduisent le savoir et le savoir-faire de l'expert du domaine indépendamment d'un problème particulier en indiquant quelles conséquences tirer ou quelles actions accomplir lorsque telle situation est établie ou à établir. Ce sont les règles.

On peut aussi parler de connaissances à court terme (cas étudié) et de connaissances à long terme (expertise).

1.2.3 Le moteur d'inférences

C'est un programme de mise en relation des connaissances factuelles et des connaissances opératoires pour produire (inférer, déduire, démontrer, ...) de nouveaux faits et éventuellement de nouvelles règles.

Ce composant du système est progammé selon un algorithme prédéfini dont le cycle de travail comprend deux temps :

- l'évaluation, c'est-à-dire la sélection de la règle à activer. Elle est réalisée en comparant la partie déclencheur de chaque règle au contenu de la base de faits.

- l'exécution des actions de la règle déclenchée.

1.2.4 Le module de dialogue et d'explication

Ce module réalise les communications entre le système déductif et l'extérieur. Il existe trois sortes d'interfaces :

- une pour les développeurs qui offre des outils pour mettre à jour la base de connaissances, pour pister le déroulement d'une consultation du système expert

- une pour les élèves qui offre des outils qui permettent seulement de consulter la base de connaissances

- une pour les utilisateurs qui offre des outils pour assurer la saisie des faits initiaux, le dialogue avec le système déductif et pour expliquer les décisions du système expert.

Dans l'interface utilisateur, la fonction d'explication est fondamentale, car le système expert doit justifier toutes ses actions en terme de connaissances de l'expert pour convaincre l'utilisateur de leur justesse. Généralement les systèmes experts possèdent deux types d'explications :

- le pourquoi qui justifie l'emploi de la dernière règle utilisée

- le comment qui retrace tout l'enchaînement de la démonstration

Les explications doivent être fournies dans un langage proche du langage naturel car l'utilisateur n'a pas à connaître le formalisme de représentation des connaissances utilisé pour développer la base de connaissances.

En résumé, nous pouvons retenir la spécification logique suivante :

> un système expert est constitué d'un système cognitif organisé autour d'un moteur d'inférences qui exploite une collection de connaissances factuelles et opératoires indépendantes et évolutives pour la résolution de problèmes dans un domaine d'expertise particulier.

Schématiquement un système expert peut être représenté de la manière décrite dans la figure 1.1.

Des quatre composants d'un système expert, le langage de représentation des connaissances et le moteur d'inférences sont les plus importants. En effet, ce sont eux qui conditionnent le développement de la base de connaissances. Nous allons donc étudier les différents modèles de représentation des connaissances et les différents fonctionnements possibles pour un moteur d'inférences.

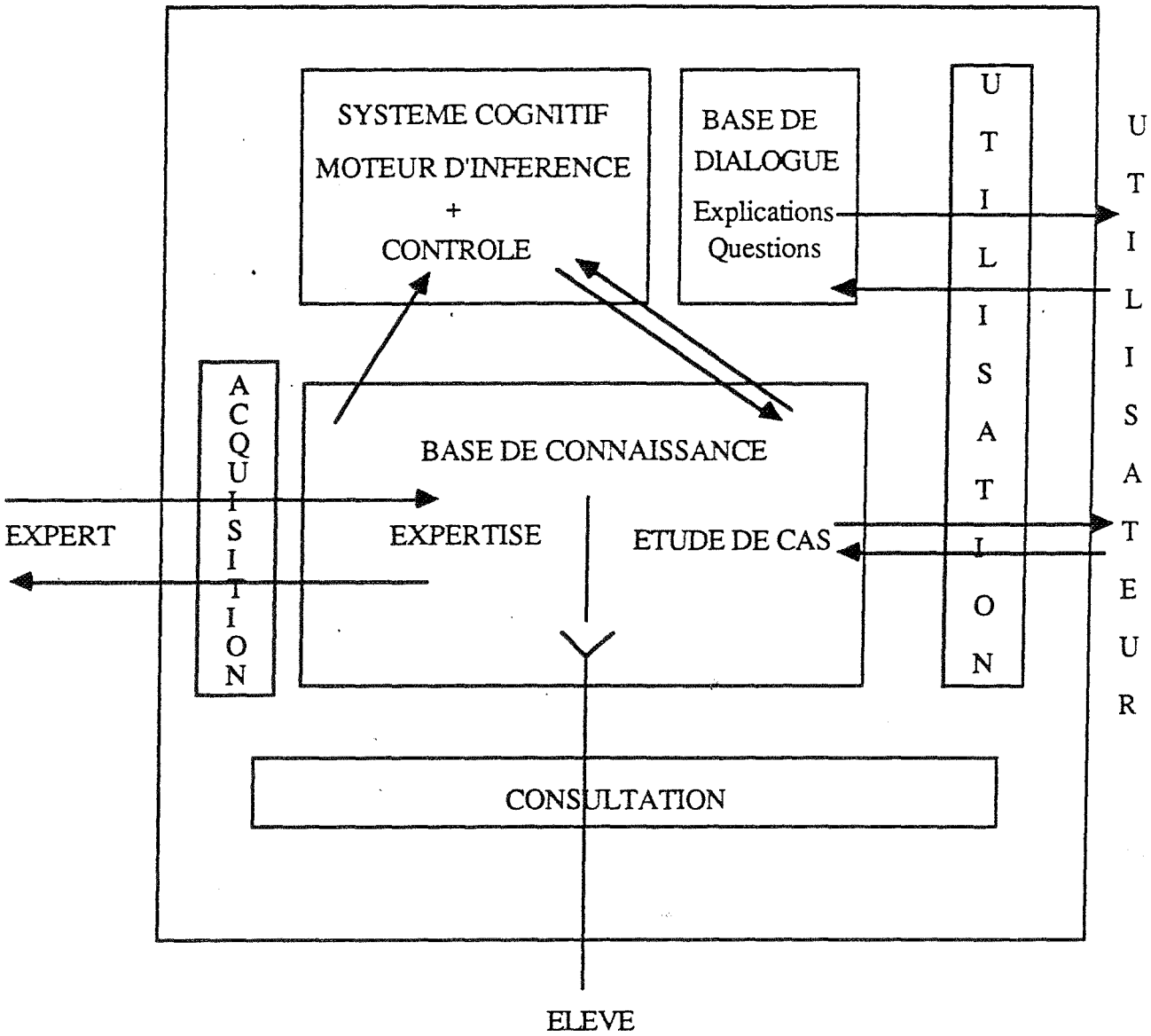


FIGURE 1.1

2 LA MODELISATION DES CONNAISSANCES

Le développement d'un logiciel, système expert ou autre, peut être considéré comme la traduction des connaissances de l'activité à informatiser dans une représentation utilisable par l'ordinateur. Actuellement, selon Sako [Sako86] les trois paradigmes de programmation les plus fréquemment utilisés pour représenter les connaissances sont :

- la programmation orientée vers les procédures
- la programmation orientée vers les objets
- la programmation orientée vers les règles

Nous allons maintenant rappeler les caractéristiques de ces trois paradigmes.

2.1 LA PROGRAMMATION ORIENTEE VERS LES PROCEDURES

Dans notre exposé, le terme de procédures désigne aussi bien une procédure qu'une fonction, une routine ou un sous-programme. Ce modèle de programmation est le plus ancien, le plus éprouvé, et c'est encore aujourd'hui le plus employé, en effet c'est celui de tous les langages traditionnels de programmation comme FORTRAN, PASCAL, C et même LISP où tout n'est que fonction. Il distingue deux sortes d'entités très différentes les procédures et les données. Les données sont de deux sortes, les instances d'entrées du système, et les résultats à obtenir ou les sorties du système. Les procédures implémentent les actions à réaliser pour produire les sorties à partir des entrées, ce qui fait qu'on parle aussi de programmation impérative. Parmi les connaissances d'un domaine d'expertise, ce paradigme de programmation permet de représenter et de traiter de manière satisfaisante les connaissances numériques. En effet, les connaissances factuelles doivent être représentées par des variables informatiques et les connaissances opératoires par des algorithmes.

2.2 LA PROGRAMMATION ORIENTEE VERS LES REGLES

Dans la programmation orientée vers les règles, l'informaticien représente les connaissances comme des ensembles de couples conditions-actions ou prémisses-conclusions. Ces ensembles ou règles décrivent des actions à réaliser et les conditions à vérifier pour que ces actions soient effectuées. Chaque règle est indépendante des autres règles de la base de règles. Pour résoudre un problème, le moteur d'inférences examine la base de règles et sélectionne la règle applicable en fonction des faits déjà connus à cet instant. Une règle constituant en fait la description d'un problème élémentaire plutôt que l'énoncé de la suite d'actions à exécuter pour le résoudre, la programmation orientée vers les règles est aussi appelée programmation déclarative.

Les formalismes les plus employés comme supports théoriques des systèmes à base de règles sont :

- les sytèmes de productions

- la logique des propositions
- la logique des propositions étendue
- la logique du premier ordre.

Il faut cependant remarquer que l'implémentation informatique d'un système à base de règles est rarement la mise en oeuvre exacte d'un de ces formalismes. En fonction des problèmes que ce système essaie de résoudre, la réalisation présente soit des extensions soit des restrictions.

2.2.1 Les systèmes de production

Dans les systèmes de production définis pour la première fois par Post, les connaissances sont représentées par des règles de production. Les règles de production expriment les relations logiques qui existent entre les entités de la base de connaissances. La forme générale d'une règle étant

```
SI condition ALORS action
```

la partie action de la règle décrit les actions à exécuter lorsque la situation définie dans la partie condition est établie. Les règles d'un système de production forment un réseau car les conclusions de certaines règles peuvent apparaître dans les prémisses d'une autre règle.

2.2.2 La logique des propositions

En logique des propositions, les connaissances factuelles d'un domaine d'expertise sont représentées par des variables propositionnelles qui peuvent seulement prendre les valeurs "vrai" ou "faux". Les connaissances opératoires sont alors représentées par des formes propositionnelles, formules où plusieurs variables propositionnelles sont reliées par des connecteurs logiques. Si la logique des propositions dispose de cinq connecteurs ("et", "ou", "non", "implication" et "équivalence"), la plupart des moteurs d'inférences conçus pour exploiter une base des connaissances écrites en logique des propositions n'autorisent que les connecteurs "et", "non", "implication". Ainsi la forme suivante est une règle correcte :

```
si A
et non B
et C
alors D
et non E
```

Les moteurs d'inférences basés sur la logique des propositions sont appelés moteurs d'inférences d'ordre 0.

Cette représentation assez grossière ne permet pas toujours de bien modéliser les raisonnements d'un expert et notamment les raisonnements à conduire lorsque les informations nécessaires à la résolution ne sont pas encore connues ou ne peuvent pas être déterminées. Dans de nombreux langages de représentation des connaissances, la logique des propositions est étendue et les variables propositionnelles peuvent prendre, en plus des valeurs "vrai" et "faux", les valeurs "inconnu" et "indéterminé".

2.2.3 La logique des propositions étendue

Pour modéliser certaines connaissances, comme dans cet exemple tiré du système expert de conseil en placement que nous avons développé :

si le client est marié
alors le nombre maximum de plans d'épargne logement que le foyer fiscal peut détenir est égal à 2 + le nombre d'enfants à charge

la logique des propositions se révèle impuissante ou inadaptée. Dans ce cas précis, en logique des propositions il faut écrire autant de règles qu'un foyer peut avoir d'enfants ce qui est très lourd et inefficace. La solution la plus simple pour résoudre ce problème consiste à remplacer les variables propositionnelles par des couples énoncé-valeur. Les valeurs autorisées peuvent aussi bien être un nombre, une expression arithmétique ou une chaîne de caractères qu'un booléen. Les valeurs "inconnu" et "indéterminé" ajoutées à la logique des propositions ne sont plus maintenant qu'une des valeurs possibles pour un fait. Avec un tel langage de représentation, la connaissance de l'exemple précédent est représentée par la règle :

```
SI    la situation de famille = "marié"
ALORS le nombre maximum de PEL = ( 2 + le nombre d'enfants à charge )
```

Les moteurs d'inférences qui utilisent ce formalisme de représentation des connaissances sont qualifiés de moteurs d'ordre 0+. La majorité des systèmes experts utilisent un moteur d'inférences de ce type.

2.2.4 La logique du premier ordre

Toutefois, même la logique des propositions étendue ne permet pas de représenter des connaissances très générales du genre :

Si X est le père de Y et Y le père de Z alors X est le grand-père de Z

car elle interdit l'emploi de variables quantifiées universellement dans la formulation des règles. Pour pouvoir modéliser une telle connaissance, il faut avoir recours à la logique du premier ordre. C'est la logique des moteurs d'inférences PROLOG [Clocksin81], SNARK [Laurière84], KOOL. Par exemple, dans le formalisme PROLOG, on écrit :

```
grand_pere(X,Z) :- pere(X,Y), pere(Y,Z).
```

pour représenter la connaissance précédente. Comme en PROLOG toutes les variables sont quantifiées universellement, les quantificateurs sont omis pour simplifier l'écriture.

Si la logique du premier ordre constitue un excellent support théorique pour un moteur d'inférences, qualifié alors de moteur d'ordre 1, la diffusion de ces moteurs est limitée par la difficulté d'une mise en oeuvre efficace. De plus avec un moteur d'ordre 1, si le programmeur n'est pas vigilant, les risques d'explosion combinatoire ou de création de boucle infinie sont très importants. Cependant, leur puissance est parfois indispensable pour aborder certains problèmes qui exigent l'examen de toutes les solutions possibles à un problème.

Par exemple, un ingénieur système sait qu'il est préférable d'utiliser les terminaux Ampex 210 en émulation *tvi925* et de choisir l'entrée correspondante dans *terminfo* pour que *vi* fonctionne correctement avec ce modèle de terminaux. Cette connaissance se traduit en KOOL par la règle en chaînage-arrière :

```
{InstRule - : BR36

 If
   *Terminal^Marque = Ampex
   *Terminal^Modele = 210
 Then
   *Terminal^Emul = tvi925}
```

Cette règle est déclenchée, quand le moteur d'inférences de KOOL cherche à déterminer l'émulation utilisée par un terminal, pour tous les terminaux de type Ampex 210.

Ces formalismes conviennent bien à la représentation de connaissances **certaines**. Mais dès les tous premiers développements de systèmes experts, ils se sont révélés insuffisants pour représenter les connaissances incertaines ou incomplètes. Pour modéliser ces dernières, différentes solutions ont été proposées, dont :

- associer à chaque fait et à chaque règle un coefficient de certitude qui traduit la confiance que l'utilisateur accorde à un fait ou que l'expert accorde à une règle. Le moteur d'inférences calcule en fonction des coefficients des prémisses et du coefficent propre à la règle le coefficient de certitude des conclusions. Cette solution a déjà été retenue par les développeurs de MYCIN [Davis77].

- introduire des logiques multivaluées [Rescher69] [Prade82], ou des logiques modales [Hughes68].

- utiliser la logique floue [Zadeh].

Seul les coefficients de certitude connaissent une utilisation répandue.

2.3 LA PROGRAMMATION ORIENTEE VERS LES OBJETS

2.3.1 Introduction

La programmation orientée vers les objets est apparue avec les langages SIMULA [Dahl70], Smalltalk [Goldberg83], les frames [Minsky75], [Bobrow77] et les acteurs [Hewitt75]. Le développement des langages orientés objets a été influencé par deux branches de l'informatique : le génie logiciel et l'intelligence artificielle. Nous allons donc, avant d'exposer les principes de la programmation orientée vers les objets, présenter les problèmes à résoudre du point de vue du génie logiciel et du point de vue de l'intelligence artificielle qui ont provoqué la définition des langages orientés objets.

2.3.2 Le point de vue du génie logiciel

Le but principal du génie logiciel est de définir des méthodologies, des techniques qui garantissent la production de logiciels de qualité. D'après Meyer [Meyer88], les cinq plus importantes qualités d'un logiciel sont :

- la justesse : un bon logiciel doit répondre exactement aux demandes de l'utilisateur et aux spécifications qui en ont été faites.

- la robustesse : capacité d'un logiciel à réagir dans des situations anormales non prévues dans les spécifications. Un logiciel ne doit pas, si un tel événement se produit, provoquer de dégâts graves au système, mais plutôt s'arrêter proprement.

- l'extensibilité : possibilité de modifier facilement un logiciel pour prendre en compte des changements dans les spécifications.

- la réutilisabilité : capacité d'un logiciel à être, en totalité ou en partie, réutilisé lors du développement de nouvelles applications. En effet, pendant le développement d'un logiciel, un effort trop important est le plus souvent consacré à la réécriture de programmes qui existent déjà.

- la compatibilité : possibilité pour un programme de communiquer facilement avec d'autres pour construire un logiciel complexe.

A ces cinq propriétés fondamentales, nous en rajoutons une sixième qui influence beaucoup la robustesse d'un système, l'intégrité ou la capacité du logiciel à protéger ses programmes, ses données contre les agressions extérieures et plus particulièrement contre les virus.

La programmation informatique classique orientée vers les procédures permet d'obtenir difficilement des logiciels possédant toutes ces qualités, notamment l'extensibilité et la réutilisabilité. En effet, la séparation des données et des procédures encourage une programmation où, l'accès au données étant réparti dans l'ensemble du programme, la moindre modification de la structure des données entraîne la mise à jour de nombreuses parties du programme. Aussi une solution proposée pour améliorer la qualité des logiciels est la programmation orientée objets, car :

- un objet qui regroupe données et fonctions exploitant ces données constitue une unité de programmation ou un module autonome et indépendant des autres modules du système. En effet, un objet définit seulement quelques interfaces qui permettent aux autres objets de l'application d'accéder à ses données visibles. De cette façon, l'objet peut ainsi être modifié plus aisément. Tant que les modifications de la structure interne n'imposent pas de modifications des interfaces, elles n'ont pas d'influence sur les autres modules. Ce principe qui ne laisse accessible à l'extérieur que certaines informations de l'objet s'appelle le masquage de données ou "data-hiding" en anglais.

- un objet est en fait défini par un modèle ou prototype abstrait appelé classe qui décrit la structure de cet objet. Du point de vue du génie logiciel, les objets existent uniquement lors de l'exécution du programme alors que les classes appartiennent seulement au texte du programme. En effet, la notion de classe correspond à la notion de type des langages traditionnels. Comme les langages orientés objets conçus pour le génie logiciel sont des langages fortement typés, le compilateur exerce un contrôle strict des types pour essayer de détecter le plus grand nombre d'erreurs avant l'exécution.

- l'utilisation, d'une part, de classes génériques et, d'autre part, la possibilité de définir une classe comme une extension ou une restriction d'une autre classe (héritage) encourage leur réutilisation plus fréquente. Malheureusement l'héritage ne définit souvent qu'un seul lien, le lien *is-a* où un objet hérite de toutes les caractéristiques de ses ancêtres, et limite ainsi les factorisations possibles. A l'inverse, la disponibilité d'un lien *a-kind-of*, qui autorise la factorisation de seulement quelques unes des propriétés d'un objet, permet d'exprimer la ressemblance entre deux objets et ainsi favorise la définition d'un objet à partir d'un objet existant. Car il est souvent utile de définir un objet à partir d'un autre non par spécialisation, mais par association parce qu'il ne possède que certaines des caractéristiques du parent en plus de ses caractéristiques propres [Michel88].

2.3.3 Le point de vue de l'intelligence artificielle

L'intelligence artificielle et, plus précisément, les systèmes experts, cherchent à développer des programmes qui, comme ils essaient de reproduire le plus fidèlement possible le raisonnement d'un expert humain, font peu appel à l'utilisation et à l'optimisation d'algorithmes bien définis. Ce but ne peut être atteint qu'en utilisant des environnements de programmation qui permettent le développement rapide de prototypes facilement modifiables. Les langages de

programmation des environnements employés en intelligence artificielle doivent donc satisfaire les quatre critères suivants :

- la modularité : elle permet de décomposer la connaissance d'un domaine d'expertise en connaissances élémentaires ou granules de connaissance indépendantes les unes des autres, et de décomposer un problème en sous-problèmes plus simples à résoudre

- l'extensibilité : pendant le développement d'un système expert, les connaissances sont obtenues de manière incrémentale et révisable, il faut donc pouvoir modifier facilement les modules déjà définis et particulièrement les structures de contrôle pour reproduire le raisonnement de l'expert

- la multiplicité des représentations : la compréhension de problèmes complexes peut demander l'utilisation de différentes représentations ou plus exactement de différents points de vues sur les entités les modélisant

- l'uniformité : elle impose plusieurs caractéristiques au langage orienté objets :

  - toute entité du langage doit être considérée comme un objet, donc en particulier une classe est un objet

  - aucun objet n'a de statut particulier, c'est-à-dire qu'il n'existe pas de différence entre un objet défini par l'utilisateur et un objet de base du système

  - tous les objets communiquent entre eux grâce à la transmission de messages.

  L'uniformité permet donc de modifier dynamiquement la structure d'un objet, et en particulier son comportement, et offre ainsi aux programmeurs la très grande souplesse nécessaire à la représentation du raisonnement d'un expert. Mais la compilation d'un langage offrant une telle souplesse est presque impossible, aussi les langages orientés objets conçus pour l'intelligence artificielle sont tous interprétrés.

Or l'apparition de langages objets interprétés à partir de 1975 qui satisfont ces critères, explique leur utilisation de plus en plus répandue en intelligence artificielle. De plus, leur association avec des environnements de programmation agréables et efficaces (graphiques, fenêtres, souris ...) a contribué à leur succès.

2.3.4 Les concepts de la programmation objet

2.3.4.1 L'objet

Un objet est un regroupement structuré d'informations qui peuvent aussi bien être des fonctions que des données. Un objet constitue une unité autonome. Comme la terminologie permettant de décrire les caractéristiques des objets n'est pas encore complètement stabilisée, nous allons préciser celle que nous utiliserons par la suite, à savoir celle de KOOL.

Les données conservées dans un objet sont les **attributs** de cet objet. Chaque attribut peut également posséder ses propres caractéristiques qui sont désignées par le terme de **descripteurs** (certains auteurs emploient plutôt le terme de facette). Par exemple, les descripteurs servent à définir le type de la valeur d'un attribut, le domaine de validité, une procédure à exécuter pour calculer la valeur de l'attribut. Enfin chaque comportement ou fonction d'un objet est représenté par un couple **sélecteur-méthode**. Les objets communiquent par envoi de **messages**. Un message est composé d'un **destinataire**, d'un **sélecteur** et de **paramètres**. La méthode associée à un sélecteur est la fonction qui implémente le comportement de l'objet lorsqu'il reçoit un message contenant ce sélecteur.

2.3.4.2 La classe

Une classe est une structure de données qui décrit une entité abstraite, le schéma d'instances d'un groupe d'objets possédant des caractéristiques ou attributs, et des comportements ou méthodes communs. Définir une classe d'objets c'est définir :

- un ensemble d'attributs.
- un ensemble de couples sélecteur-méthode.

La structure de chaque attribut est elle-même déterminée par ses descripteurs.

Une classe apparaît donc comme un moule car elle est avant tout une description des potentialités des objets qu'elle engendre. Ces objets sont les instances de la classe. Dans les langages objets développés pour l'intelligence artificielle, comme une classe est également un objet, instance d'une méta-classe, il existe dès lors deux niveaux de manipulations :

- la manipulation des classes : création d'instances, modification dynamique de leurs caractéristiques (attributs et méthodes), création d'une nouvelle classe ...
- la manipulation des objets : affectation des valeurs des attributs, déclenchement d'une méthode ...

Par contre dans les langages objets conçus pour le génie logiciel, les classes jouent le même rôle que les types PASCAL ou C, et donc il n'existe qu'un niveau d'opérations, celui des objets.

2.3.4.3 L'instanciation

L'instanciation est le mécanisme de création dynamique d'un objet à partir d'une classe qui établit les relations sémantiques entre la classe et l'instance générée. Une instance est un objet qui représente un cas particulier de la classe la définissant, elle possède des valeurs propres pour chacun des attributs de la classe mais suit les comportements de la classe. Naturellement, il est possible de créer plusieurs instances d'une même classe qui se différencient par les valeurs des attributs. L'instanciation regroupe deux opérations :

- la création de l'objet qui consiste à allouer de la place mémoire pour ses attributs et à rendre les méthodes définies dans la classe applicables sur cet objet. Elle est généralement assurée par l'envoi d'un message dont le sélecteur est NEW.

- l'initialisation qui donne des valeurs initiales aux attributs. Sauf indications contraires, la plupart des langages objets donnent lors de l'instanciation des valeurs indéfinies (*nil* en LISP) aux attributs.

2.3.4.4 Les comportements

Nous venons de voir que l'instanciation d'un objet est réalisée par l'envoi d'un message. Mais l'envoi de message est aussi le moyen usuel de déclencher une procédure ou fonction appartenant à un objet. L'envoi de message n'est qu'une extension de l'appel classique de fonction, il faut seulement préciser en plus l'objet destinataire. Un message est formé :

- d'un receveur : l'objet destinataire

- d'un sélecteur : l'information permettant au receveur de trouver la méthode à appliquer parmi les méthodes de la classe

- d'arguments éventuels : les valeurs des paramètres de la méthode.

A la réception d'un message, l'objet destinataire devient l'objet actif. L'exécution d'un programme écrit dans un langage objet n'est qu'une suite d'envois de message entre les objets de ce programme.

Certains langages objets développés pour des applications en intelligence artificielle disposent également d'une autre technique que l'envoi de message pour déclencher des fonctions : l' **attachement procédural**. L'attachement procédural est un concept qui vient des frames. Son emploi permet l'écriture de programmes dont l'exécution est dirigée par les données ce qui peut dans certains cas convenir à la représentation du raisonnement d'un expert.

L'attachement procédural est le mécanisme qui gère l'exécution des procédures ou démons associés à un attribut d'un objet. Ces démons peuvent être déclenchés :

- pour obtenir la valeur de l'attribut

- pour exécuter un traitement lorsque l'attribut vient de recevoir une valeur

- pour exécuter un traitement lorsque la valeur de l'attribut est invalidée.

2.3.4.5 l'héritage

Le dernier concept important des langages objets est l'héritage. En effet l'héritage est très intéressants du point de vue du génie logiciel [Meyer88]. A partir d'une classe déjà définie, l'héritage permet de créer une nouvelle classe par adaptation. Dès sa déclaration une sous-classe possède les caractéristiques de ses parents. Pour définir totalement cette sous-classe il ne reste plus qu'à modifier certaines caractéristiques héritées, à ajouter celles qui lui sont propres ou à ajouter de nouvelles contraintes sur les valeurs des attributs. L'héritage favorise donc une conception par raffinements successifs qui petit à petit rajoutent de nouvelles caractéristiques aux classes.

Par conséquent la création d'un graphe d'héritage permet de structurer hiérarchiquement les classes en factorisant les propriétés communes à plusieurs classes dans une classe supérieure appelée super-classe. Les classes héritant d'une super-classe constituent ses sous-classes. Chaque objet hérite alors des attributs, des sélecteurs et des méthodes définis dans toutes les classes dont il descend. Cependant, une propriété définie dans le schéma d'instances d'une classe peut être redéfinie dans une sous-classe.

Le regroupement des propriétés communes à plusieurs classes dans une super-classe entraîne également une factorisation de la programmation. Comme, grâce à l'héritage, une propriété est définie une et une seule fois dans un programme, d'une part les sources du programme sont plus compactes, et d'autre part toute modification d'une propriété est automatiquement transmise aux sous-classes qui héritent de la classe où cette propriété est définie. Cette dernière caractéristiques des langages objets diminue considérablement la difficulté de la maintenance d'un programme car il n'est plus nécessaire d'explorer la totalité du programme pour corriger toutes les utilisations de la propriété modifiée.

Il existe deux sortes d'héritages :

- l'héritage simple où une classe ne peut posséder qu'une super-classe. Dans ce cas le parcours du graphe d'héritage est très simple, il suffit de remonter l'arbre jusqu'à la classe où la propriété est disponible.

- L'héritage multiple où une classe peut hériter de plusieurs super-classes. Quand un même attribut apparaît dans plusieurs super-classes, il se pose alors un problème de résolution de conflit pour déterminer laquelle des définitions possibles utiliser pour cet attribut. Chaque langage objet disposant de l'héritage multiple propose une stratégie de parcours du graphe d'héritage ou une méthode comme le renommage dans EIFFEL pour résoudre ce problème. Comme Ducournau et Habib [Habib88], nous pensons qu'effectivement un langage objet doit proposer une ou plusieurs stratégies de parcours du graphe d'héritage, mais qu'il doit laisser libre le concepteur de choisir ou même de redéfinir une stratégie pour chaque attribut de chaque classe. En effet, quelle que soit la stratégie proposée et quelles que soient les considérations algorithmiques avancées pour la justifier, elle correspond rarement à la sémantique de l'héritage propre à chaque

application.

2.3.4.6 Un exemple de langage objet : KOOL

KOOL est un environnement de développement de systèmes experts construit autour d'un langage objet, examinons les caractéristiques de ce langage.

KOOL, pour faciliter la tâche des développeurs, fournit un certain nombre de descripteurs prédéfinis, parmi lesquels les plus utilisés sont :

- *Mode* qui précise si l'attribut possède une seule valeur (*Mode :Mono*) ou peut recevoir plusieurs valeurs (*Mode : Multi*)

- *MaxCard* qui fixe le nombre maximum de valeurs autorisées pour un attribut multi-valué

- *Type* qui définit le type (nombre, chaîne de caractères, objet) de la valeur de l'attribut

- *Askable* qui indique s'il est possible d'obtenir la valeur de l'attribut en interrogeant l'utilisateur, les valeurs possibles pour ce descripteur sont *Yes* et *No*

- *Question* qui contient le texte de la question à poser pour demander la valeur de l'attribut

- *Info* qui contient un texte d'explication sur l'attribut que l'utilisateur peut consulter lorsque KOOL lui demande la valeur de cet attribut.

KOOL pour gérer ses objets utilise des messages dont les sélecteurs sont les suivants :

- *New* pour créer une instance d'une classe

- *Kill* pour détruire un objet

- *Show* pour afficher sur l'écran un objet

- *Edit* pour éditer puis modifier un objet

KOOL dispose également de l'attachement procédural. Les descripteurs prédéfinis *ToCompute*, *WhenFilled*, *WhenRemoved* permettent d'introduire des démons pour respectivement calculer la valeur d'un attribut, exécuter des calculs lorsqu'une valeur est affectée à un attribut, exécuter des calculs lorsque la valeur d'un attribut est supprimée.

Par exemple la classe représentant la périphérie d'un ordinateur s'écrit :

```
{InstClass : Peripherie
 Super : Object

 <  -- NbTerm : ()
       Mode : Mono
       Type : [0..32]
       Askable : Yes
       ToCompute : askNT
       Info : {Les micro-ordinateurs personnels ne sont pas a prendre
               en compte, ils apparaissent dans les attributs NbMach
               et MachV24. Les bitmaps seront egalement traites a part}
       WhenFilled : [demTerm]
    -- Term : ()
       Mode : Multi
       Type : Terminal
       Askable : No
       MaxCard : 32
    -- NbImpPar : ()
       Mode : Mono
       Type : [0..4]
       Askable : Yes
       Question : {Combien d'imprimantes paralleles sont connectees
                      sur votre machine ?}
       WhenFilled : [demImpPar]
    -- ImpPar : ()
       Mode : Multi
       Type : ImpriPar
       Askable : No
       MaxCard : 4
    >}
```

L'héritage dans KOOL est un héritage simple. Ainsi, nous utilisons une classe imprimante regroupant les informations communes à toutes les imprimantes qui possède deux sous-classes : les imprimantes gérées par une interface série et celles disposant d'une interface parallèle. En KOOL, ces trois classes sont définies de la manière suivante :

```
{InstClass : Imprimante
 Super : Object

 <  -- Marque : ()
       Mode : Mono
       Type : String
       Askable : Yes
       Question : {Quelle est la marque de l'imprimante * ?}
```

```
    -- Modele : ()
       Mode : Mono
       Type : String
       Askable : Yes
       Question : {Quel est le modele de l'imprimante * ?}
    -- Nom : ()
       Mode : Mono
       Type : String
       Askable : Yes
       Question : {Quel nom voulez-vous donner a l'imprimante * ?}
       WhenFilled :[demNomImp]
    -- Interface : ()
       Mode : Mono
       Type : [serie,parallele]
       Askable : No
    >}

{InstClass : ImpriPar
 Super : Imprimante

 <  -- Interface : parallele
    >}

{InstClass : ImpriSer
 Super : Imprimante

 <  -- Interface : serie
    -- Vitesse : ()
       Mode : Mono
         Type : [|19200|,|9600|,|4800|,|2400|,|1200|,|300|,|console|,|PAD|]
       Askable : Yes
       Question : {A quelle vitesse travaille l'imprimante * ? }
    -- Voie : ()
       Mode : Mono
       Type : Voie
       Askable : No
    >}
```

2.4 LES REPRESENTATIONS HYBRIDES

Très rapidement, lors du développement des premiers systèmes experts, il est apparu que l'utilisation d'un seul paradigme de programmation peut conduire à des impasses, car certaines connaissances se représentent mal uniquement avec des procédures, des règles ou des objets. Ainsi il est reconnu que l'utilisation simultanée des modes de représentation des connaissances facilite la modélisation des connaissances. Selon Aikins [Buchanan84], les avantages de cette approche sont de deux sortes :

- ceux concernant la représentation des connaissances
- ceux concernant le raisonnement et les performances

2.4.1 La représentation des connaissances

Lors du développement d'un système expert avec pour seule représentation des connaissances les règles, comme nous avons pu le constater pendant l'étude menée pour la Société Lyonnaise de Banque, l'ingénieur cogniticien est contraint, pour contrôler le déclenchement des règles, d'ajouter dans la formulation des règles des faits totalement étrangers à l'expertise mais indispensables pour obtenir un comportement du système semblable à celui d'un expert humain. Ainsi la base de règles devient moins lisible pour cet expert et le transfert d'expertise est rendu plus délicat. Non seulement la lisibilité de la base de règles est réduite, mais aussi ces faits parasites pertubent le module explicatif car ils ne correspondent à aucune expertise, ils traduisent uniquement des contraintes informatiques. En revanche, avec des objets et des règles, le contrôle disparaît des règles pour se concentrer dans les objets. Chaque règle devient ainsi une véritable "granule" d'expertise.

Un autre avantage apporté par les objets est une représentation plus facile de certaines connaissances, et notamment des objets du monde réel. Une instance d'une classe forme exactement un calque informatique d'un modèle de l'objet réel représenté par cette instance, car les valeurs de ses attributs correspondent aux caractéristiques mêmes du modèle de l'objet réel représenté. La modélisation de ces connaissances est donc immédiate. De plus, grâce aux démons et aux méthodes, nous pouvons définir exactement le comportement de chaque objet en toutes circonstances, notamment lors de la création et l'initialisation de cet objet.

Le troisième avantage consiste en une maintenance du système beaucoup moins coûteuse. Pour faire évoluer un système à base de règles, il faut tenir compte de la partie contrôle de chaque règle, or cela devient très difficile dès que le système est suffisamment important et complexe. Avec des objets et des règles, chaque unité de savoir (règle ou objet) est indépendante des autres, il est donc aisé de la modifier, de la détruire, ou d'en rajouter une nouvelle. La maintenance est, de plus, facilitée lors de la modification d'une classe existante par l'encapsulation des données, des fonctions et des règles dans les objets. De cette façon lors de la modification d'un attribut, nous connaissons immédiatement les procédures, les règles qui l'utilisent et nous pouvons à leur tour les modifier.

2.4.2 Le raisonnement et les performances

Les comportements d'un objet étant explicitement définis par ses démons et ses méthodes, le développeur maîtrise totalement l'ordre d'acquisition des informations dont le système a besoin pour conduire son raisonnement, et plus particulièrement l'ordre des questions à poser à l'utilisateur pour obtenir de nouveaux renseignements. De cette manière, il peut plus aisément modéliser le raisonnement de l'homme de l'art, et obtenir des réactions paraissant plus humaines et plus naturelles. Par conséquent ce système raisonnant comme un homme sera vraisemblablement mieux accepté par les utilisateurs.

De plus, les systèmes mêlant objets et règles sont plus efficaces que les systèmes basés sur un seul paradigme de programmation. En effet le système associe, lors de la compilation des règles grâce à l'algorithme de Rete [Forgy82], par exemple, à chaque attribut de chaque objet un descripteur qui contient la liste des règles l'utilisant, et ainsi à chaque cycle le moteur d'inférences connaît immédiatement les règles intéressantes. Ce sont celles concernant l'objet courant (l'objet possédant l'attribut dont on cherche à déterminer la valeur). L'étape de restriction du cycle du moteur d'inférences consiste seulement à lire dans le descripteur de l'attribut correspondant de l'objet courant la liste des règles à examiner, au lieu de garder l'ensemble des règles de la base comme c'est souvent le cas. L'étape de filtrage est alors bien plus courte, car moins de règles sont à tester.

Enfin, dans un environnement de développement de systèmes experts hybride, les règles sont souvent elles-mêmes considérées comme des objets. Ainsi un programme peut manipuler les règles comme tout autre objet de l'application, et en particulier il peut créer une nouvelle règle, modifier ou détruire une règle existante. Cette vision objet des règles offre donc une grande puissance pour modéliser le raisonnement d'un expert.

Toutes ces raisons font que les règles et les objets sont souvent utilisés conjointement en Intelligence Artificielle, notamment pour développer des systèmes experts. En effet, ils conviennent bien à la programmation incrémentale nécessaire dans ce domaine grâce à l'indépendance des règles et des objets. Une règle doit alors pouvoir accéder aux objets et aux attributs d'un objet. Comme en KOOL chaque objet possède un nom, si l'objet représentant le calculateur à installer s'appelle *Cal*, pour déterminer la modèle de ce calculateur en fonction de sa marque, on utilise la règle suivante :

```
{InstRule - : BR1

 If
   Cal^Marque = Telmat
 Then
   Cal^Modele = SM90
 Extern {La marque du calculateur est Telmat, son modèle est donc SM90}
}
```

Grâce, d'une part, au fonctionnement du moteur d'inférences d'un système à base de règles, et d'autre part, à l'attachement procédural des objets, l'utilisation des règles et des objets permet la construction de systèmes dont l'exécution est dirigée par les données au lieu d'être définie lors de l'écriture du programme, et de développer plus facilement des logiciels "intelligents".

2.5 COMMENT CHOISIR UNE REPRESENTATION DES CONNAISSANCES ?

Nous venons de montrer que l'utilisation simultanée des trois paradigmes de programmation permet de résoudre des problèmes plus complexes. Mais il existe aussi des problèmes qui se résolvent très bien avec seulement des procédures, des règles ou des objets. Quels critères faut-il examiner pour déterminer quel est le modèle qui convient le mieux pour représenter les connaissances d'un problème. Pour ce faire, il faut arriver à déterminer les caractéristiques de ces connaissances.

Tout d'abord, il faut identifier les types de représentation qui conviennent aux connaissances élémentaires manipulées :

- elles sont booléennes
- elles sont numériques
- elles sont symboliques (chaînes de caractères)

ainsi que les structures des représentations des connaissances :

- des listes
- des arbres
- des tableaux
- des enregistrements.

Ensuite, il faut essayer de déterminer :

- la structure de la base de connaissances
- la part de la programmation déclarative et de la programmation procédurale dans la résolution, et comment ces deux programmations doivent être combinées
- si les connaissances à traiter sont certaines, incertaines ou incomplètes
- le type du raisonnement de l'expert, monotone ou non.

Ces informations sur les caractéristiques des connaissances du domaine sont obtenues grâce à une première étude qui permet donc de fixer la ou les représentations des connaissances à utiliser. Par exemple, si les connaissances peuvent être représentées par des booléens

et des symboles et si elles ne sont pas organisées, la logique des propositions convient parfaitement à la résolution d'un problème de ce type. Par contre, si on doit représenter des objets complexes, en nombre a priori inconnu, et si l'expert utilise dans ses raisonnements élémentaires seulement quelques propriétés intéressantes des objets, alors l'utilisation d'une représentation hybride, règles avec variables et objets, s'impose.

3 LE FONCTIONNEMENT D'UN MOTEUR D'INFERENCES

Le moteur d'inférences est le programme dans un système expert qui exploite la base de connaissances, règles et faits, pour déduire de nouvelles connaissances. Nous allons étudier :

- le cycle de base d'un moteur d'inférences
- les modes fondamentaux d'invocation des règles
- les problèmes

3.1 LE CYCLE DE BASE D'UN MOTEUR D'INFERENCE

Lors du lancement du moteur d'inférences d'un système expert, la base de connaissances contient :

- les faits avérés et les faits à établir, c'est la base de faits
- les connaissances opératoires, c'est la base de règles.

Le moteur d'inférences enchaîne des cycles de travail qui comportent deux phases :

- l'évaluation : le moteur d'inférences cherche s'il existe, dans la base de règles courante, des règles qui pourront être déclenchées en fonction de l'état de la base de faits courante
- l'exécution : le moteur d'inférences déclenche les règles retenues par l'évaluation.

L'arrêt du moteur d'inférences peut être provoqué soit pendant l'évaluation en l'absence de règles déclenchables, soit pendant l'exécution, si la règle déclenchée demande explicitement l'arrêt du moteur.

Pour le moteur d'inférences, une règle est formée de deux parties :

- le déclencheur qui définit les conditions de déclenchement de la règle
- le corps qui définit les effets de la règle.

La sélection d'une règle ne se fait ni par son nom, ni par son adresse mais de façon associative par la donnée d'un ensemble de faits compatibles avec la partie déclencheur de la règle. L'accès à la règle est donc réalisé grâce à un fragment même de la règle. Cette sélection associative de la règle à déclencher constitue un des principaux intérêts de ce paradigme de programmation. En effet, elle assure l'indépendance des règles et par conséquent une grande facilité pour ajouter ou supprimer des règles. De plus elle garantit, lors de l'ajout d'une nouvelle règle, que cette règle sera déclenchée dès que ses conditions seront satisfaites. Ainsi le programmeur, qui n'a à gérer explicitement ni les appels aux règles ni l'ordre dans lequel ils se font, comme il le fait en programmation orientée vers les procédures, n'aura aucun problème pour intégrer une nouvelle règle ou en supprimer une.

3.1.1 L'évaluation

Elle se décompose en trois étapes :

- la sélection ou restriction
- le filtrage
- la résolution de conflits

3.1.1.1 La sélection

Cette première étape consiste à déterminer, à partir de l'état courant de la base de faits et de l'état courant de la base de règles, un sous-ensemble de faits et un sous-ensemble de règles jugés intéressants qui seront comparés lors de l'étape de filtrage. Il existe différentes manières de réalisation la sélection.

La détermination de sous-ensembles peut être faite par des méta-règles qui indiquent au moteur d'inférences quelles règles et quels faits il doit examiner.

Quand les règles de la base de connaissances sont regroupées en contextes, qui correspondent à de grandes classes de raisonnement et qui peuvent être assimilés à des préconditionnements de l'utilisation des règles, il est alors possible dans un corps de règle d'activer le contexte à utiliser pour les prochains cycles du moteur d'inférences. Par conséquent les règles appartenant au contexte actif seront les seules à être testées. Par exemple, dans un système expert de diagnostic médical, nous pouvons distinguer a priori les règles propres aux maladies infantiles et les règles propres aux analyses de sang.

Pour les faits, de nombreux systèmes distinguent les faits établis des buts, la restriction peut alors consister à privilégier les buts et même le dernier but apparu par rapport aux autres faits.

Dans le cas des systèmes à base d'objets et de règles, la sélection consiste à ne conserver que les règles où l'objet courant apparaît soit dans les prémisses soit dans les conclusions. Il s'ensuit comme nous l'avons déjà affirmé, une plus grande efficacité par rapport aux autres systèmes où la sélection est inexistante, et où l'étape suivante travaille sur l'ensemble de la base de faits et l'ensemble de la base de règles.

3.1.1.2 Le filtrage

Pendant cette étape, le moteur d'inférences compare les déclencheurs des règles retenues par la sélection avec les faits retenus par la sélection, et détermine celles dont les conditions de déclenchement sont satisfaites. Ces règles potentiellement activables constitue l'ensemble de conflit.

Les techniques de filtrage les plus employées sont :

- l'identité stricte : les faits et les déclencheurs doivent être strictement identiques
- l'identité et la synonymie : pour chaque fait des synonymes ont été déclarés, et le moteur compare les faits avec les synonymes possibles des déclencheurs
- l'indentité significative : les faits et les déclencheurs doivent être identiques après élimination des composants non significatifs, comme les articles, les auxiliaires de conjugaison
- la semi-unification : c'est l'unification des prédicats du premier ordre restreint au cas où la présence des variables n'est autorisée que dans les règles
- l'unification

3.1.1.3 La résolution de conflits

La résolution de conflits consiste à choisir parmi les règles de l'ensemble de conflit celle qui sera effectivement déclenchée. Souvent, la stratégie utilisée pour déterminer cette règle est fondée sur des critères qui sont propres au moteur d'inférences et n'ont aucune signification dans le contexte de l'application.

Nous pouvons citer comme stratégies de résolution de conflits :

- souvent le moteur d'inférences choisit la première règle introduite dans l'ensemble de conflit. Cette stratégie est en effet fort simple à mettre en oeuvre, dès que le moteur trouve une règle déclenchable, il arrête sa recherche et il la déclenche.
- si la base de règles est ordonnée, le moteur choisit celle qui apparaît en premier.
- si le moteur associe une complexité aux règles, il peut choisir la plus simple.
- si le moteur conserve un historique de l'utilisation des règles, il peut retenir celle qui a

le moins servi.

- si à chaque règle est associé un coefficient d'intérêt, le moteur sélectionne la plus intéressante. Chaque règle KOOL possède un attribut *Interest* qui définit sa priorité. La valeur par défaut est 50. Par exemple la règle

```
{InstRule - : BR13

If
  *Disk^Modele = dma
  *Disk^Genre = mobile
Then
  *Disk^ModFormat = dmam
  *Disk^Taille = 9792
  *Disk^PartUnit = 16
  *Disk^Format = |-|
  *Disk^Partition = non
Extern
  {Le disque est la partie mobile d'un dma
    THUS
    le modele de formatage est le dmam, sa taille est 9792
    les partitions doivent avoir une taille multiple de 16
    on ne s'interesse pas a son formatage
    et on ne cree pas de partitions.}
-- Interest : 60
}
```

est déclenchée avant toutes les règles dont la valeur de leur *Interest* est inférieure à 60.

Le plus souvent le moteur d'inférences ne déclenche qu'une seule règle pendant un cycle.

3.1.2 L'exécution

Au cours de cette phase, le moteur d'inférences lance la mise en oeuvre des actions contenues dans le corps de la règle activée. Ces actions peuvent être :

- l'ajout de nouveaux faits dans la base de faits. Dans de nombreux moteurs d'inférences, c'est la seule action possible.

- le retrait des faits existants

- le changement de contexte

- l'ajout ou le retrait d'une règle

- le changement de la base de règles

- le lancement d'une procédure externe de calcul
- la consultation d'une base de données.

Si aucune connaissance, fait ou règle, ne peut être retirée de la base de connaissances, et si aucune connaissance ajoutée à la base de connaissances n'introduit de contradiction, le fonctionnement du moteur d'inférences est alors qualifié de monotone. Par contre, si le moteur d'inférences autorise le retrait de faits ou l'ajout et le retrait de règles qui risquent d'introduire des contradictions dans la base de connaissances et d'interdire des déductions possibles avant cette modification, on parle de moteur d'inférences non-monotone.

Finalement le fonctionnement d'un moteur d'inférences est très bien résumé par le schéma de la figure 3.1 [Farreny85].

3.1.3 Les régimes de contrôle

Le régime de contrôle d'un moteur d'inférences définit la manière dont celui-ci réagit lorsqu'il rencontre un ensemble de conflit vide.

Si le moteur s'arrête, ou s'il essaie, à partir d'un ensemble de conflit précédent, de déclencher de nouvelles règles mais sans remettre en cause les actions des règles dont l'activation a conduit à l'impasse, son régime de contrôle est **irrévocable.**

A l'inverse, si le moteur efface les actions des règles remises en cause, son régime de contrôle est un régime de contrôle **par tentatives** : et on dit que le moteur effectue alors un retour-arrière (backtrack en anglais).

3.2 LES MODES FONDAMENTAUX D'INVOCATION DES REGLES

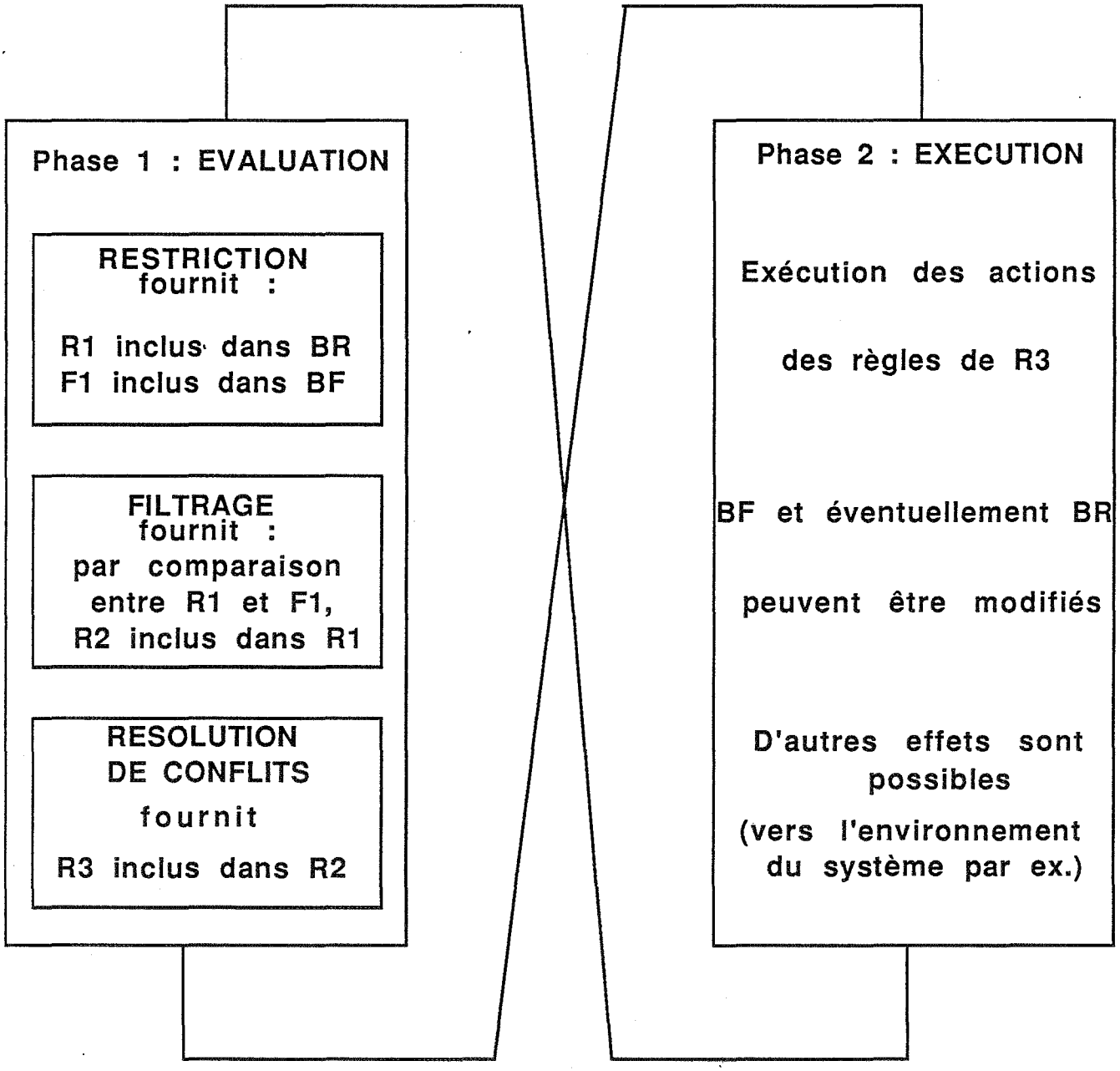
Il existe trois modes fondamentaux d'invocation des règles par un moteur d'inférences :

- le chaînage-avant
- le chaînage-arrière
- le chaînage-mixte.

Nous allons présenter la définition de ces notions [Nilsson80], [Winston84].

3.2.1 Le chaînage-avant

En chaînage-avant, le moteur d'inférences utilise comme déclencheurs les prémisses des règles et les faits déjà établis. Quand il travaille de cette manière, le moteur déduit de nouvelles informations à partir de celles déjà connues, ou autrement dit il conduit un raisonnement des données vers les buts. Si un moteur d'inférences travaillant en chaînage-avant s'arrête seulement quand aucun déclenchement de règle ne peut apporter une modification à base de faits, on dit que ce moteur s'arrête par saturation.



Eventuellement :
arrêt ou retour-arrière
ou autres effets sur le
contrôle du moteur.

Eventuellement :
arrêt ou retour-arrière
ou autres effets sur le
contrôle du moteur.

FIGURE 3.1

3.2.2 Le chaînage-arrière

Un moteur d'inférences fonctionne en chaînage-arrière lorsque les déclencheurs se trouvent parmi les conclusions et sont comparés à des buts ou faits à établir. Lors de l'activation d'une règle, si ses conditions ne sont pas vérifiées, elles viennent se rajouter à la liste des problèmes à résoudre. Un problème est alors considéré comme résolu lorsque tous les sous-problèmes introduits par le déclenchement d'une règle sont eux-mêmes résolus. En chaînage-arrière, un moteur d'inférences mène un raisonnement des buts vers les données, il s'arrête quand les buts à démontrer correspondent à des assertions.

3.2.3 Le chaînage-mixte

Certains moteurs d'inférences sont également capables de marier les deux modes précédents, on dit alors qu'ils fonctionnent en chaînage-mixte.

3.3 LE CHOIX D'UN MOTEUR D'INFERENCES

Dans un système expert, nous avons dit que les règles sont indépendantes les unes des autres. Mais cette indépendance est toute relative, car elle dépend fortement des stratégies de sélection et de résolution de conflits. En effet, ces stratégies sont souvent des propriétés intrinséques du moteur d'inférences et correspondent plus ou moins bien au mode de raisonnement de l'expert.

Pour obtenir un comportement du système expert semblable à celui de l'expert humain, nous sommes alors obligés d'introduire dans la formulation des règles des faits qui ne représentent aucune connaissance de l'expert mais servent uniquement à contrôler le fonctionnement du moteur d'inférences. L'introduction de ces contrôles réduit donc l'indépendance des règles car le programmeur force par ces contrôles le comportement du moteur d'inférences à suivre un chemin déterminé à l'avance de la même manière qu'il fixe le graphe d'appels des procédures dans un programme traditionnel.

Dans le cadre de la maintenance et du développement, il est alors extrêmement difficile de vérifier lors de l'ajout, du retrait, de la modification d'une règle si ce changement de la base de connaissances ne dégrade pas le comportement du système pendant la résolution d'autres problèmes, ou pire n'y introduit pas une incohérence. Actuellement la seule possiblité de valider le comportement d'une base de connaissances est de la tester sur des problèmes dont les résultats sont déjà connus, il faut donc disposer de jeux de tests.

Nous voyons que, malgré les facilités qu'elle apporte, la programmation par règles se heurte à des problèmes similaires à ceux de la programmation procédurale.

En conséquence, de la même manière que la représentation des connaissances doit être choisie soigneusement en fonction de l'expertise à modéliser, la détermination du moteur d'inférences, et notamment de ses stratégies de sélection et de résolution de conflits, est cruciale pour le succès ou l'échec du développement d'un système expert.

TROISIEME PARTIE
LA REALISATION

Chapitre 4

LE DEROULEMENT DE L'ETUDE

1 INTRODUCTION

Ce travail de thèse a débuté au printemps 1985 et peut être découpé en quatre phases qui sont :

- l'acquisition des connaissances nécessaires à l'administration et à l'installation d'UNIX
- le développement d'un système expert de conseil en placement pour la Société Lyonnaise de Banque
- la recherche d'un environnement de développement de systèmes experts
- la réalisation du système expert pour installer UNIX.

Dans ce chapître, nous présentons succinctement chacune de ces étapes. En revanche, dans les chapîtres suivants nous détaillerons la réalisation du système expert de conseil en placement, la modélisation des connaissances nécessaires à l'installation d'UNIX et la présentation du système expert pour installer UNIX.

2 L'ACQUISITION DES CONNAISSANCES POUR INSTALLER UNIX

L'acquisition des connaissances nécessaires à l'administration et à l'installation d'UNIX a tout naturellement débuté par l'étude approfondie des manuels UNIX concernant l'administration du système, à savoir :

- le guide de l'administrateur [ATT84]
- le manuel de référence de l'administrateur [ATT84]
- le guide de l'opérateur [ATT84]
- le guide d'installation et d'utilisation de SMX 4.1 (UNIX vesion 7) sur SM90 [GIPSI85]

Les premières connaissances, ainsi obtenues, ont ensuite servi à réaliser l'installation de SMX 4.1 sur les SM90 du département informatique de l'Ecole des Mines. Cette phase d'acquisition des connaissances s'est terminée par la réalisation d'une maquette d'installateur UNIX sous la forme de shell-scripts. Nous avons décidé de travailler en shell par commodité parce que la plupart des commandes d'installation sont elles-mêmes des shell-scripts. Nous avons utilisé cette maquette lors des dernières installation de SMX 4.1, et nous avons apprécié l'aide apportée même si cet installateur se contentait d'enchaîner mécaniquement les commandes d'installation dans le bon ordre à partir des renseignements fournis par l'ingénieur système.

Cependant, nous avons constaté qu'ainsi nous ne pourrions pas atteindre notre objectif, à savoir développer un logiciel qui permette à un non-spécialiste d'installer un UNIX adapté aux spécificités de sa machine. En effet, si cette première version d'un installateur assurait la génération de commandes syntaxiquement correctes, elle ne possédait pas les connaissances expertes qu'un ingénieur système utilise pour déterminer l'organisation d'un système. De plus cet installateur était totalement dépendant des caractéristiques de SMX 4.1 et de la SM90, et sa complexité était telle qu'elle rendait sa maintenance presque impossible. D'ailleurs nous l'avons abandonné lors de la diffusion de SMX V.1 (UNIX system V) parce que sa mise à jour aurait exigé un effort trop important.

Cette première phase nous a également permis de mieux définir le processus de l'installation d'UNIX et les caractéristiques des connaissances à représenter. En fait, un ingénieur système qui installe UNIX sur un ordinateur détermine la paramétrisation d'UNIX à mettre en place en fonction, d'une part, des caractéristiques matérielles et logicielles de la configuration, et d'autre part, de l'utilisation prévue pour obtenir les meilleures performances, puis il exécute les commandes d'administration nécessaires. Les connaissances qu'utilise l'ingénieur système lors de l'installation d'UNIX sont :

- volumineuses (matériels, logiciels, utilisateurs...)
- difficiles à obtenir car elles se cachent dans de nombreux manuels peu lisibles
- sujettes à de fréquentes mises à jour pour prendre en compte la disponibilité de nouveaux produits.

Par conséquent, le but de l'outil d'aide à l'installation d'UNIX que nous développons est de rendre cette installation possible par un non-spécialiste.

Pour pouvoir modéliser aisément l'expertise de l'ingénieur système, pour résoudre le problème de la maintenance de la base de connaissances, c'est-à-dire suivre l'évolution des logiciels, des matériels, nous avons donc décidé de développer un système expert. Le coeur de ce système expert est constitué des principes généraux d'installation d'UNIX valables pour toutes les machines et pour tous les différents systèmes UNIX ou "UNIX-like". Ainsi, autour de ce noyau nous pouvons facilement prendre en compte les spécificités de chaque machine ou version de système grâce à la méthodologie de développement incrémental des systèmes

experts.

De plus notre décision de développer un système expert a été confortée par le fait qu'un bon système expert doit convaincre l'utilisateur de la justesse de son raisonnement. Or cet objectif ne peut être atteint que si, à chaque étape, le système expert présente les actions qu'il réalise, et les justifie. Ainsi, notre système expert transférera progressivement à l'utilisateur les compétences nécessaires à l'administration d'un système UNIX.

3 LE DEVELOPPEMENT D'UN SYSTEME EXPERT DE CONSEIL EN EPARGNE

Nous avons profité de l'opportunité qui nous était offerte par le développement d'une maquette de système expert de conseil en placement pour la Société Lyonnaise de banque pour acquérir une première expérience du développement de ce type de logiciels. Le but de la Société Lyonnaise de Banque était de vérifier si l'approche système expert pouvait s'appliquer à certaines expertises du domaine bancaire et d'identifier les problèmes de développement particulier à ce domaine. Notre but était plus général, il visait à définir une méthodologie de développement de systèmes experts.

Plus précisément, le but de cette étude était de développer une maquette de système expert qui permette de diffuser auprès des guichetiers de la banque l'expertise des gestionnaires de patrimoine et d'améliorer ainsi la qualité des conseils donnés aux clients. Ce système expert détermine, à partir du profil du client et des caractéristiques des produits disponibles, quel est le placement qui répond le mieux aux aspirations du client. Ensuite il établit le rendement du placement et éventuellement les droits au prêt acquis et les plans de remboursement. Un outil de ce type devient nécessaire parce que :

- les produits proposés par la banque sont de plus en plus nombreux et complexes
- les guichetiers ne peuvent maîtriser tous les produits
- l'évolution de la réglementation, de la fiscalité, de l'environnement financier modifie constamment l'intérêt respectif des produits.

Alors qu'a priori le conseil en épargne et l'installation d'UNIX semblent constituer deux domaines très différents, dans les deux cas nous constatons que :

- le problème est un problème d'optimisation sous contraintes
- les connaissances sont volumineuses, évolutives et difficiles à obtenir
- il existe un fort besoin pour des outils d'aide destinés aux non- spécialistes.

Par conséquent, nous avons pu transposer directement l'expérience acquise grâce au développement du système de conseil en épargne à la réalisation d'un outil d'aide à l'installation d'UNIX.

Effectivement, cette étude nous a permis d'exhiber les principaux problèmes rencontrés lors de la réalisation d'un système expert, à savoir :

- la définition du domaine d'expertise. Cette définition doit être extrêmement précise pour éviter d'avoir une base trop importante de connaissances. L'installation d'UNIX est un domaine suffisamment limité et structuré, par conséquent nous n'avons pas eu à restreindre le domaine d'expertise lors du développement.

- la disponibilité des expert. En effet, les experts sont très occupés et souvent ils ne peuvent consacrer beaucoup de temps au transfert d'expertise ce qui ralentit parfois le travail. Nous ne connaissons pas ce type de problèmes car nous sommes nos propres experts.

- Le choix de l'environnement de développement. Si l'outil utilisé n'est pas adapté au domaine d'expertise, le système expert risque fort d'être un échec, car la représentation des connaissances avec un modèle inadéquat est longue et rend les modifications très difficiles. C'est pourquoi nous avons sélectionné l'environnement de développement que nous utilisons avec le plus grand soin.

  Les critères les importants dans le choix d'un environnement de développement :

- les modèles de représentation des connaissances disponibles et les possibilités de les marier

- les stratégies du moteur d'inférences

- le degré de paramétrisation offert à l'utilisateur pour fixer le comportement de l'outil.

4 LA RECHERCHE D'UN ENVIRONNEMENT DE DEVELOPPEMENT DE SYSTEMES EXPERTS

Pour modéliser efficacement les connaissances utilisées par un ingénieur système pendant l'installation d'UNIX, nous avons cherché un environnement de développement de systèmes experts hybride, c'est-à-dire qui offre les deux paradigmes de programmation : objets à base de frames disposant de l'attachement procédural et règles de production avec variables. En effet, comme lors de l'installation d'UNIX sur une machine nous travaillons avec des entités concrètes : les composants matériels et logiciels du système et comme le nombre de chaque composant varie d'une installation à une autre, c'est l'utilisation simultanée d'objets et de règles avec variables qui convient le mieux à la représentation de ce genre de connaissances. Nous avons aussi décidé de représenter les connaissances avec des objets et des règles d'une part pour éviter de rencontrer à nouveau des difficultés identiques à celles rencontrées pendant le développement du système expert en épargne, et d'autre part car il est reconnu que l'utilisation simultanée de ces deux modes de représentation des connaissances facilite la modélisation des connaissances.

Or l'installation d'UNIX est un domaine d'expertise où les connaissances évoluent rapidement. Si les principes d'installation d'UNIX et les informations nécessaires varient très peu, les annonces de la disponibilité de nouveaux matériels et de nouveaux logiciels sont fréquentes. L'environnement de développement doit donc permettre une prise en compte facile des nouveaux composants. C'est possible avec un générateur de systèmes experts hybride. Il suffit, pour traiter un nouvel élément, de créer soit une nouvelle instance si ce composant correspond à une classe déjà définie, une nouvelle classe s'il n'appartient à aucune des classes existantes. Cependant, si cette nouvelle classe possède des caractéristiques communes avec une classe existante le mécanisme d'héritage des langages orientés objets permet alors de la définir comme une sous-classe. Les comportements de ce nouvel élément sont représentés, si nécessaire, par de nouvelles règles et de nouvelles procédures .

Nous avons donc cherché un générateur de systèmes experts répondant aux exigences que nous venons de définir. Comme nous avons également souhaité, pour des raisons de rentabilité de l'investissement, disposer d'un outil disponible sur une des machines du département informatique de l'Ecole des Mines de Saint-Etienne, à savoir un SPS9 de BULL, nous avons finalement retenu l'environnement de développement de systèmes experts KOOL de BULL.

KOOL est un environnement de développement de systèmes experts, écrit en LE_LISP, constitué d'un langage orienté objets à base de frames et d'un système à base de règles avec variables. Comme il a été développé en LE_LISP, les démons et les méthodes KOOL s'écrivent en LE_LISP, le cogniticien dispose ainsi de toute la puissance de la programmation fonctionnelle de LE_LISP pour représenter les connaissances de type procédural de l'application. De plus, KOOL répond totalement au critère d'uniformité que nous avons défini pour les langages objets de l'intelligence artificielle, c'est-à-dire que les classes et même les règles KOOL sont des objets. Les classes et les règles peuvent être modifiées dynamiquement par le programme. Cette facilité offre une grande souplesse au concepteur pour modéliser le raisonnement des experts.

5 LA REALISATION DU SYSTEME EXPERT POUR INSTALLER UNIX

L'environnement de développement de systèmes experts KOOL s'est révélé très agréable à utiliser et parfaitement adapté à la représentation des connaissances du domaine d'expertise retenu : l'installation d'UNIX. Le problème avec un outil de cette qualité n'est pas de réussir à représenter les connaissances dans le modèle imposé par l'outil mais plutôt de déterminer quel modèle utiliser parmi les différents modèles disponibles.

Notamment le choix entre les démons et les règles est parfois difficile. Par exemple, doit-on utiliser une règle ou un démon *WhenFilled* pour définir la taille d'un disque en fonction de son modèle? Nous avons décidé de privilégier l'utilisation des règles car, si la valeur d'un attribut est donnée par une règle, KOOL affiche lors du déclenchement de la règle sa forme externe, et peut, par la suite, expliquer comment il a déduit cette valeur. A l'opposé,

KOOL est incapable de fournir le moindre renseignement sur une valeur qui a été calculée par un démon.

En fait , comme notre système expert est implanté sur une machine dédiée différente de la machine à installer, la principale difficulté que nous avons recontrée a été de modéliser clairement le déroulement de l'installation d'UNIX réalisée avec notre système expert. Finalement, nous avons été contraints, parce que certaines actions doivent impérativement être réalisées en mode mono-utilisateur, et d'autres sur la machine à installer, de découper l'installation en trois phases :

- tout d'abord, notre système expert dialogue avec l'utilisateur pour saisir les caractéristiques de la machine à installer, détermine les caractéristiques du futur système UNIX et génère les fichiers de commandes pour l'installer ainsi que ses fichiers d'administration

- ensuite sur la machine où se trouve le système expert, en mode mono-utilisateur, nous exécutons un fichier de commandes généré par la phase précédente qui assure la création sur un disque d'une partition identique à la future partition système de la machine à installer, la copie du système sur cette partition et la sauvegarde de cette partition sur un support magnétique de distribution (cartouche, bande ou disque amovible)

- enfin sur la machine à installer, toujours en mode mono-utilisateur, nous copions sur le disque système le contenu du support magnétique généré à l'étape précédente, puis nous lançons un fichier de commandes créé par le système expert qui, si c'est nécessaire, formate les disques, crée les partitions, les systèmes de fichiers, les répertoires... et installe les logiciels complémentaires.

Chapitre 5

UN SYSTEME EXPERT DE CONSEIL EN PLACEMENT

1 PRESENTATION DU PROJET

1.1 LES OBJECTIFS

L'équipe Intelligence artificielle du département Informatique Appliquée de l'Ecole des Mines a développé pour la Société Lyonnaise de Banque une maquette de système expert de conseil expert en placement. Le travail a duré dix mois, de novembre 1985 à septembre 1986. De nombreuses expertises de la profession bancaire, comme le choix des moyens de financement pour un particulier ou une entreprise, l'identification des entreprises pouvant être introduites sur le second marché boursier, l'aide aux particuliers dans le choix de produits d'épargne, se prêtent à une approche système expert. Parmi ces domaines d'expertise, nous avons retenu le conseil dans le choix d'un ou plusieurs produits d'épargne adaptés à un client en nous limitant au traitement des placements courants, de quelques milliers ou dizaines de milliers de francs, pour les raisons suivantes :

- la complexité du domaine est suffisante pour que l'expérience soit significative, mais reste relativement limitée pour permettre une réalisation rapide.

- les sommes envisagées sont celles qui sont le plus fréquemment traitées dans l'ensemble du réseau bancaire, l'utilisation d'un système expert permettra donc de faire bénéficier le plus grand nombre de clients des compétences des spécialistes de la banque.

- la disponibilité d'experts motivés.

Le problème que le système expert doit résoudre se pose donc en ces termes : pour un client caractérisé par les renseignements suivants :

- sa situation familiale, son âge, sa tranche marginale d'imposition

- le résumé de son patrimoine mobilier et immobilier

- la somme disponible

- ses objectifs

déterminer le ou les produits

- les mieux adaptés à sa situation (en le convainquant)
- qu'il accepte de prendre

A contrario, le système expert ne doit pas :

- proposer des produits qu'il ne peut cumuler, auxquels il n'a pas accès légalement
- proposer des produits déconseillés dans sa situation (fisc, érosion monétaire ...).

Le but de cette étude était de démontrer la validité de l'approche système expert dans le domaine bancaire et d'exhiber les problèmes rencontrés lors du développement d'un système expert dans le domaine. Nous avons donc décidé de travailler sur un micro-ordinateur avec un progiciel de développement de systèmes experts, car cette solution présente les avantages suivants :

- un investissement faible
- une autonomie de développement et de test
- une mise en oeuvre rapide
- une interface simple et conviviale pour des non-informaticiens.

Le développement a été effectué sur un IBM/PC (avec une mémoire de 512 Ko et un disque dur de 10 Mo) avec le logiciel Intelligence Service de GSI-TECSI.

1.2 LE DEROULEMENT DE L'ETUDE

Une fois le domaine d'expertise choisi et l'outil de développement sélectionné, la Société Lyonnaise de Banque a désigné quatre experts pour participer au transfert de connaissances, à savoir :

- un chef d'agence pour représenter les intérêts des "exploitants de base"
- deux responsables de gestion de patrimoine dans des succursales : deux experts du domaine
- un membre de la direction Epargne et Gestion Institutionnelle pour tenir compte des orientations stratégiques de la banque en matière d'épargne

Ensuite, le transfert d'expertise s'est déroulé en trois phases successives :

phase 1 : établissement d'un vocabulaire commun aux experts de la banque et aux ingé-

nieurs cogniticiens, définition précise du domaine d'expertise et définition des objectifs du système expert.

phase 2 : définition d'une grille des caractéristiques des différents produits d'épargne proposés par la SLB, caractérisation de chaque produit selon la grille précédente, développement d'une première maquette.

phase 3 : critique ou validation du comportement des versions successives de la maquette.

1.2.1 Phase 1

Les premières réunions ont été consacrées d'une part à l'établissement d'un vocabulaire de base commun aux deux groupes participant au transfert de connaissances : les experts de la banque et les ingénieurs cogniticiens, et d'autre part à la définition plus précise du domaine d'expertise et des objectifs du système expert.

Nous avons tout d'abord présenter aux experts les possibilités et les particularités de l'intelligence artificielle, et plus spécialement des systèmes experts, par rapport à l'informatique traditionnelle, puis le formalisme qui sera utilisé dans le cadre de cette étude pour modéliser les connaissances : la logique des propositions.

Ensuite les experts de la Société Lyonnaise de Banque nous ont exposé les différents paramètres à considérer pour sélectionner un produit d'épargne, à savoir :

- les caractéristiques du client (situation familiale, profession, patrimoine,...)
- les besoins du client
- les caractéristiques des produits de la banque
- les connaissances fiscales
- les connaissances financières
- les intérêts commerciaux de la banque

Différentes approches sont possibles pour traiter toutes ces informations et établir un diagnostic :

- partir des besoins du client et déterminer le produit qui les satisfait le mieux.
- partir des caractéristiques des produits et déterminer les clients auxquels ils conviennent.
- réaliser un bilan patrimonial.

Les experts ont également exprimé des souhaits différents quant aux objectifs du système expert, l'un d'eux notamment aurait aimé le développement d'un outil destiné à l'assister dans ses décisions. Le domaine envisagé s'est donc révélé très ample et fortement évolutif, et nous avons dû restreindre le sujet et les objectifs de l'étude.

Parmi les différentes approches possibles, nous avons retenu l'approche qui part des besoins exprimés par le client et essaie de déterminer le ou les produits les mieux adaptés. Elle nous a semblé la plus intéressante parce qu'elle permet de structurer le domaine en quatre modules :

- constitution d'un capital
- défense d'un capital
- consommation d'un capital
- transmission d'un capital

qui correspondent aux quatre grands types de besoins que peut avoir un client.

Dans un premier temps, nous avons décidé de nous limiter à l'étude de la faisabilité, au développement et à la validation d'une maquette traitant la constitution d'un capital dont le but est d'éviter les erreurs de diagnostic.

1.2.2 Phase 2

Le travail s'est poursuivi par la définition des caractéristiques des principaux produits rencontrés dans la situation de constitution d'un capital (plan d'épargne logement, compte d'épargne logement, livret d'épargne retraite, CODEVI, compte d'épargne en actions, ...). Une fois en possession de toutes les caractéristiques des produits, nous avons pu commencer le développement du système expert.

Mais nous n'avons pas pu soumettre la première maquette aux critiques des experts avant que celle-ci ne traite presque la totalité de la constitution du capital. Les experts avaient de grosses difficultés à travailler sur une partie réduite du domaine (un ou deux produits isolés), car les cas théoriques examinés étaient alors trop éloignés des cas réels auxquels ils sont confrontés. Ceci nous a conduits à développer une première version, couvrant l'ensemble du domaine, à partir des connaissances que nous avions déjà recueillies avant de la présenter aux experts.

1.2.3 Phase 3

La présentation de la première maquette du système expert aux experts marque le début de la troisième phase du développement :

la validation progressive et l'affinement de cette première maquette.

Les différentes études de cas soumises par les experts à la maquette permettent de mettre en évidence les principales limites et faiblesses du produit, et ce, tant sur le fond (pertinence, pérennité, exhaustivité des connaissances, fidélité des raisonnements), que sur la forme (présentation, langage, documents imprimés, ...) et conduisent à élaborer en conséquence de nouvelles requêtes au cahier des charges.

Il a en particulier été décidé de :

- mettre en évidence sur un exemple particulier (PEL), la possibilité d'associer au diagnostic d'un produit, les procédures s'y rapportant (calcul du rendement, du droit au prêt, du plan de remboursement, affichage et impression d'une fiche produit, ...)

- mettre en oeuvre au début de chaque session, un questionnaire préalable d'identification du client (âge, situation, patrimoine, ...), afin d'orienter au plus vite et plus précisément le diagnostic vers l'un des 4 domaines précités.

- regrouper sur un même document les renseignements obtenus au début d'une session, et les propositions de produits déduites par le système expert pour fournir une trace du diagnostic.

Au terme de cette étude, la maquette de système expert traite à peu près complètement la situation de constitution d'un capital, et aborde dans les grandes lignes celle de défense. Si elle est encore un peu élémentaire, elle permet à n'en pas douter de conclure à la pertinence et à la faisabilité de l'approche Système Expert dans un domaine aussi vaste, aussi nuancé et évolutif. Elle apparaît comme un premier pas encourageant pour la réalisation sinon dans un premier temps d'un produit en vraie grandeur à utilisation banalisée, du moins pour celle d'un outil d'enseignement assisté par ordinateur, et d'un outil d'aide à la structuration et à l'archivage dynamique de la connaissance des experts.

Le travail effectué permet également de conclure à la faisabilité d'outils annexes au diagnostic, tels que les procédures de calcul, les outils d'impression (fiches produits, bilan de session, etc.), même si tous ces aspects ne sont pas intégrés à la maquette actuelle.

1.2.4 Méthodologie dégagée

D'un point de vue méthodologique, après l'expérience acquise dans cette étude nous pensons qu'il est important d'insister sur les points suivants :

- pendant la phase préliminaire essayer de définir le mieux possible le domaine d'expertise.

- commencer à travailler avec un seul expert pour éviter que les réunions se transforment en débats entre spécialistes où les cogniticiens ont extrêment de mal à extraire les connaissances fondamentales. Cependant après la réalisation d'une première version, il est nécessaire de la faire valider par plusieurs experts pour aboutir à une acceptation par l'ensemble de la profession. Ainsi le système expert devient représentatif de la culture d'une société, et non pas du mode de pensée d'un des spécialistes de cette société. Son introduction sera alors grandement facilitée.

- parvenir à présenter le plus rapidement possible une ébauche suffisamment complète du système expert pour que ses déductions ne soient plus triviales. Dès lors, nous avons constaté une forte augmentation de l'efficacité du travail avec les experts. En effet, il est beaucoup plus facile d'obtenir des experts une confirmation ou une infirmation de la justesse du raisonnement du système expert sur des cas réels que de leur proposer des cas théoriques et d'essayer d'exhiber leur mode de pensée. Cette nécessité de pouvoir présenter rapidement un prototype explique l'utilisation des environnements "quick and dirty" de prototypage en intelligence artificielle.

Maintenant que nous avons rappelé le cadre et les buts de cette étude, nous allons présenter les caractéristiques du générateur de systèmes experts utilisé, puis nous exposerons les problèmes que nous avons rencontrés du fait de l'emploi de cet outil.

2 LES CARACTERISTIQUES DU PROGICIEL DE DEVELOPPEMENT DE SYSTEMES EXPERTS

2.1 MODELISATION DES CONNAISSANCES

Intelligence Service est construit autour d'un moteur d'inférence de type 0+, c'est-à-dire que les connaissances peuvent être représentées sous forme de propositions logiques, et aussi sous forme de faits prenant un ensemble discret de valeurs comme :

```
la situation de famille = "célibataire", "marié" ou "veuf"
```

et sous forme de faits à valeur numérique comme :

```
le plafond du CEA = 7000.
```

A chaque fait est associé une option qui précise le comportement du moteur d'inférence vis-à-vis de ce fait pendant une résolution. Les options possibles sont :

- affichable
- demandable
- mixte
- invisible
- visible

Le moteur d'inférence d'Intelligence Service est capable de fonctionner soit en chaînage arrière soit en chaînage avant. Pour développer ce système expert de conseil en épargne, nous avons utilisé le chaînage-avant.

2.2 STRATEGIE DE RESOLUTION DE CONFLITS

En chaînage avant, Intelligence Service travaille par saturation, donc il déclenche toutes les règles activables. Quand toutes ces règles ont été déclenchées, il cherche à obtenir des informations supplémentaires, en interrogeant l'utilisateur sur la valeur des faits demandables, pour pouvoir déclencher de nouvelles règles. Cette recherche est conduite avec la stratégie suivante :

> Intelligence Service calcule pour toutes les règles de la base de connaissances le rapport entre le nombre de faits connus en prémisse de la règle et le nombre total de faits en prémisse. Puis il sélectionne la règle ayant le rapport inférieur à un le plus élevé. Si plusieurs règles obtiennent le même ratio, il retient celle qui apparaît en premier dans la base de connaissances.

2.3 MONOTONIE

Le moteur d'inférence d'Intelligence Service est un moteur monotone, c'est-à-dire qu'il travaille de façon irrévocable. Quand un fait reçoit une valeur, il la garde jusqu'à la fin de la session en cours, il est impossible d'écraser cette valeur par une autre. Si jamais on essaie, Intelligence Service affiche un message d'erreur et s'arrête.

2.4 APPEL DE FONCTIONS EXTERNES

Intelligence Service permet d'appeler dans la partie conclusion des règles des fonctions externes pour réaliser des calculs. Il n'y a pas de langage privilégié pour développer ces fonctions. Le passage de paramètres entre Intelligence Service et une fonction est réalisé par l'intermédiaire de deux fichiers texte. Intelligence Service écrit les valeurs des faits à fournir à la fonction dans un fichier que doit lire cette fonction. En sens inverse, cette dernière écrit

dans un autre fichier les résultats qu'Intelligence Service lira et affectera aux faits correspondants.

2.5 INTERFACE HOMME-MACHINE

Grâce à son système de fenêtres et de menus Intelligence Service offre à l'utilisateur une interface très conviviale et facile à maîtriser. Par contre Intelligence Service impose certaines limites à la formulation des faits. Notamment la longueur totale de la chaîne de caractères représentant un fait ne peut pas dépasser soixante caractères. Et si l'ingénieur cogniticien peut fixer la forme négative d'un fait, il ne maîtrise pas la manière dont le système va interroger l'utilisateur, en effet à chaque type de fait correspond une formulation de l'interrogation et une seule.

3 LES PROBLEMES RENCONTRES

3.1 REPRESENTATION DES CONNAISSANCES

Intelligence Service est un moteur d'ordre 0+, il n'offre pas la possibilité d'utiliser dans l'écriture des règles des variables, et des prédicats du premier ordre. Cette absence de variables conduit à la multiplication du nombre de règles dans la base de connaissance. Par exemple, pour tous les produits traités dans la maquette nous avons écrits une règle de la forme :

SI étudier produit_X
et la somme à placer sur produit_X >= le versement minimum d'accès au produit_X
alors proposer ouvrir produit_X

Comme la maquette considère trente-trois produits, il y a trente-trois règles de ce style, alors qu'avec un moteur d'inférence utilisant des variables, une seule règle aurait permis de traiter tous les cas, d'où une base de règles plus concise et plus facile à maintenir.

3.2 STRATEGIE DE RESOLUTION DE CONFLITS

Dès que la maquette a été suffisamment développée pour traiter quelques produits, nous avons découvert que dans certaines situations le système expert avait un comportement très différent de celui d'un expert humain. Par exemple :

- pendant l'étude d'un plan d'épargne logement, il interrogeait brusquement l'utilisateur sur des faits concernant le compte d'épargne en actions, ou le livret d'épargne retraite

- sachant déjà que le client acceptait les risques liés aux actions, il demandait : "est-ce que le client accepte malgré tout le risque des actions? ".

Ce comportement surprenant était provoqué par la seule stratégie de sélection des règles offertes par le progiciel utilisé. Cette heuristique extrêmement simple ne permet pas de reproduire le mode de raisonnement d'un expert dans ce domaine. En effet, un expert commence par recueillir quelques renseignements, puis à partir de ces premières informations s'oriente vers un produit, et ensuite essaie de vérifier si ce produit satisfait le client en utilisant les seules règles qui concernent ce produit. Or, à chaque cycle, le moteur d'inférences d'Intelligence Service examine toutes les règles de la base de connaissances et toutes sont potentiellement déclenchables. Il n'existe pas de moyens qui permettent de focaliser la recherche sur, par exemple, les règles qui traitent du plan d'épargne logement. Pour que la maquette ait un comportement semblable à celui d'un expert, nous avons dû rajouter des faits, des règles et jouer sur les options des faits (visible, invisible, affichable, demandable) pour contrôler la résolution en limitant artificiellement les règles déclenchables à chaque cycle. Typiquement le contrôle que nous avons installé vise à éviter que pendant l'étude du plan d'épargne logement le système interroge l'utilisateur sur des faits concernant le compte d'épargne en actions, ou le livret d'épargne retraite. Nous essayons aussi d'éviter que le système pose des questions redondantes à l'utilisateur, par exemple :

> lorsque le système sait déjà que le client accepte le risque des actions, il ne doit pas demander : "est-ce que le client accepte malgré tout le risque des actions?"

Mais ce contrôle supplémentaire a entraîné une augmentation non négligeable du volume de la base de règles, et ce qui est beaucoup plus grave, **la disparition de l'indépendance des règles** . Or l'indépendance des règles est primordiale dans le développement d'un système expert, car c'est elle qui garantit la facilité d'évolution par ajout ou suppression sans conséquence sur l'environnement. La perte de cette indépendance est donc une des raisons qui font que l'amélioration de la maquette est maintenant devenue très pénible. Vu la complexité de la maquette nous maîtrisons mal les conséquences d'une modification.

Intelligence Service offre la possibilité de travailler en chaînage avant ou en chaînage arrière. Nous avons développé un contrôle pour le chaînage avant, le mode de fonctionnement du moteur d'inférence le mieux adapté au domaine choisi, de ce fait nous ne pouvons plus utiliser le chaînage arrière, et l'opérateur n'a plus la possibilité de vérifier si un produit convient à un client.

Il faut noter qu'Intelligence Service ne possède pas la notion de contexte qui permet de structurer la base de règles en entités indépendantes, et celle associée de métarègles pour déterminer le contexte dans lequel le système doit travailler. Si cela avait été le cas, nous aurions pu beaucoup plus facilement contourner la faiblesse de la stratégie de choix des règles d'Intelligence Service. Une solution à ce problème peut consister à découper la base en plusieurs petites bases et à charger uniquement la base utile à un instant donné, malheureusement ce n'est pas réalisable avec Intelligence Service.

3.3 MONOTONIE

Le comportement monotone d'Intelligence Service nous a principalement gênés dans l'établissement de propositions comportant plusieurs produits, et notamment dans la gestion de la somme disponible après l'utilisation partielle du capital initial apporté par un client. Si nous avions utilisé le fait réel LA SOMME DISPONIBLE nous aurions pu au maximum enchaîner le diagnostic de deux produits différents. Pour s'affranchir de cette limite, nous réalisons la gestion de la somme disponible par une procédure externe au moteur d'inférences qui lit et écrit un fichier. En conséquence, l'information somme disponible n'intervient pas dans la résolution, et n'est présentée à l'utilisateur qu'à titre indicatif.

Cette impossibilité de modifier la valeur des faits nous a également interdit d'étudier, au cours d'une même résolution, le même produit dans des contextes différents, ou invalider des affectations déjà réalisées. Un exemple typique de ce problème est le livret d'épargne retraite qui peut être proposé soit pour constituer un capital en vue de la retraite, soit pour obtenir une réduction d'impôt. Si au début du diagnostic nous l'examinons dans le but d'obtenir une réduction d'impôt, nous ne pouvons plus par la suite le réétudier pour la préparation à la retraite.

Avec un moteur autorisant les raisonnements non-monotones, c'est-à-dire un moteur qui permet de modifier la valeur d'un fait au cours de la résolution d'un problème, nous n'aurions pas rencontré de difficultés pour gérer la somme disponible et pour étudier un produit avec des objectifs différents.

3.4 APPEL DE FONCTIONS EXTERNES

Le mécanisme d'interface avec des fonctions externes ne nous a pas véritablement handicapés, mais nous aurions aimé disposer d'une interface plus souple. Par exemple, tous les renseignements saisis au début d'une session de consultation sont transformés en faits d'Intelligence Service en conclusion de la première règle déclenchée par le moteur d'inférence en chaînage avant. Mais ces renseignements pourraient provenir d'une base de données, il serait alors plus judicieux de n'y chercher la valeur d'un fait que lors de son utilisation. Cet accès à l'information uniquement au moment opportun est très facile à programmer avec l'attachement procédural, grâce au déclenchement de démons si-besoin.

3.5 INTERFACE HOMME-MACHINE

Les contraintes imposées par Intelligence Service dans l'expression des faits ont beaucoup indisposé les experts pendant la phase de validation de la maquette. Souvent ces derniers étaient bloqués par la formulation maladroite des dialogues, et n'arrivaient pas à faire abstraction de ce qui n'est qu'un problème de présentation et à se concentrer sur la logique du système. En effet nous avons été obligés d'employer des abréviations, des sigles et des expressions tels que :

le client ne désire pas aussi faire un investissement immobilie[1]!!!

pour financer des travaux immobiliers étudier EL

attention durée minimum pour réduction d'impôt 6 ans.

Par conséquent, il est primordial pour l'efficacité du transfert de connaissances que l'environnement de développement offre soit un module de traitement sophistiqué de la langue naturelle, soit la possibilité au concepteur de définir totalement les dialogues.

4 CONCLUSION

Le choix de l'outil de développement a donc une influence importante sur le déroulement du transfert d'expertise, nous pensons que plus il est souple, plus il offre de possibilités et plus le travail des cogniticiens est aisé même si la maîtrise en est plus difficile. Pour réaliser une application non triviale, il nous semble important que l'outil de développement offre les services suivants :

- la possibilité de structurer la base de connaissances, et de la modifier dynamiquement (ajouter des règles, des objets ou en retirer)
- la possiblité de fixer librement la stratégie de résolution de conflits
- l'utilisation de variables
- l'attachement procédural
- une interface utilisateur agréable permettant l'emploi du langage naturel.

1 le "r" qui manque à la fin d'immobilie est oublié volontairement car Intelligence Service limite la formulation des énoncés à soixante caractères

Chapitre 6

REPRESENTATION DES CONNAISSANCES NECESSAIRES POUR INSTALLER UNIX

1 INTRODUCTION

Grâce à l'étude des différentes étapes de l'installation d'UNIX, nous avons montré que pour la mener à bien l'ingénieur système UNIX doit posséder des connaissances très vastes et très variées. Rappelons brièvement quelques unes de ces connaissances. L'ingénieur système doit parfaitement connaître les caractéristiques des matériels constituant la configuration dont il a la charge, comme la taille des disques, l'unité de découpage de ces disques pour que les frontières des partitions coïncident avec des fins de cylindres, les types des voies asynchrones... Ensuite à partir de l'emploi prévu de la machine, il découpe les disques en partitions système, utilisateurs, installe l'arborescence des fichiers UNIX, crée les procédures de sauvegarde, définit les différents fichiers qui permettent à UNIX de dialoguer avec l'environnement, détermine les démons à lancer lors du démarrage de la machine, définit les utilisateurs et leurs privilèges. Toutes ces opérations dépendent fortement de la configuration matérielle, et possèdent également des relations étroites entre elles, de ce fait elles doivent être exécutées dans un ordre bien précis dont il est presque impossible de s'écarter sans risque d'erreur. Les erreurs les plus fréquentes sont l'accès à des fichiers inexistants ou inaccessibles, et l'installation de fichiers dans des répertoires inexistants suite à des erreurs de montage ou démontage de volumes. Enfin l'ingénieur système doit interpréter les messages d'erreurs et savoir comment réagir pour corriger ces erreurs.

Pour pouvoir réaliser notre système expert d'installation d'UNIX, nous avons été conduits à modéliser et structurer les connaissances de l'expert du domaine : l'ingénieur système. Ces connaisssances sont de deux sortes : les connaissances factuelles qui représentent la description de la configuration et les connaissances opératoires qui à partir de cette description permettent de définir l'organisation logicielle d'UNIX à mettre en place, puis les actions à entreprendre pour l'installer. Nous étudierons tout d'abord les connaissances factuelles et les connaissances opératoires, puis nous présenterons les modèles de représentation des connaissances utilisés et nous proposerons quelques exemples de représentations des connaissances nécessaires pour installer UNIX dans notre système expert.

2 LES CONNAISSANCES DE L'INSTALLATION D'UNIX

2.1 LES CONNAISSANCES FACTUELLES

Nous avons regroupé les connaissances factuelles en trois grandes catégories :

- le matériel composant la configuration
- l'utilisation de la machine
- les logiciels nécessaires à l'emploi envisagé.

2.1.1 Le matériel

Les caractéristiques matérielles de la configuration constituent les données de base sur lesquelles va être construite l'installation. Nous distinguons parmi celles-ci les caractéristiques du calculateur et celles de la périphérie. Un calculateur est caractérisé par :

- son unité centrale
    - la CPU
    - la taille mémoire
    - les processeurs auxiliaires éventuels (processeur de calcul en virgule flottante, de calcul vectoriel, de calcul symbolique)
- ses modules d'échange
    - modules d'échange disques
    - modules d'échange bande magnétique
    - modules d'échange lignes asynchrones
    - modules d'échange ETHERNET
    - modules d'échange X25
    - modules d'échange graphique

  Les modules d'échange peuvent soit être intégrés à la carte principale, soit se présenter sous forme de cartes additionnelles.
- ses disques. Un disque pour l'installation d'UNIX sur une machine est entièrement décrit par :

  - le numéro du module d'échange qui le pilote
  - son numéro
  - le mineur et les majeurs des fichiers spéciaux qui le représentent
  - le nom de ses fichiers spéciaux
  - son genre, c'est-à-dire dur, fixe, mobile ou disquette
  - son modèle
  - la taille de ses secteurs
  - son modèle de formatage
  - l'indication s'il est formaté ou non
  - sa taille
  - son unité de découpage des partitions
  - la liste de ses partitions
- ses lignes asynchrones.

La périphérie d'un ordinateur rassemble l'ensemble des éléments réalisant des entrées-sorties avec l'extérieur de l'ordinateur que nous installons. On y trouve entre autres :

- les terminaux
- les imprimantes
- les tables traçantes
- les scanners
- les mémoires d'images
- d'autres ordinateurs.

Pour un terminal, par exemple, nous avons besoin de savoir s'il est graphique ou alphanumérique, son modèle, à quelle vitesse il est configuré, dans quel mode d'émulation il est utilisé.

Bien entendu, certaines configurations peuvent comporter d'autres éléments aussi bien dans la périphérie que dans le calculateur, nous ne présentons ici que les composants les plus fréquents.

2.1.2 L'utilisation de la machine

L'emploi qui est fait d'une machine est totalement déterminé par les applications qui y sont installées, et ses utilisateurs. Pour l'instant, nous avons recensé six grandes classes d'applications :

- le développement de logiciels
- les communications
- la bureautique
- la CFAO
- la gestion
- l'exploitation de la machine.

Plusieurs applications peuvent naturellement coexister sur une même machine. Chaque application nécessite la présence d'éléments matériels et logiciels pour pouvoir fonctionner. Grâce aux connaissances déjà acquises sur les options matérielles et logicielles disponibles, nous contrôlons pour chaque application les logiciels qui peuvent être installés sur cette configuration. Si, par exemple, une machine possède seulement un module d'échange lignes asynchrones, les communications avec d'autres sites ne se feront que par *cu*, *uucp*, ou *kermit*, et il est donc inutile d'installer les logiciels réseaux. Par contre, l'installation de *uucp* est une des étapes les plus délicates de la génération d'un système UNIX.

En fonction des applications qu'ils désirent utiliser, les utilisateurs sont rassemblés en groupes possédant les mêmes droits. Ensuite à chaque utilisateur est attribué un nom de connexion, un répertoire d'accueil, et le programme qui sera exécuté lors de sa connexion.

2.1.3 Les logiciels

Enfin, il est nécessaire de préciser les logiciels employés dans chaque application. Pour installer un logiciel, nous avons besoin de connaître :

- des informations qui lui sont propres, comme :
    - le matériel et les logiciels indispensables à son bon fonctionnement (par exemple la base de données utilisée par une application de gestion)
    - l'espace qu'il occupe sur disque
    - la taille mémoire nécessaire à son exécution
- des informations qui dépendent de l'installation en cours, comme :

- dans quels répertoires de l'arborescence UNIX installer les divers fichiers

- les paramètres à initialiser (par exemple les tables de routage des logiciels réseaux) qui sont conservés dans les fichiers d'administration du logiciel à installer

- le nombre d'utilisateurs simultanés.

Comme pour les applications, nous vérifions si l'emploi des logiciels sera possible sur la configuration matérielle et logicielle déjà décrite. Si nous détectons une impossibilité, nous la signalons à l'utilisateur qui pourra agir en conséquence.

2.2 LES CONNAISSANCES OPERATOIRES

Ces connaissances représentent l'expertise de l'ingénieur système, nous en distinguons trois sortes :

- les déductions immédiates

- les principes d'installation

- les connaissances des commandes UNIX

2.2.1 les déductions immédiates

Ce sont les connaissances qui permettent à l'ingénieur système de déduire, à partir des caractéristiques matérielles et logicielles qu'un utilisateur non-expert est capable de fournir (informations figurant sur le bon de livraison), tous les renseignements nécessaires à l'installation d'UNIX, comme par exemple :

- si la configuration comprend un disque dma mobile de numéro x alors sa taille est 9792 blocs, son unité de découpage 16 blocs, son modèle de formatage dmam, et un disque dma fixe de numéro x+1 et de modèle de formatage dmaf lui est associé

- si une SM90 ne possède qu'un module d'échange lignes asynchrones, son numéro est 1

- si une SM90 ne possède qu'un module d'échange ETHERNET, son numéro est 15

- les exécutables communs à tous les utilisateurs, autres que ceux du système d'exploitation, sont à installer dans le répertoire */usr/local/bin*

- le kit de développement PASCAL de SMX occupe 939 blocs sur disque

2.2.2 Les principes d'installation

Une fois les renseignements indispensables à l'installation obtenus, l'ingénieur système va utiliser ces connaissances qui forment l'expertise proprement dite pour décider comment configurer le système pour satisfaire les besoins des utilisateurs. Ce sont ces connaissances que nous avons essayé de dégager dans la présentation de l'installation d'UNIX. Nous pouvons donner comme exemples de ces principes d'installation :

- si la configuration dispose de plusieurs disques, installer la partition système et la zone de swap sur le plus performant

- pour connecter une imprimante Laserwriter, il n'est pas nécessaire d'utiliser une voie V24 complète

- la taille de la zone de swap doit être comprise entre le double et le triple de la taille de la mémoire centrale disponible.

2.2.3 Les connaissances des commandes UNIX

La configuration déterminée, il ne reste plus qu'à l'ingénieur système à déclencher dans l'ordre convenable toutes les actions nécessaires pour mettre en place le système. C'est à ce moment qu'il fait intervenir ses connaissances des commandes UNIX ou, pour les commandes qu'il ne maîtrise pas parfaitement, ses connaissances de la documentation UNIX pour trouver les informations voulues. Par exemple, pour créer le fichier spécial qui correspond à la voie asynchrone 11, et fixer les droits d'accès des différents utilisateurs sur cette voie, il exécutera les commandes :

```
/etc/mknod /dev/tty11 c 1 17
/bin/chmod 622 /dev/tty11
```

Ou encore, le disque de numéro 300 est représenté par les fichiers spéciaux */dev/dsk/c3d0* et */dev/rdsk/c3d0* dont le mineur est 96. Par conséquent ces fichiers sont créés par les commandes :

```
/etc/mknod /dev/dsk/c3d0 b 0 96
/etc/mknod /dev/rdsk/c3d0 c 4 96
```

3 MODELISATION DES CONNAISSANCES

Maintenant que nous avons défini à la fois les connaissances utilisées lors de l'installation d'UNIX et les représentations possibles, nous allons présenter et justifier les représentations retenues pour chaque type de connaissances. Les choix effectués l'ont été pour nous éviter de rencontrer à nouveau des problèmes similaires à ceux que nous avons dû résoudre pendant le développement du système expert de conseil en placement pour la Société Lyonnaise de

Banque.

3.1 LES CONNAISSANCES FACTUELLES

Comme nous venons de le voir dans la partie précédente, lors de l'installation d'UNIX nous manipulons des entités concrètes comme des disques, des terminaux, des logiciels et nous nous intéressons seulement aux quelques caractéristiques de ces entités qui interviennent dans l'installation. L'analyse de ces connaissances nous a montré qu'elles sont très structurées, aussi avons-nous cherché une représentation des connaissances qui permette d'exprimer facilement cette organisation et de modéliser des entités concrètes. Des trois paradigmes de programmation, c'est la programmation orientée vers les objets qui convient le mieux pour représenter ce type de connaissances. Nous avons donc décidé de représenter les entités concrètes par des objets dont les attributs sont les caractéristiques utiles à l'installation. Car un langage objet permet une modélisation immédiate de ces entités.

Avec une vingtaine de classes, nous arrivons à décrire tous les principaux composants d'une machine comme les disques, les modules d'échange lignes asynchrones, les modules d'échange Ethernet, les terminaux, les imprimantes, les utilisateurs... Comme les classes représentent les fonctionnalités logiques et non les caractéristiques physiques de chaque composant, cette généralité nous permettra de modéliser aisément de nouvelles machines même si, pour l'instant, notre outil possède seulement l'expertise pour traiter les configurations des machines de l'Ecole des Mines de Saint-Etienne. Pour prendre en compte de nouveaux éléments, comme, par exemple, un module d'échange DR11 ou un module d'échange IEEE488 ou un terminal graphique, il nous suffira de rajouter une classe pour décrire chaque nouveau composant.

La machine sur laquelle nous installons UNIX est alors représentée par une instance de la classe MachineAInstaller, les valeurs des attributs de cet objet étant des instances des classes correspondant aux composants présents dans cette configuration.

Ainsi un disque est représenté par une instance de la classe *Disk* dont les attributs sont :

- *NumMED* : le numéro du module d'échange qui pilote le disque
- *Numero* : le numéro du disque
- *Mineur* : le mineur des fichiers spéciaux du disque, comme les majeurs des fichiers spéciaux représentant un disque sont constants : 4 pour le fichier de type caractère et 0 pour le fichier de type bloc, il est alors inutile de prévoir un attribut pour les mémoriser
- *NumDiskPhys* : le nom des fichiers spéciaux du disque dans les répertoires */dev/dsk* et */dev/rdsk*
- *Modele* : le modèle du disque
- *Genre* : le genre du disque

- *SecFormat* : la taille des secteurs physiques du disque
- *ModFormat* : le modèle de formatage du disque
- *Format* : l'indication si le disque est formaté ou non
- *Taille* : la taille du disque
- *PartUnit* : l'unité de découpage en nombre de blocs pour que les frontières des partitions correspondent à des fins de cylindres
- *Partition* : l'indication si l'utilisateur veut définir lui-même les partitions du disque
- *ListPart* : la liste des partitions du disque

Plus précisément la classe *Disk* est définie de la manière suivante :

```
{InstClass : Disk
 Super : Object

 <  -- NumMED : ()
          Mode : Mono
          Type : Number
          Askable : No
    -- Numero : ()
          Mode : Mono
          Type : Number
          Askable : No
         WhenFilled : [dem_NumDisk]
    -- Mineur : ()
          Mode : Mono
          Type : Number
          Askable : No
    -- NumDiskPhys : ()
          Mode : Mono
          Type : String
          Askable : No
    -- Modele : ()
          Mode : Mono
          Askable : Yes
          Question : {Quel est le modèle du disque * ?}
          WhenFilled : [dem_GenreDisk]
    -- Genre : ()
          Mode : Mono
          Askable : Yes
```

```
        Question : {Quel est le genre du disque * ?}
        WhenFilled : [demModDisk,dem_FormDisk]
    -- SecFormat : ()
        Mode : Mono
        Type : [256,1024]
        Askable : Yes
        Question : {Quelle est la taille (en octets) des secteurs physiques
                     utilisée pour le formatage du disque * ?}
    -- ModFormat : ()
        Mode : Mono
        Askable : Yes
    -- Format : ()
        Mode : Mono
        Type : [oui,non,|ne sais pas|,-]
        Askable : Yes
        Question : {Le disque * est-il formaté ?}
        WhenFilled : [dem_FrmtDsk]
    -- Taille : ()
        Mode : Mono
        Type : Number
        Askable : Yes
        Question : {Quelle est la taille (en blocks de 512 octets)
                      du disque * ?}
    -- PartUnit : ()
        Mode : Mono
        Type : Number
        Askable : Yes
        Question : {Quelle est l'unité de découpage (nombre de blocs)
                    pour que les frontières des partitions
                    correspondent à des fins de pistes ?}
        WhenFilled : [com_Formatage]
    -- Partition : ()
        Mode : Mono
        Type : [oui,non]
        Askable : Yes
        Question : {Voulez-vous créer manuellement des partitions
                     sur le disque * ?}
        WhenFilled : [dem_Partition]
    -- ListPart : ()
        Mode : Multi
        Type : Partition
        Askable : No
        MaxCard : 32 >}
```

De plus, nous ne connaissons pas a priori le nombre d'instances de chaque classe dont nous aurons besoin pendant une installation. Nous avons donc besoin d'un langage objet qui nous offre la possibilité de créer des objets dynamiquement, seulement en cas de nécessité. De cette manière nous évitons la déclaration d'un nombre arbitraire de structures d'accueil comme avec un langage traditionnel. Pour créer les instances d'une classe, nous utilisons des règles pour les modules d'échange et des démons pour toutes les autres. En effet, comme nous ne connaissons pas de machine comportant plus de deux modules d'échange du même type, les instances des classes représentant les modules d'échanges sont générées par deux règles en chaînage avant :

- une règle pour créer une instance
- une règle pour créer deux instances

Si la machine possède, par exemple, un module d'échange disque, l'instance le représentant est créée par le déclenchement de la règle :

```
{InstRule + : FR1

If
  Cal^NbMED = 1
Then
  (Do
      (Make *MED MED_1)
; 1 et 2 ne signifient rien ils sont là uniquement pour que getObjectValue
; ait 3 arguments
      (getObjectValue MED_1 1 2)
      (addValue Cal MED MED_1)
      (addOwner MED_1 InstIL)
  )
Extern
  {Il n'y a qu'un MED
    THUS on le crée et on demande les valeurs de ses attributs}
}
```

Par contre, pour tous les autres composants d'une machine notre système expert demande combien la configuration en contient, et à cet attribut est associé un démon qui exécute une boucle de création, de saisie et de calcul de certains des attributs de l'objet nouvellement créé. Les instances des disques sont, par exemple, créées par le démon si-ajout associé à l'attribut *NombreDisque* de la classe *MED*. Ce démon assure également la saisie contrôlée du numéro du disque.

Nous voulons aussi disposer de la possibilité de modifier les classes pendant une session, et plus particulièrement pouvoir définir les valeurs autorisées pour un attribut et ainsi contrôler très strictement les données fournies par l'utilisateur. Par exemple, en fonction du modèle de l'ordinateur nous voulons pouvoir définir les modèles des disques utilisées sur cet ordinateur. Pour ce faire il faut pouvoir définir le descripteur contenant les valeurs possibles pour l'attribut *Modele* de la classe *Disk* une fois le modèle de l'ordinateur connu. C'est possible grâce à la fonction *putClassValue* de KOOL qui permet de définir la valeur d'un attribut ou d'un descripteur d'attribut d'une classe, toutes les futures instances de cette classe hériteront alors de cette valeur. En revanche, les instances de cette classe créées avant l'appel de la fonction *putClassValue* héritent de cette nouvelle valeur si et seulement si la valeur de cet attribut n'a pas été redéfinie localement.

Comme, dans un premier temps, notre système interroge l'utilisateur pour recueillir un minimum d'information sur la configuration, nous ne devons accepter que des valeurs légales pour garantir l'exactitude des décisions de notre logiciel. Ce contrôle des saisies peut se faire de différentes manières, en associant à un attribut soit, comme nous venons de le voir, un descripteur contenant les valeurs autorisées, soit un démon qui sera exécuté pour obtenir sa valeur. Ce démon implémente alors le dialogue avec l'utilisateur et s'assure de la validité des réponses données par ce dernier. C'est pourquoi nous voulons un langage objet disposant de l'attachement procédural avec des démons si-besoin. Quand notre système expert interroge l'utilisateur pour obtenir, par exemple, le nombre de terminaux qui seront connectés sur la machine, il doit refuser toute réponse supérieure au nombre de voies V24 disponibles. Nous avons donc défini un démon *ToCompute askNT* pour l'attribut *NbTerm* de la classe *Peripherie* qui réalise la saisie du nombre de terminaux et refuse la réponse tant qu'elle n'est pas acceptable.

3.2 LES CONNAISSANCES OPERATOIRES

Si l'utilisation d'un langage objet permet de bien représenter les connaissances factuelles, elle ne permet pas une représentation aisée, donc facilement accessible et compréhensible par l'utilisateur de certaines connaissances opératoires. Nous allons maintenant présenter lesquelles de ces connaissances il est possible de modéliser dans un langage objet et lesquelles demandent un autre modèle de représentation.

Certaines connaissances opératoires s'expriment très bien avec un langage objet à base de frames disposant de l'attachement procédural. Grâce à l'utilisation de démons de type si-ajout qui se déclenchent immédiatement après l'affectation d'une valeur à un attribut d'un objet, nous propageons des informations dans la base de connaissances. Par exemple, si la configuration comprend un disque dma mobile de numéro 300, l'ingénieur système sait qu'il est toujours associé à un dma fixe de numéro 301. Pour créer l'instance de la classe *Disk* qui représentera le dma fixe 301, nous avons donc défini le démon *WhenFilled demModDisk* qui est associé à l'attribut *Genre* de la classe *Disk*. Quand l'objet courant représente le dma mobile 300, ce démon crée une nouvelle instance de la classe *Disk* pour le dma fixe 301 et remplit les attributs *NumMED*, *Numero*, *Modele*, *Genre* dont les valeurs sont déjà connues.

L'objectif de notre système expert est d'installer UNIX sur une machine, pour l'atteindre nous devons donc générer des fichiers de commandes UNIX qui réaliseront cette installation et les fichiers d'administration du futur système. Ces fichiers sont créés par des démons si-ajout. Ainsi, une fois que toutes les informations concernant un disque sont connues, elles sont traduites en commandes qui formateront ce disque, le découperont en partitions, créeront des systèmes de fichiers, et elles servent à créer les fichiers */etc/checklist* et */etc/mountlit*. De cette façon nous maîtrisons parfaitement leur enchaînement, et nous sommes sûrs que les actions seront réalisées au moment opportun, et qu'ainsi l'installation se fera sans erreur.

Par contre des connaissances comme l'affectation des voies asynchrones d'une machine aux différents périphériques de la configuration se prêtent mal à une représentation par des démons ou des couples messages-méthodes. Aussi avons-nous choisi de les représenter par des règles. Comme les connaissances factuelles sont modélisées sous forme d'objets, les règles doivent être capables d'accéder aux valeurs des attributs des objets. Une configuration comprend généralement plusieurs composants identiques, ainsi pour construire le fichier */etc/inittab* nous savons que :

```
si la voie tty X est inutilisée alors
il faut positionner l'entrée correspondante du fichier
/etc/inittab à off
```

et cette règle doit s'appliquer à toutes les voies restées libres. Mais le nombre de voies libres est différent pour chaque configuration, donc il est impossible de traiter chaque voie par une règle particulière, il faut pouvoir traiter toutes ces voies avec une seule règle. Cette exigence condamne l'emploi de règles en logique des propositions stricte et impose de travailler avec des règles de production avec des variables, le moteur d'inférences assurant l'instanciation des variables libres d'une règle avec les objets qui peuvent rendre cette règle applicable. La connaissance précédente se traduit en KOOL par la règle en chaînage arrière :

```
{InstRule - : BR27

If
  (UnKnown *Voie Utilisation)
Then
  *Voie^Utilisation = inutilisee
  *Voie^Vitesse = |9600|
  *Voie^Terminal = "un"
  *Voie^Inittab = off
Extern
  {La voie est inutilisée
    THUS la vitesse est fixée par defaut à 9600
     et l'entrée correspondante dans inittab est mise à off
     et le type du terminal est "un"} }
```

Car notre système expert travaille de la manière suivante. Il affecte, tout d'abord, au mieux, en fonction de leurs caractéristiques, une voie à chaque périphérique de la configuration, et ensuite il déclenche la règle BR27 pour traiter le cas des voies inutilisées.

De plus, comme nous voulons disposer de l'outil le plus souple possible, nous voulons non seulement accéder aux objets et à leurs attributs, mais aussi appeler des fonctions, notamment pour modifier le schéma d'une classe, créer un objet, construire une chaîne de caractères, calculer la valeur d'un attribut et ceci aussi bien en prémisse qu'en conclusion d'une règle. Par exemple, les types des disques possibles dans une configuration sont déterminés en fonction du modèle de l'ordinateur par une règle qui appelle en conclusion la fonction *putClassValue* pour le descripteur *Type* de l'attribut *Modele* de la classe *Disk.* Si la machine à installer est un SPS7/300, les disques possibles sont obtenus grâce à la règle :

```
{InstRule - : BR5
If
  Cal^Modele = |SPS7/300|
Then
  UC^CPU = M68020
  (putClassValue Disk Modele Type '(oneOf d570 MXT190 floppy))
Extern
  {La machine est un SPS7/300
    THUS son microprocesseur est un 68020
    et ses disques peuvent être un Vertex d570 ou un Maxtor MXT190
    ou un floppy}
}
}
```

Enfin, pour représenter des connaissances comme :

```
pour relier deux ordinateurs par une liaison asynchrone,
il vaut mieux utiliser une voie V24 complète
```

il faut pouvoir privilégier le déclenchement des règles qui traitent de l'affectation des voies V24 complètes par rapport au déclenchement des règles sur les voies V24 simplifiées, c'est-à-dire il faut pouvoir contrôler la stratégie de résolution de conflits du moteur d'inférences. Une solution possible consiste à associer à chaque règle une priorité, le moteur d'inférences sélectionnant dans l'ensemble de conflit la règle dont la priorité est la plus grande. Nous avons donc recherché un système à base de règles disposant d'un mécanisme de ce type. Nous avons représenté cette connaissance, pour une machine qui appelera la machine à installer, par les deux règles suivantes :

```
{InstRule - : BR42
If
  (UnKnown *Voie Terminal)
  (UnKnown *Voie Utilisation)
  *Voie^Genre = V24c
  *MachV24^Status = maitre
Then
  *MachV24^Voie = *Voie
  /*Voie^Terminal = *MachV24^TermEmu
  /*Voie^Utilisation = machine
  /*Voie^Inittab = respawn
  /*Voie^NomLien = (catenate "tty" *MachV24^Nom)
  /*Voie^Vitesse = *MachV24^Vitesse
Extern
  {Si on trouve une V24 complète inutilisée
    THUS on l'affecte à la machine à connecter
     et on positionne les attributs de cette voie.}
-- Interest : 100
}

{InstRule - : BR44

If
  (UnKnown *Voie Terminal)
  (UnKnown *Voie Utilisation)
  *Voie^Genre Diff V24c
  *MachV24^Status = maitre
Then
  *MachV24^Voie = *Voie
  /*Voie^Terminal = *MachV24^TermEmu
  /*Voie^Utilisation = machine
  /*Voie^Inittab = respawn
  /*Voie^NomLien = (catenate "tty" *MachV24^Nom)
  /*Voie^Vitesse = *MachV24^Vitesse
Extern
  {ATTENTION toutes les V24 complètes sont déjà employees
    j'utilise une V24 simplifiée ou une voie de maintenance
    pour connecter cette machine cela peut être source de
    problèmes}
}
```

Lorsque des voies V24 complètes et des voies simplifiées ou de maintenance sont libres, ces deux règles sont déclenchables. L'attribut *Interest* de la règle BR42, avec sa valeur de 100,

force le moteur d'inférences de KOOL à la déclencher avant la règle BR44. Ainsi tant qu'il restera des voies V24 complètes disponibles, notre système expert les utilisera pour connecter un autre ordinateur. En revanche, si les seules voies, encore libres, sont des voies simplifiées ou de maintenance, il déclenchera la règle BR44 pour en choisir une et préviendra l'utilisateur de l'existence d'un risque de problèmes.

Comme les principes d'installation (règles BR27 et BR42 par exemple) ne dépendent pas du modèle de la machine à installer, pour que notre outil d'aide puisse installer un nouveau modèle, grâce à l'indépendance des règles il suffit d'intégrer dans la base de connaissances les nouvelles règles qui représentent les déductions immédiates (règles BR5 par exemple) spécifiques à ce modèle.

La prise en compte d'un nouveau modèle impose également la mise à jour des démons qui représentent certaines des déductions immédiates et qui réalisent la génération des fichiers d'installation et d'administration en fonctions des particularités de ce modèle et de sa version d'UNIX. Pour résoudre ce problème, nous avons développé les démons qui dépendent du modèle de la machine à installer sous la forme d'une procédure de choix multiples qui appelle la procédure correspondant au modèle étudié. Ainsi nous pouvons facilement ajouter une nouvelle machine en introduisant une nouvelle étiquette dans les choix multiples et en écrivant les nouvelles procédures indispensables. Par exemple, afin de pouvoir installer SPIX sur un SPS9, nous avons dû écrire de nouvelles fonctions qui réalisent le formatage, le découpage des disques en partitions et le calcul des mineurs des fichiers spéciaux.

3.3 CONCLUSION

Installer UNIX sur un ordinateur est une tâche complexe et difficile qui demande des connaissances très diverses (caractéristiques des matériels, des logiciels et principes d'installation) et une grande expérience. La modélisation efficace de ces connaissances par un système expert est possible mais exige l'utilisation conjointe des trois paradigmes de programmation :

- les objets pour représenter les connaissances factuelles, c'est-à-dire les entités concrètes du domaine d'expertise

- les règles pour représenter les connaissances opératoires qui correspondent à des déductions logiques

- les procédures pour représenter les connaissances opératoires qui demandent une programmation impérative car elles correspondent aux actions à exécuter pour installer UNIX ou aux calculs d'informations nouvelles.

La sélection du générateur de systèmes experts KOOL, outil qui combine les trois paradigmes de programmation, nous a permis de représenter aisément les connaissances nécessaires à l'installation d'UNIX. En effet, KOOL est à la fois un langage objet complet et un système à base de règles performant, et comme il est écrit en LE_LISP, chaque fois que c'est nécessaire nous pouvons appeler une fonction LE_LISP pour représenter une connaissance algorithmique.

De plus, le mariage intime d'un langage objet et d'un système à base de règles avec la programmation fonctionnelle LE_LISP est parfaitement adapté à la représentation de l'expertise d'un ingénieur système UNIX car il permet de définir des granules de connaissances (objets, règles, procédures) indépendantes les unes des autres. Cette indépendance est très importante : c'est elle qui garantit que notre système pourra suivre l'évolution des connaissances qui est très rapide dans ce domaine d'expertise.

Chapitre 7

LA REALISATION

1 QUELQUES CARACTERISTIQUES DE KOOL

Au cours de cet exposé, nous avons déjà présenté le formalisme de KOOL pour définir les classes et les règles. Nous allons maintenant présenter les quelques caractéristiques de KOOL qui ont directement influencé la représentation des connaissances et le développement de notre système expert. Ces caractéristiques sont :

- la structure d'une base de connaissances KOOL
- les règles
- la stratégie de KOOL pour déterminer la valeur d'un attribut
- la fonction *koolStart*
- les fonctions prédéfinies de KOOL
- la fonction *getObjectValue*

Examinons dans le détail chacune de ces propriétés et leur influence sur la conception de la base de connaissances du système expert pour installer UNIX.

1.1 LA STRUCTURE D'UNE BASE DE CONNAISSANCES KOOL

Une base de connaissances KOOL est formée de quatre sous-ensembles :

- les classes
- les instances
- les règles
- les fonctions

Généralement, la définition d'une base de connaissances est conservée dans un fichier où sont définis les fichiers représentant chaque sous-ensemble. La déclaration de la base de connaissances de notre système expert pour installer UNIX est la suivante :

```
{KBase : InstKB
  -- ClassFiles : ["instsys.cla","inst.cla"]
  -- RuleFiles : ["inst.frul","inst.brul"]
  -- FunctionFiles : ["inst.ll","instsys.ll","inst.com","inst.voie",
                      "inst.perif","inst.spe","inst.part"]
  -- InstanceFiles : ["inst.ins"]
  }
```

1.2 LES REGLES

Kool permet de définir trois modes d'invocation pour une règle :

- le chaînage arrière
- le chaînage avant
- le chaînage mixte

Mais une règle déclarée en chaînage arrière n'est utilisée que lorsque le système cherche à établir un des faits qui apparaît dans ses conclusions. De la même manière, une règle en chaînage avant ne peut être déclenchée que pour déduire de nouvelles informations lorsque ses conditions sont vraies. Dans KOOL, les règles en chaînage arrière et les règles en chaînage avant constituent deux mondes totalement indépendants. Pour qu'une règle soit utilisée tantôt en chaînage arrière tantôt en chaînage avant, le concepteur de l'application doit spécifier comme mode d'invocation le chaînage mixte. C'est la valeur par défaut que KOOL associe au mode d'invocation d'une règle.

Les variables dans une règle KOOL s'écrivent *<*nom-de-classe*>, comme par exemple **Disk*, **Voie*. De cette façon, elles sont typées, ce qui signifie que, lors de l'évaluation d'une règle par le moteur d'inférences, l'instanciation d'une variable libre est limitée aux objets de la classe de la variable. Le typage des variables permet donc d'améliorer l'efficacité du moteur d'inférences et de garantir le déclenchement des règles seulement pour les instances d'une classe.

1.3 L'HERITAGE DANS KOOL

L'héritage dans KOOL est un héritage simple, à savoir qu'une classe ne peut posséder qu'une seule super-classe. En revanche, le mode d'héritage peut être défini pour chaque attribut d'une classe grâce au descripteur *Inheritance*. KOOL propose quatre modes d'héritage différents :

- *UNION* pour les attributs multivalués, la valeur héritée est l'union des valeurs définies dans le schéma d'instances de la classe et des valeurs héritées

- *STRICT* la valeur de l'attribut ne peut pas être redéfinie dans une sous-classe

- *DEFAULT* la valeur héritée par une instance est la valeur définie pour l'attribut dans la classe la plus proche de l'instance

- *NO* seules les instances directes héritent de la valeur définie dans la classe

Nous avons très peu exploité le mécanisme d'héritage car les classes que nous avons définies pour représenter les connaissances nécessaires à l'installation d'UNIX possèdent peu de caractéristiques communes. Seule la classe *Imprimante* possède deux sous-classes :

- la classe des imprimantes à interface série *ImpriSer*

- la classe des imprimantes à interface paralléle *ImpriPar*

En revanche, si KOOL avait disposé de l'héritage multiple, nous l'aurions beacoup plus utilisé, par exemple nous aurions modélisé tous les périphériques à interface série par une classe ayant deux super-classes : la classe des périphériques à interface série et la classe correspondant au type de périphériques. Ainsi la classe des imprimantes à interface série aurait hérité de la classe des périphériques à interface série et de la classe des imprimantes.

1.4 LA STRATEGIE DE KOOL POUR DETERMINER LA VALEUR D'UN ATTRIBUT

La valeur de l'attribut d'un objet est déterminée par KOOL lors de l'appel de la fonction *getValue*. Pour calculer cette valeur KOOL procède de la manière suivante :

- il regarde si la valeur est présente localement ou si elle est défine dans une des super-classes de la classe

- sinon il cherche dans le descripteur *Determine* de l'attribut le nom de la fonction à invoquer pour obtenir la valeur

Par défaut KOOL propose les fonctions *determineMono* pour un attribut monovalué et *determineMulti* pour un attribut multivalué. L'algorithme de ces deux fonctions est le suivant :

- elles essaient d'évaluer les règles en chaînage arrière où l'attribut apparaît en conclusion

- puis si la tentative précédente échoue, elles examinent le descripteur *ToCompute* et s'il contient le nom d'un démon elles le déclenchent

- si le démon n'existe pas ou ne fournit aucun résultat, elles regardent si l'attibut est demandable (son descripteur *Askable* a pour valeur *Yes*). Si c'est le cas, elles posent

une question à l'utilisateur (affichage à l'écran de la valeur du descripteur *Question*) et elles attendent sa réponse

- si l'attribut n'est pas demandable ou si l'utilisateur ignore la réponse, alors elles retournent la valeur par défaut (valeur du descripteur *Default* ou *Defaults*) si elle existe.

Si la stratégie complexe de ces procédures ne convient pas au domaine d'expertise à modéliser, le développeur peut définir une nouvelle procédure mieux adaptée. Pour développer le système expert pour installer UNIX, nous avons utilisé les procédures par défaut. L'architecture de sa base de connaissances résulte pour beaucoup de ce choix. En effet, la stratégie des procédures par défaut de KOOL nous a conduits à représenter le plus possible les connaissances opératoires par des règles en chaînage-arrière pour bénéficier de leur déclenchement lors des appels à la fonction *getValue*.

1.5 LA FONCTION KOOLSTART

La fonction *koolStart* est définie par le cogniticien qui réalise une application. Elle est obligatoire car elle contient les actions nécessaires au lancement de l'application, le plus souvent des appels à la fonction *getValue* pour déterminer les valeurs de certains attributs, et à l'exploitation des résultats. De cette façon, le cogniticien contrôle parfaitement le déroulement d'une session de consultation du système expert développé avec KOOL. En effet, chaque appel à la fonction *getValue* doit être considéré comme le point d'entrée dans la résolution d'un sous-problème, car il lance la recherche de la valeur d'un attribut avec la stratégie associée à cet attribut.

Dans le système expert pour installer UNIX, la fonction *koolStart* commence par demander des informations générales sur la machine à installer comme la marque, le modèle, le type de l'unité centrale, la taille de la mémoire. Ensuite, pour chaque classe de modules d'échange, elle en saisit le nombre présent dans la machine et exécute une phase de chaînage avant pour que KOOL déclenche la règle (cf paragraphe 3.1 du chapitre précédent) qui va créer les instances correspondantes et appeler pour ces instances la fonction *getObjectValue* qui détermine les valeurs des attributs demandables d'un objet. Après les modules d'échange, *koolStart* demande le nombre d'ordinateurs, d'imprimantes, de terminaux à connecter à la machine ce qui permet de déclencher des démons pour créer et déterminer les attributs des objets correspondants.

1.6 LES FONCTIONS PREDEFINIES DE KOOL

Nous avons déjà présenté deux fonctions prédéfinies de KOOL : *putClassValue* et *getValue*, en fait KOOL met à la disposition du développeur un ensemble de fonctions qui permettent de réaliser toutes les opérations possibles sur les classes, les objets, les attributs, les descripteurs et les règles depuis une fonction LE_LISP. Les fonctions les plus fréquemment utilisées sont :

- *addValue* pour ajouter une valeur à un attribut ou à un descripteur multivalué

- *askable?* pour savoir si un attribut est demandable
- *emptyConflictSet* pour forcer le déclenchement des règles présentes dans l'ensemble de conflit
- *getInstances* pour obtenir la liste des instances d'une classe
- *getValue* pour déterminer la valeur d'un attribut ou d'un descripteur
- *object?* pour tester si une variable LE_LISP représente un objet
- *putClassValue* pour fixer la valeur d'un attribut ou d'un descripteur dans le schéma d'instance d'une classe
- *putValue* pour fixer la valeur d'un attribut ou d'un descripteur d'un objet
- *userAttributeNames* pour obtenir la liste des attributs d'un objet

Grâce à cet ensemble de fonctions très puissantes, il est possible de construire un système expert qui suive une démarche semblable à celle d'un expert du domaine.

1.7 LA FONCTION GETOBJECTVALUE

Le but de notre système expert est de déduire la configuration du système UNIX à installer en fonction des caractéristiques matérielles et logicielles de la machine à laquelle il est destiné. Une consultation pour installer une machine débute donc par la saisie des valeurs des attributs des objets représentant ces caractéristiques, puis se poursuit par la déduction des valeurs des autres attributs. Par conséquent la détermination des valeurs des attributs constitue l'activité principale du système expert.

Comme la machine et son système sont représentés par des objets, nous avons développé la fonction *getObjectValue* qui admet comme paramètre l'objet dont on cherche les valeurs des attributs. Cette fonction détermine la liste des attributs de cet objet, puis pour les attributs demandables cherche à déterminer leur valeur. Si le type de l'attribut courant est une classe définie par le développeur, *getObjectValue* crée une instance de cette classe, détermine récursivement les valeurs de ses attributs avant d'affecter l'instance ainsi générée comme valeur à l'attribut courant. Pour les autres attributs elle appelle la fonction *getValue.*

De cette façon, nous n'avons pas à connaître les attributs de tous les objets et, avec une seule fonction, nous traitons toutes les classes de l'application. De plus, en ne déterminant que les valeurs des seuls attributs demandables nous maîtrisons le déroulement d'une session de consultation de notre système expert.

2 L'ARCHITECTURE GENERALE

Pour modéliser les connaissances factuelles, c'est-à-dire construire la structure de données qui accueille les informations caractéristiques des composants matériels et logiciels de la machine à installer, nous avons défini vingt-trois classes. Cette structure de données est organisée de la manière suivante :

- *Machine* la machine à installer
  - *Materiel* les composants matériels
    - *Calculateur* les composants matériels qui se trouvent dans l'armoire de la machine
      - *UniteCentrale* l'unité centrale
        - *Voie* la console système
      - *MED* les modules d'échange disques
        - *Voie* la voie de maintenance
        - *Disk* les disques
          - *Partition* les partitions
      - *MEV24* les modules d'échange lignes asynchrones
        - *Voie* les lignes asynchrones
      - *METH* les modules d'échange Ethernet
        - *PseudoTTY* les pseudo-terminaux
        - *Voie* les lignes asynchrones
      - *METH* les modules d'échange X25
      - *PAD* les voies logiques PAD
    - *Peripherie* les périphériques
      - *Term* les terminaux

        Voie la voie asynchrone
      - *ImpriSer* les imprimantes à interface série

Voie la voie asynchrone

- *ImpriPar* les imprimantes à interface paralléle
- *MachV24* les machines reliées par une voie asynchrone

Voie la voie asynchrone

- *Utilisation* les composants logiciels
  - *Application* les applications
    - *Logiciel* les logiciels
  - *Utilisateur* les utilisateurs

La hiérarchie qui apparaît dans ce schéma traduit seulement des liens attributs-valeurs, par exemple l'attribut *Disque* de l'objet *MED_1* (instance de la classe *MED*) reçoit comme valeurs les instances de la classe *Disk* qui représentent les disques gérés par ce module d'échange disques et l'objet *MED_1* est lui-même une valeur de l'attribut *MED* de l'objet *Cal* (instance de la classe *Calculateur*), etc...

2.1 LE DEROULEMENT D'UNE INSTALLATION D'UNIX AVEC NOTRE OUTIL

Une installation d'UNIX avec notre système expert, se déroule en trois étapes :

1. une session du système expert pour construire les fichiers d'installation et d'administration sur une machine différente de la machine à installer
2. création du système destiné à la machine à installer sur la machine où se trouve le système expert
3. la génération du système sur la machine à installer

2.1.1 La consultation du système expert

Quelque soit la configuration de la machine à installer, nous avons toujours besoin pour la représenter des sept objets suivants :

- *Mai* une instance de la classe *Machine*
- *Mat* une instance de la classe *Materiel*
- *Uti* une instance de la classe *Utilisation*
- *Perif* une instance de la classe *Peripherie*
- *Cal* une instance de la classe *Calculateur*

- *UC* une instance de la classe *UniteCentrale*

- *Csl* une instance de la classe *VOIE* qui représente la console système

Ces objets sont créés automatiquement lors du lancement du système expert, ils sont définis dans le fichier *inst.ins*. Ensuite le système expert détermine les caractéristiques des différents composants présents dans la configuration :

- les modules d'échange disques
    - pour chaque module d'échange les disques qu'il pilote
- les modules d'échange lignes asynchrones
    - les lignes asynchrones V24
- les modules d'échange Ethernet
    - les pseudo-terminaux
    - les lignes asynchrones éventuelles
- les modules d'échange X25
- les voies logiques PAD si la machine comprend un module d'échange Ethernet ou un module d'échange X25
- les ordinateurs connectés par liaison V24
- les imprimantes à interface série
- les imprimantes à interface parallèle
- les terminaux
- les applications
    - pour chaque application les logiciels utilisés
- les utilisateurs

Une fois toutes ces informations obtenues, le système expert peut définir le découpage des disques en partitions. Toutefois, l'utilisateur, s'il le désire, peut imposer un autre découpage. Les fichiers d'installation et d'administration du futur système sont construits par des démons *WhenFilled* dès l'obtention des informations nécessaires à leur exécution.

2.1.2 La création du système

Cette étape s'effectue en mode mono-utilisateur sur la machine où a été consulté le système expert. Sur cette machine, l'espace disque disponible doit être suffisamment important pour pouvoir y créer la partition où sera construite la partition système destinée à la machine à installer. La création de cette partition est réalisée par le fichier de commandes *shlPlanAct* qui contrôle l'enchaînement de toutes les actions nécessaires, à savoir :

- la création de cette partition par le fichier de commandes *shlMkafs*
- le montage de cette partition sur le répertoire d'installation
- la création sous le répertoire d'installation des principaux répertoires de l'arborescence UNIX par le fichier de commandes *utlRep*
- la copie des fichiers d'installation et d'administration par le fichier de commandes *utlCopi*
- la création des fichiers spéciaux par le fichier de commandes *shlFichSpe*
- la préparation du formatage des disques par le fichier de commandes *shlFormat*
- la sauvegarde de cette partition sur une bande ou une cartouche.

2.1.3 La génération du système

L'administrateur poursuit l'installation d'UNIX en démarrant la machine à installer à partir de la bande qui vient d'être créer et en la copiant sur le disque qui deviendra le disque système. Une fois cette copie effectuée, l'administrateur redémarre la machine en mode mono-utilisateur à partir du disque système nouvellement généré, puis il lance l'exécution du fichier de commandes *configPlanAct* qui appelle successivement :

- le fichier de commandes *configFormat* pour formater les disques
- le fichier de commandes *configMkapar* pour partitionner les disques
- le fichier de commandes *configMkafs* pour créer les systèmes de fichiers et leur répertoire d'accrochage, et monter les partitions.

L'architecture du système UNIX étant installée, il ne reste plus qu'à installer les options logicielles puis à sauvegarder une première fois les partitions du système avant de pouvoir démarrer la machine en mode multi-utilisateur et la mettre à la disposition des utilisateurs.

2.2 LES FICHIERS DE L'APPLICATION

Nous avons utilisé la systématique suivante pour distinguer les fichiers selon l'étape où ils interviennent :

- les fichiers *inst.** constituent la base de connaissances du système expert
- les fichiers de commandes *shl** sont exécutés pendant l'étape 2
- les fichiers de commandes *config** sont exécutés pendant l'étape 3

De plus, nous avons défini des petits utilitaires pour créer, détruire des fichiers, des répertoires, le nom de leur fichier commence par *utl.*

L'étude de nombreux systèmes UNIX nous a montré que plusieurs commandes d'initialisation sont communes à tous les systèmes. Nous avons donc créé des fichiers qui contiennent ces commandes, les fichiers *init**. Le système se contente alors d'y rajouter les commandes spécifiques à la machine à installer. Pendant l'étape 2, ces fichiers sont copiés, comme les fichiers *config**, dans le répertoire */etc* du futur système et deviennent ses fichiers d'administration.

3 L'ETAT ACTUEL

Dans son état actuel, le système expert est composé de :

- 20 classes
- 60 règles en chaînage arrière
- 11 règles en chaînage avant
- 91 fonctions LE_LISP
    - pour déterminer certaines informations
    - pour les traiter
    - pour construire les fichiers de commandes d'installation et les fichiers d'administration

Bien que le développement soit réalisé sur un SPS9, notre système expert est actuellement capable d'installer SMX V.2 sur des SM90 et des SPS7, car ce sont les machines que nous connaissons le mieux. La base de connaissance prend en considération les options matérielles et logicielles dont nous disposons à l'Ecole des Mines, à savoir :

comme modules d'échange :

- les modules d'échange 8 lignes asynchrones de TELMAT et BULL
- les modules d'échange disques SASI de TELMAT et BULL
- les modules d'échange disque SMD de HACKERS
- les modules d'échange ETHERNET de HACKERS
- les modules d'échange X25 de OST

et comme utilisations :

- le développement de logiciels
- les communications.

Notre système expert assure notamment :

- le formatage des disques
- le découpage des disques en partitions
- l'affectation des voies de communication
- la mise en place des utilisateurs
- l'installation des logiciels de communication

Comme la saisie des caractéristiques des applications et des logiciels n'est pas encore réalisée, le système ne peut pas proposer automatiquement une base de découpage des disques en partitions qui en découlerait. L'utilisateur est donc encore obligé de lui fournir les caractéristiques de toutes les partitions. En revanche, celles de la zone de swap sont déterminées à partir de la mémoire disponible sur la machine et de la partition système.

Nous avons également débuté la paramétrisation de la base de connaissances pour prendre en compte les différences qui existent entre l'installation de SPIX 931 sur un SPS9 et celle de SMX V.2 sur des machines à architecture SM90. Cela se traduit par la création de nouvelles règles et l'adaptation de certaines fonctions.

4 LES DIFFICULTES

Les difficultés que nous avons rencontrées pendant le développement de ce système expert-sont de deux sortes :

- les difficultés liées au domaine d'expertise à modéliser
- les difficultés liées au générateur de systèmes experts employé

Examinons chacun de ces types de difficultés.

4.1 LES DIFFICULTES LIEES AU DOMAINE D'EXPERTISE

Grâce au choix de KOOL comme environnement de développement, nous n'avons pas recontré de problèmes pour représenter les connaissances et la démarche d'un ingénieur système qui installe UNIX sur une machine. Nous avons apprécié de ne pas être confrontés à des difficultés semblables à celles que nous dû résoudre pendant le développement du système expert de conseil en épargne pour que le système expert se comporte presque comme un expert humain.

En fait, comme nous l'avons déjà exposé au chapitre 4, la seule véritable difficulté que nous avons rencontrée a été de réussir à bien formaliser le déroulement d'une installation avec notre système expert. Ce n'est que progressivement, au fur et à mesure du développement, que nous avons pris conscience des problèmes à résoudre du fait de l'exécution du système expert sur une machine différente de la machine à installer. L'emploi d'une machine auxiliaire modifie la procédure d'installation par rapport à une installation exécutée "manuellement" et nous avons dû la mettre au point.

L'idéal serait cependant que le système expert travaille sur la machine à installer, car il pourrait alors surveiller l'exécution des commandes d'installation. En définissant des stratégies de reprise en cas d'erreurs, nous construirions un logiciel qui représenterait totalement l'expertise d'un ingénieur système. Mais cela suppose de réaliser au préalable une pré-installation minimum pour permettre à notre système expert de s'exécuter, ou mieux de réussir à l'intégrer dans le programme de lancement de la machine ce qui suppose d'être capable :

- soit d'exécuter comme programme de lancement un programme aussi volumineux qu'un système expert
- soit d'en générer une version exécutable beaucoup plus concise.

4.2 LES DIFFICULTES LIEES AU GENERATEUR DE SYSTEMES EXPERTS

KOOL est un outil extrêment puissant mais complexe. Sa maîtrise demande donc de bonnes connaissances de chaque mode de représentation des connaissances et un certain temps d'apprentissage. Les principales difficultés pour maîtriser KOOL proviennent de l'attachement procédural et de la souplesse de KOOL. Ces difficultés sont :

- le choix entre un démon et une règle pour représenter une connaissance opératoire (cf. chapitre 4)

- la mise au point des programmes

- la rigueur dans l'écriture des programmes

4.2.1 Le choix de la meilleure représentation des connaissances

Nous pensons que chaque fois que c'est possible il vaut mieux représenter une connaissance opératoire par une règle. Car d'une part, une règle est beaucoup plus lisible qu'une fonction LE_LISP et permet de bénéficier du module explicatif associé au moteur d'inférences. De cette façon, le système expert peut plus facilement convaincre l'utilisateur du bien-fondé de son raisonnement. Et d'autre part, grâce à l'indépendance des règles et à leur bonne lisibilité, la maintenance d'une base de règles est plus facile que la maintenance d'un ensemble de procédures LE_LISP.

4.2.2 La mise au point des programmes

Il est difficile de suivre l'exécution d'un programme écrit avec des objets et des démons parce que cette exécution n'est plus séquentielle. En effet si un démon affecte une valeur à un attribut qui possède lui-même un démon *WhenFilled*, ce dernier s'exécute avant que l'exécution du premier se poursuive. Quand, dans un programme, les déclenchements de démons s'enchaînent en cascade, le développeur risque d'être confronté à deux sortes de problèmes :

- contrôler l'ordre de déclenchement des démons pour respecter la logique de l'application

- détecter une erreur.

Le premier problème apparaît principalement lors de la maintenance d'un programme. L'ajout ou la suppression de l'affectation d'une valeur à un attribut dans un démon peut alors complètement bouleverser l'ordre de déclenchement des démons et le rendre incohérent. Eviter ces problèmes et être capable de descendre les cascades de déclenchements de démons exige l'acquisition de nouveaux réflexes par rapport à ceux du programmeur "traditionnel".

Comme son moteur d'inférences est non-monotone, et comme KOOL autorise l'utilisation simultanée de trois paradigmes de programmation, KOOL ne propose pas, pour l'instant, d'outils pour vérifier et maintenir automatiquement l'intégrité de la base de connaissances lors d'une modification. Cette tâche reste donc entièrement à la charge du programmeur. Cependant KOOL apporte une aide conséquente au développeur en lui permettant de sauvegarder des sessions de test et de les réexécuter automatiquement, ce dernier peut de la sorte vérifier rapidement qu'une mise à jour de la base de connaissances n'altère pas le comportement du système expert sur des problèmes déjà résolus de manière satisfaisante.

4.2.3 La rigueur dans l'écriture des programmes

Du point de vue du génie logiciel, les langages orientés objets sont intéressants pour l'encapsulation des données et des procédures qui les manipulent et pour le masquage des données. En effet l'encapsulation des données et le masquage des données permettent de créer des modules logiciels autonomes qui offrent seulement quelques interfaces d'accès aux autres modules. De cette façon, les langages orientés objets permettent le développement de logiciels extensibles, compatibles et réutilisables.

A l'opposé, KOOL, pour offrir la plus grande souplesse dans la représentation des connaissances, permet au développeur de modifier un objet depuis une règle, un démon, une méthode grâce à son uniformité et à ses fonctions prédéfinies. Si le développeur ne prend pas garde de bien regrouper dans les méthodes d'un objet les fonctions qui travaillent sur les attributs de cet objet et d'imposer leur appel par envoi de message depuis un autre objet, il risque de construire un système impossible à maintenir.

5 EXTENSIONS

La version actuelle de notre système expert démontre largement qu'une approche qui utilise les trois paradigmes de programmation est indispensable pour réussir à développer un système expert pour installer UNIX. Cependant notre système représente seulement nos propres connaissances et notre propre expérience. Il constitue une base qui peut être enrichie progressivement. Aussi, nous allons, dans cette partie, décrire quelques unes des extensions possibles :

- l'enrichissement de la base de connaissances
- l'amélioration des dialogues avec l'utilisateur
- l'ajout de nouveaux services
- l'amélioration de l'implantation
- la définition d'un générateur

5.1 L'ENRICHISSEMENT DE LA BASE DE CONNAISSANCES

Pour le moment, notre système expert ne prend en compte que les matériels et logiciels employés à l'Ecole des Mines. Par conséquent la première extension que nous envisageons est d'enrichir la base de connaissances des caractéristiques des différentes options matérielles des SPS7,SPS9 et maintenant des modèles de la gamme DPX et des différentes options logicielles (langages, bases de données, logiciels applicatifs) proposées par BULL pour pouvoir réaliser l'installation de SPIX sur ces machines. Ce travail demande seulement de rassembler les informations caractéristiques de chaque composant et de déterminer les règles d'installation à observer. Mais nous devons le mener à bien si nous voulons que notre système intéresse le

plus grand nombre d'utilisateurs et connaisse une large diffusion.

De la même façon, nous pouvons envisager d'étendre la base de connaissances de notre outil pour installer des ststions de travail Sun ou la gamme HP9000 de Hewlett-Packard. Mais, pour pouvoir traiter un grand nombre de machines différentes, nous serons sans doute amenés à structurer la base de connaissances en plusieurs sous-ensembles. Chaque sous-ensemble sera alors dédié à un modèle et selon le type de la machine à installer, notre outil chargera seulement le sous-ensemble correspondant.

5.2 L'AMELIORATION DES DIALOGUES

Dans son état de développement présent, notre outil n'est qu'un prototype qui démontre la validité de la démarche. Pour obtenir un produit fiable et réellement utilisable par de nombreux utilisateurs, il faut d'une part renforcer le contrôle des renseignements fournis par l'utilisateur, et d'autre part améliorer les dialogues avec celui-ci. Pour ce faire, il faut rendre la formulation des questions plus accessibles à des non-spécialistes et il faut améliorer les explications données par notre système. Ces améliorations nécessitent de :

- définir le descripteur *Info*, qui explique la signification d'un attribut, pour tous les attributs de tous les objets

- définir un nouveau mécanisme d'explication pour les attributs dont la valeur est calculée par un démon.

5.3 LES NOUVEAUX SERVICES

L'installation d'un système d'exploitation sur un ordinateur ne constitue qu'une faible partie des activités d'un ingénieur système qui conçoit, installe et gère un système informatique pour satisfaire les besoins des utilisateurs. Pour des machines qui utilisent UNIX comme système d'exploitation, notre système expert peut être le point de départ du développement d'un logiciel automatisant l'ensemble des activités d'un ingénieur système. Afin de construire ce logiciel, il faut ajouter à notre système expert des modules qui assurent :

- la génération du noyau UNIX

- l'installation d'un système réparti

- la maintenance

- la conception du système d'informations

Etudions chacun de ces points plus précisément.

5.3.1 La génération du noyau

Nous avons supposé, pour simplifier notre système expert d'installation d'UNIX, que le noyau fourni est convenablement configuré. Cette hypothèse est justifiée car généralement les noyaux UNIX intégrent les pilotes des modules d'échange les plus employés et sont paramétrés de manière satisfaisante pour la plupart des utilisations. Cependant, il n'y a pas de raison pour ne pas appliquer la même démarche pour la génération du noyau que pour l'installation, c'est-à-dire générer un noyau personnalisé qui correspondent exactement à la configuration. Par conséquent, une première extension possible pour notre système expert est de prendre en charge la génération du noyau UNIX.

5.3.2 L'installation d'un système réparti

Si notre système expert est capable d'installer, sur une machine, les logiciels de communication et d'accès à un réseau local d'une machine, nous n'avons pas encore étudié les problèmes posés par l'installation et l'administration d'un système d'informations réparti. Pourtant, comme la maîtrise de ce genre d'architecture qui mélange postes de travail et serveurs de fichiers ou de calculs, est beaucoup plus complexe que celle d'une machine isolée, l'approche système expert se justifie alors pleinement.

En effet, les problèmes de partitionnement des disques deviennent encore plus ardus à cause de la définition des associations serveur-poste de travail pour répartir la charge uniformément sur l'ensemble du système, et de la définition de modes de fonctionnement dégradé. De plus, dans un système réparti, il apparaît de nouveaux problèmes d'installation comme celui de la mise en place des annuaires (*YellowPage*) et de la définition des serveurs et des clients, des maîtres et des esclaves. Une autre extension possible consiste donc à développer un logiciel de configuration d'un système réparti qui s'exécute sur la "machine maître" du réseau [Kincl86] et installe toutes les autres machines.

5.3.3 La maintenance

A la fin d'une installation, notre système expert possède dans sa base de faits une description complète de la machine à installer et de son système. Aussi une autre extension possible consiste à utiliser cette base de faits pour assurer la maintenance de cette configuration. Parmi les opérations de maintenance qu'un système expert pourra prendre en charge, nous pouvons citer :

- la définition des procédures de sauvegarde
- l'installation de nouveaux composants matériels ou logiciels
- l'installation de nouvelles versions des logiciels du système
- le contrôle de l'activité du système pour détecter les intrusions et les virus et proposer des mesures pour les interdire et améliorer la sécurité du système

- l'analyse des performances
- l'ajustement des paramètres du système pour améliorer les performances
- la restauration du système après un incident

5.3.4 La conception d'un système d'informations

Dans son état présent, notre système expert suppose également que le choix du matériel et des logiciels nécessaires à la satisfaction des besoins des utilisateurs a été déterminé auparavant. Cette détermination qui est souvent délicate peut aussi être réalisée par un système expert à partir de la formulation des souhaits des utilisateurs. En effet, la détermination d'une configuration qui réponde aux besoins des utilisateurs est un problème très semblable au conseil en épargne et à l'installation, il s'agit aussi d'un difficile problème d'optimisation sous contraintes.

A partir des caractéristiques des éléments matériels et logiciels et des désirs du client, il faut bâtir une configuration cohérente et réalisable qui permette de satisfaire les besoins des utilisateurs. Par conséquent la création d'un tel outil apportera une aide précieuse aux ingénieurs commerciaux qui ont de plus en plus de mal à maîtriser tous les paramètres qui interviennent dans la conception d'un système informatique. Le développement d'un système expert regroupant l'ensemble des extensions que nous venons de présenter aboutira à la création d'un logiciel qui assumera l'ensemble de la mise en place d'un système d'informations, depuis la conception, en passant par l'installation, jusqu'à la maintenance.

5.4 L'IMPLANTATION

Comme tous les logiciels, notre système expert bénéficiera de l'amélioration des performances des composants matériels et des logiciels de services. Grâce à l'évolution constante des techniques informatiques, nous pouvons espérer que nous disposerons bientôt de mémoire ROM, dont la taille sera suffisante pour contenir des programmes aussi volumineux qu'un système expert, et d'outils de "compilation" qui permettront de générer des versions exécutables compactes des systèmes experts. Nous pourrons alors développer une version de notre système expert intégrant toutes les commandes nécessaires à l'installation d'UNIX qui s'exécutera lors du boot de la machine et contrôlera totalement l'installation en analysant les comptes-rendus des exécutions des différentes commandes. Une étape, vers ce système capable de détecter les erreurs et de les réparer, peut être l'implantation de notre système expert sur un disque "transportable" qui sera connecté à la machine à installer seulement lors de l'installation. L'ingénieur système démarrera la machine sur ce disque d'installation et lancera l'exécution du système expert.

5.5 LE GENERATEUR

Pour modifier la base de connaissances de notre système expert (créer, modifier, supprimer une règle, une classe, une fonction), nous devons pour l'instant utiliser un éditeur de texte pour accéder au fichier où l'information correspondante est conservée. Par conséquent, toute mise à jour du système ne peut être effectuée que par une personne connaissant parfaitement KOOL et le système expert. Pour autoriser l'enrichissement ou la modification de la base de connaissances par un utilisateur qui connaît simplement les caractéristiques des différents composants possibles d'un ordinateur, nous imaginons de développer une interface qui saisira, par exemple, les caractéristiques d'un nouvel élément et en déduira les modifications à apporter à la base de connaissances. La réalisation de ce générateur exigera la représentation des connaissances qui ont permis la construction du système expert. Or les objets sur lesquels travaillent ces connaissances sont les classes et les règles du système expert, il faut donc pouvoir les considérer comme des objets comme les autres. KOOL qui est un langage orienté objet uniforme permettra donc le développement de ce générateur.

CONCLUSION

L'originalité de ce travail réside à la fois dans le domaine d'expertise retenu : l'installation d'UNIX, et dans la solution proposée pour résoudre ce problème : le développement d'un système expert mariant les trois paradigmes de programmation, procédures, objets et règles. En effet, si les progrès de l'informatique ont permis de mettre à la disposition des utilisateurs, y compris les développeurs de logiciels, des outils extrêment puissants qui améliorent spectaculairement leur productivité, jusqu'à présent l'administration des sytèmes informatiques n'a pas bénéficié des mêmes avancées. Excepté quelques logiciels de surveillance de l'activité des machines qui permettent de déterminer les goulots d'étranglement et les modifications à apporter à la configuration matérielle pour les supprimer, l'ingénieur système dispose de peu d'outils pour l'assister dans son travail, notamment lors de l'installation et de la maintenance d'un système.

Or installer UNIX sur un ordinateur est une opération longue et délicate, dont dépend le bon fonctionnement ultérieur du système, qui nécessite des connaissances approfondies des caractéristiques des matériels et des logiciels ainsi qu'une expérience certaine. Cependant ces connaissances sont limitées à un domaine très technique, et très structuré, par conséquent l'approche système expert s'applique parfaitement à la résolution de ce problème comme le démontre la version actuelle de notre installateur. Il est également important de noter que le développement de cet installateur n'a été possible que grâce à l'utilisation d'une représentation hybride des connaissances.

Depuis quelques années, UNIX connaît une diffusion grandissante et il y a de grandes chances qu'UNIX s'impose comme un des standards de fait des systèmes d'exploitation pour les prochaines années. Actuellement, UNIX est principalement installé sur les super-micro-ordinateurs ou les stations de travail pour des applications scientifiques. Par conséquent, la plupart des utilisateurs ne sont pas des informaticiens et encore moins des ingénieurs système, aussi déplorent-ils la complexité de son administration qui est plus proche de celle des gros systèmes informatiques que de celle des micro-ordinateurs. Comme le but de notre système expert et de ses évolutions futures est de rendre l'installation et l'administration d'UNIX accessible à n'importe quel utilisateur, nous espérons qu'il permettra de répondre à cette critique et qu'il connaîtra un grand succès.

Bien qu'il apporte déjà, dans sa version actuelle, une aide appréciable à l'administrateur d'un système UNIX, notre système expert ne dispose pas encore toutes les fonctionnalités que devrait posséder un logiciel d'aide à l'installation UNIX. S'il permet de générer toutes les commandes nécessaires à l'installation UNIX et des principaux logiciels de communication sur un réseau local, par manque de temps nous n'avons pas encore réalisé la détermination du partitionnement des disques en fonction des logiciels à installer et des utilisateurs prévus. Par conséquent, les ressources disques ne sont pas affectées de façon à optimiser les performances.

De plus, pour arriver à un produit de qualité industrielle réellement exploitable par un utilisateur qui n'est pas un administrateur système UNIX, il reste encore à modifier la formulation de certaines questions pour les rendre plus compréhensibles et à enrichir les capacités d'explication de notre système expert. En effet, actuellement il se contente d'afficher les formes externes des règles lors de leur déclenchement.

Un produit industriel devra également intégrer tous les composants de l'ensemble des modèles de la gamme d'un constructeur. Comme la description de tous les éléments représentera une importante quantité d'informations, leur conservation et leur gestion imposera vraisemblablement le recours à une base de données. Si cette base de données est une base de données relationnelle avec comme langage d'interrogation SQL, notre outil devra donc disposer d'une interface avec SQL pour pouvoir accéder aux informations de la base de données. Mais si, comme nous pouvons l'espérer, les bases de données des années 1990 sont des bases de données orientées objets, il faudra interfacer notre outil avec une base de données de ce type comme cela a déjà été réalisé pour l'environnement de développement de sytèmes experts PROTEUS et la base de données orientée objet ORION par une équipe de MCC [Ballou88].

Enfin, notre outil d'aide à l'installation gère seulement l'installation d'une seule machine à la fois alors que la diffusion d'UNIX est surtout liée à celle des systèmes distribués composés de stations de travail et de serveurs. Mais avant de pouvoir installer un système distribué, il faut savoir installer une machine, notre outil constitue donc la première étape vers la conception d'un logiciel d'installation de systèmes distribués. Les utilisateurs de ce type de systèmes commencent d'ailleurs à prendre conscience que, vus leur complexité et leur caractère non algorithmique, seul un système expert permettra de résoudre les problèmes posés par la conception, l'installation et la maintenance des sytèmes répartis. Par exemple, les partenaires du projet Esprit 2 MMI2 ont retenu comme application de démonstration le développement d'un système expert de conception de réseaux locaux Ethernet de stations de travail.

De mon point de vue personnel, l'intérêt de cette thèse tient également au sujet abordé : l'installation d'UNIX. J'ai pu ainsi acquérir les compétences d'un ingénieur système. De plus, la partie réalisation de cette thèse m'a permis de me familiariser avec un environnement de développement de systèmes experts performant et de maîtriser les trois paradigmes de programmation. Mais le principal intérêt de cette thèse réside dans la formation à la recherche que j'ai reçue. Je pense que je retirerai un grand profit de ce travail. Enfin, je voudrais insister sur l'environnement particulièrement favorable à cette recherche que j'ai trouvé à l'Ecole des Mines.

BIBLIOGRAPHIE

1 ARTICLES ET OUVRAGES DE SYNTHESE SUR L'INTELLIGENCE ARTIFICIELLE

[Barr83] A. BARR, E.A. FEIGENBAUM, "The handbook of Artificial Intelligence", 3 volumes, Pitman 1983.

[Buchanan84] B. BUCHANAN, E. SHORTLIFFE, "Rule-based Expert Systems - The MYCIN Experiment of the Standford Heuristic Progamming Project", Addison Wesley Publishing Company,1984.

[Charniak85] E. CHARNIAK, D. MAC DERMOTT, "Introduction to Artificial Intelligence", Addison Wesley Publishing Company,1985.

[Davis77] R. DAVIS, B. BUCHANAN, E. SHORTLIFE, "Production rules as a representation for a knowledge-based consultation program", Artificial Intelligence 8(1977) 15-45.

[Davis82] R. DAVIS, D.B. LENAT, "Knowledge-based sytems in AI", Mc Graw-Hill,1982.

[Feigenbaum63] A. FEIGENBAUM, J. FELDMAN, "Computers and thought", Mc Graw Hill, 1963.

[Forgy82] C.L. FORGY, "Rete: a fast algorithm for the many pattern/many object pattern match problem", Artificial Intelligence 19 (1982) 17-37.

[Hewitt77] C. HEWITT, "Viewing control structures as patterns for passing messages", Artificial Intelligence 8 (1977) 323-364.

[Kayser84] D. KAYSER, "Examen de diverses méthodes utilisées en représentation des connaissances", Actes du 4ème Congrès Reconnaissance des formes et Intelligence Artificielle, Paris janvier 1984 tome 2, 115-144.

[Kowalski79] R. KOWALSKI, "Algorithmic = logic + control", Comm. of the ACM, vol 22, (1979) 424-436.

[Kowalski83] R. KOWALSKI, "Logic for problem solving", North-Holland 1983.

[Laurière82] J.L. LAURIERE, "Représentation et utilisation des connaissances", TSI vol 1 N° 1 et 2 (1982) deux articles.

[Michie68] D. MICHIE, "Machine Intelligence", 3 volumes, Edinburgh University Press, 1968.

[Minsky68] M. MINSKY, "Semantic information processing", The MIT Press, 1968.

[Minsky75] M. MINSKY, "A framework for representing knowledge", in P.H. WINSTON, The psychology of computer vision, Mc Graw-Hill, 1975.

[Moore82] R.C. MOORE, "The role of logic in knowledge representation and commonsense reasoning", Proc. of the National Conf. on Artificial Intelligence, 1982, 428-433.

[Nilsson71] N.J. NILSSON, "Problem-solving methods in artificial intelligence", McGraw-Hill, 1971.

[Nilsson82] N.J. NILSSON, "Principles of Artificial Intelligence", Springer-Verlag, 1982.

[Pitrat85] J. PITRAT, "Textes, ordinateurs et compréhension", Eyrolles 1985.

[Prade84] H. PRADE, "Modèles de raisonnement approché pour les systèmes experts", Actes du colloque INRIA-AFCET : Reconnaissance des formes et intelligence artificielle (1984) tome 2, 355-373.

[Shapiro83] E.Y. SHAPIRO, "Algorithmic program debugging", The MIT Press, 1983.

[Schank79] R.C. SCHANK, "Interestingness: controlling inferences", Artificial Intelligence 12(1979) 273-297.

[Schank82] R.C. SCHANK, "Dynamic memory", Cambdridge University Press, 1982.

[Schank85] R.C. SCHANK, L. HUNTER, "The quest to understand thinking", Byte, vol 10, N°4 (1985) 143-155.

[Shortliffe76] E.H. SHORTLIFFE, "Computer-based medical consultations: MYCIN", Artificial Intelligence series 2, Elsevier 1976.

[Vignard86] P. VIGNARD, "Représentations de connaissances mécanismes d'exploitation et d'apprentissage", INRIA Collection Didactique 1986

[Wertz85] H. WERTZ, "Intelligence artificielle, application à l'analyse de programmes", Masson, 1985.

[Winograd83] T. WINOGRAD, "Language as a cognitive process", vol 1 syntax, Addison-Welsey 1983.

[Winston84] P.H. WINSTON, "Artificial Intelligence", 2nd edition, Addison-Welsey, 1984.

[Winston79] P.H. WINSTON, R.H. BROWN, "Artificial Intelligence: an MIT perspective", vol. I and II, 1979, the MIT Press.

[Wos85] L. WOS et al (environ une dizaine), "An overview of automated reasoning", Journal of Automated Reasoning, 1(1985) 5-48.

2 LOGIQUES

[Apt81] K.R. APT, "Ten years of Hoare's Logic", ACM Trans. on Prog. Languages and Systems, Vol 3, N° 4 (1981) 431-483.

[Apt82] K.R. APT, M.H. VAN EMDEN, "Contributions to the theory of logic programming", JACM, vol 29, N°3 (1982) 841-862.

[Chang73] C.L. CHANG, R.C. LEE, "Symbolic logic and mechanical theorem proving", Academic Press, 1973.

[Chenique74] F. CHENIQUE, "Comprendre la logique moderne", Dunod, 2 tomes, 1974.

[Dubois85] DIDIER DUBOIS, HENRI PRADE, "Théorie des posibilités applications à la représentation des connaissances en informatique", Masson 1985

[Hogger85] C.J. HOGGER, "Introduction to logic programming", Academic Press, 1985

[Hughes68] G. E. HUGHES, M. J. CRESWELL, "An introduction to modal logic", Metuen&Co, Londres 1968

[Pabion76] J.F. PABION, "Logique mathématique", Paris Hermann 1976.

[Prade82] H. PRADE, "Modèles mathématiques de l'imprécis et de l'incertain en vue d'application au raisonnement naturel", Thèse d'état Université Paul Sabatier, Toulouse 1982

[Rescher69] NICHOLAS RESCHER, "Many-valued logic", Mac Graw Hill 1969

[Zadeh75] LOTFI A. ZADEH, "Fuzzy logic and approximated reasoning", Synthese Vol 30 p407-425, 1975

3 SYSTEMES FORMELS ET MODELES DE CALCUL LIES A L'INTELLIGENCE ARTIFICIELLE

[Chandra81] A.K. CHANDRA, D.C. KOZEN, L.J. STOCKMEYER, "Alternation", JACM, vol 28, N°1 (1981) 114-133.

[Georgeff82] M.P. GEORGEFF, "Procedural control in production systems", Artificial Intelligence 18(1982) 175-201.

[Levesque84] H.J. LEVESQUE, "Foundations of a functional approach to knowledge representation", Artificial Intelligence 23 (1984) 155-122.

4 HEURISTIQUES

[Cohen85] P.R. COHEN, "Heuristic reasoning about uncertainty: an artificial intelligence approach", Research Notes in Artificial Intelligence 2, Pitman 1985.

[Lenat82] D. B. LENAT, "The nature of heuristics", Artificial Intelligence 19 (1982) 189-249.

[Newell69] A. NEWELL, "Heuristic programming: ill structured problems", in ARONOFOSKY (Ed.), Progress in operation research, Wiley (1969) 361-414.

[Pearl84] J. PEARL, "Heuristics: intelligence search strategies for computer problem solving", Addison-Welsey 1984.

[Wilkins82] D.E. WILKINS, "Using knowledge to control tree searching", Artificial Intelligence 18 (1982) 1-51.

5 SYSTEMES EXPERTS

[Bonnet81] A. BONNET, "Applications de l'intelligence Artificielle, les systèmes experts", RAIRO Informatique, vol 15, N°4 (1981).

[Bonnet86] A. BONNET, J.P. HATON, J.M. TRUONG-NGOC, "Systèmes experts : vers la maîtrise technique", InterEditions IIA 1986

[Benchimol86] G. BENCHIMOL, P. LEVINE, J.C. POMEROL, "Systèmes experts dans l'entreprise", Hermes, Gestion et Productique 1986

[Brignon87] J. BRIGNON, A. ANDRE, "NOEMIE : un système expert intégré dans le système d'information du groupe BULL", Septièmes journées internationales Les systèmes experts et leurs applications Avignon mai 1987

[Cordier84] M.O. CORDIER, "Les systèmes experts", La Recherche, N°151, vol 5 (1984) 60-70.

[McDermott82] J. McDERMOTT, "R1: a rule based configurer of computer systems", Artificial Intelligence 19 (1982) 39-88.

[Dhar86] V. DHAR, M. JARKE, "Using teleological design knowledge for large systems development and maintenance", Sixièmes journées internationales Les systèmes experts et leurs applications Avignon avril 1986

[Farreny80] H. FARRENY, "Un système pour l'expression et la résolution de problèmes orienté vers le contrôle de robot", Thèse d'Etat, Université Paul SABATIER, 1980.

[Farreny85] H. FARRENY, "Les systèmes experts, principes et exemples", Techniques Avancées de l'Informatique, Cepadues Editions, 1985.

[Gilmore86] J. F. GILMORE, C. HOWARD, "Expert system tool evaluation",Sixièmes journées internationales Les systèmes experts et leurs applications Avignon avril 1986.

[Gondran84] M. GONDRAN, "Introduction aux systèmes experts", Eyrolles 1984.

[Hart88] A. HART, "Acquisition du savoir pour les systèmes experts", Masson, Sciences Cognitives, 1988.

[Hayes-Roth83] F. HAYES-ROTH, D. WATERMAN, D. LENAT, "Building expert systems", Addison-Welsey Pub. Company, 1983.

[Kemppainen86] P. KEMPPAINEN, E. REILIO, "XCM a configuration management expert system for mass production of embedded systems software", Sixièmes journées internationales Les systèmes experts et leurs applications Avignon avril 1986

[Laurent84] J.P. LAURENT, "La structure de contrôle dans les systèmes experts", TSI (1984) , 161-177.

[Laurière84] J. L. LAURIERE, "Un moteur d'inférences pour systèmes experts en logique du premier ordre : SNARK", Bull. de l'INRIA N°97 pp24-28, 1984

[Pinson81] S. PINSON, "Représentation des connaissances dans les systèmes experts", RAIRO informatique, vol 15, N°4 (1981) 343-367.

[Richer86] M. H. RICHER, "An evaluation of expert system development tools", Expert Systems, July 1986, Vol. 3, n. 3, p. 166-182.

[Stefik82] M. STEFIK, J. AIKINS, R. BALZER, J. BENOIT, L. BIRNBAUM, F. HAYES-ROTH, E. SACERDOTI, "The organization of expert systems, a tutorial", Artificial Intelligence 18(1982) 135-173.

[Taillibert86] P. TAILLIBERT, S. VARENNES, "MI4 ou comment développer des systèmes experts avec Prolog", Sixièmes journées internationales Les systèmes experts et leurs applications Avignon avril 1986.

[Vial86] P. VIAL, C. BERTIN, BOURQUIN, DE GASQUET, DESROCHES, LONGEON, "Rapport sur la maquette de système expert de conseil en placement destiné aux particuliers", Ecole des Mines de Saint-Etinne, Département Informatique appliquée, rapport de recherche N°86.3 Décembre 1986

[Vial88] P. VIAL, "Un système expert pour installer UNIX", Huitièmes journées internationales Les systèmes experts et leurs applications Avignon Juin 1988

[Voyer87] R. VOYER, "Moteurs de systèmes experts", Eyrolles, 1987.

6 PROLOG

[Bratko86] I. BRATKO, "Prolog programming for Artificial Intelligence", Addison Wesley Pub. Company, 1986.

[Clocksin81] W.F. CLOCKSIN, C.S. MELLISH, "Programming in Prolog", Springer-Verlag 1981.

[Coelho80] H. COELHO, J.C. COTTA, L.M. PEREIRA, "How to solve it with Prolog", Lab. Nat. de Engenharia Civil, Lisboa 1980.

[Colmerauer83] A. COLMERAUER, KANOUI, VAN CANEGHEM, "Prolog, bases théoriques et développements actuels", TSI (1983) 271-311.

[Condillac86] CONDILLAC (Nom collectif pour le groupe Prolog de l'AFCET), "Prolog: fondements et applications", Dunod, 1986.

[Giannesini85] F. GIANNESINI, H. KANOUI, R. PASERO, "Prolog", Interéditions 1985.

[Sterling86] L. STERLING, E. SHAPIRO, "The art of Prolog", Advanced Programming techniques, M.I.T. Press, 1986.

7 LISP

[McCarthy84] J.McCARTHY, C. TALCOTT, "Lisp, programming and proving", Stanford University, septembre 1984.

[Chailloux86] JEROME CHAILLOUX, M. DEVIN, O. GUILLAUMIN, J. M. HULOT, B. SERLET, J. VUILLEMIN, "LE_LISP version 15.2 le manuel de référence", INRIA Mai 1986 seconde édition

[Girardot85] J.J. GIRARDOT, "Les langages et les sytèmes lisp, une introduction", Editests, 1985.

[Greussay84] P. GREUSSAY, H. WERTZ, "Outils de développement pour la mise au point et la lecture de programmes lisp", Rapport LITP 84-9, mars 1984, Paris.

8 PROGRAMMATION ORIENTEE OBJET

[Ballou88] N. BALLOU, H.T. CHOU, J.F. GARZA, W. KIM, C. PETRIE, D. RUSSINOFF, D. STEINER, D. WOELK, "Coupling an expert system shell with an object-oriented database system", Journal of Object-Oriented Programming vol 1 N°3 12-21 June/July 1988

[Bobrow77-1] D.G. BOBROW, R.M. KAPLAN, M. KAY, D.A. NORMAN, H. THOMPSON, T. WINOGRAD, "Gus, a frame-driven dialog system", Artificial Intelligence 8(1977) 155-173.

[Bobrow77-2] D.G. BOBROW, T. WINOGRAD, "An overview of KRL, a knowledge representation language", Cognitive Science 1, 3-46, (1977).

[Bobrow79] D.G. BOBROW, T. WINOGRAD, "KRL another perspective", Cognitive Science 3, 1-28, (1979).

[Bobrow81] D.G. BOBROW, M. STEFIK, "The loops manual", Xerox Corp., 3333 Coyote Hill Road, Palo Alto, California 94304.

[Cointe81] P. COINTE, "Fermetures dans les lambda interprètes, applications aux langages lisp, plasma et smalltalk", Thèse de 3ème cycle, Université Paris VI, 1981.

[Cox] B. COX, "Object programming"

[Dahl70] O. DAHL, B. MYHRHAUG, K. NYGAARD, "SIMULA-67 common base langage", S-22, Norvegian Computing Center, Oslo october 1970, Simula Information

[Ducournau86] R. DUCOURNAU, J. QUINQUETON, "YAFOOL : encore un langage objet à base de frames", Rapport technique N°72 INRIA, août 1986

[Forgy77] C. FORGY, Mc DERMOTT, "OPS a domain-independant production system language", Proceedings of the 5th IJCAI, 1977.

[Goldberg83] A. GOLDBERG, D. ROBSON, "Smalltalk 80, the language and its implementation", Addison Wesley Publishing Company 1983

[Habib88] M. HABIB, R. DUCOURNAU, "Mechanisms and algorithms for multiple inheritance in object oriented systems", Laboratoire d'Infomatique de Brest, rapport N°1/88 Avril 1988

[KOOL86] BULL Cediag, "Manuel de référence KOOL", 1986

[KOOL87] BULL Cediag, "Manuel de programmation avancée KOOL", 1987

[Lehnert79] W. LEHNERT, Y. WILKS, "A critical perspective on KRL", Cognitive Science 3, 1-28, (1979).

[Meyer88] B. MEYER, "Object Oriented Software Construction", Prentice Hall, 1988.

[Michel88] C. MICHEL, R. ROUSSEAU, M. RUEHER, "Expérimentation d'EFFEIL : un premier bilan", 4ème Colloque de Génie Logiciel Paris 1988

[Riesbeck80] C. RIESBECK, D. McDERMOTT, "Artificial Intelligence programming", L. ERLBAUM Asso. Publishers, Hillsdale, New-Jersey, 1980.

[Sako86] S. SAKO, "Aspects du système APL90 : une extension orientée objet du langage APL", Thèse Ecole des Mines de Saint-Etienne, décembre 1986

[Stefik83] M. STEFIK, D. G. BOBROW, S. MITTAL, L. CONWAY, "Knowledge programming in LOOPS : report on an experimental course", The AI Magazine, Fall 1983.

[Stroustrup86] B. STROUSTRUP, "The C++ programming language" , Addison Wesley Publishing Company, 1986.

[Vial88-2] P. VIAL, "Une approche orientée objet à l'installation d'UNIX", 4ème colloque Génie Logiciel, AFCET Paris Octobre 1988

9 UNIX

[ATT84] UNIX SystemV release 2.0, "Administrator guide", 307-111 Issue2 April 84
UNIX SystemV release 2.0, "Administrator reference manual", 307-111 Issue2 April 84

[Bach86] M. J. BACH, "The design of the UNIX operating system", Prentice Hall Software Series 1986

[BULL86] SPIX System, "Administrator guide", october 1986
SPIX System, "Administrator reference manual", september 1986

[Foxley85] E. FOXLEY, "UNIX for super-users", Addison Wesley Publishing Company, International Computer Science Series 1985

[GIPSI85] GIPSI SM90, "L'installation et l'utilisation de SMX sur SM90", 1985

[GOULD86] GOULD INC. Computer Systems Division, "Product definition for GOULD UTX32/S release1.0", Ft Lauderdale Florida 1986

[Harris86] P. N. HARRIS, "COGITO : an expert system that gives advice for making and installing UNIX 4.2 BSD on VAX 11 series computers", MS dissertation University of Arizona 1986

[Kernighan84] B. KERNIGHAN, R. PIKE, "The UNIX programming environment", Bell Telephone Laboratories, Prentice Hall 1984

[Kincl86] N. KINCL, T. JIN, R. MICHAELS, "Managing a distributed environment", EUUG Manchester Autumn 1986

[Ritchie78] D. M. RITCHIE, K. THOMPSON, "The UNIX time-sharing system", The Bell System Technical Journal vol 57 N°6 part 2 p1905-1930 July/August 1978

Annexe A

LES CLASSES

```
{ClassLayer : InstCL}

{InstClass : Machine
 Super : Object
 <  -- Nom : ()
         Mode : Mono
         Type : String
         Askable : Yes
         Question : {Quel nom voulez-vous donner à votre machine ?}
       WhenFilled : [dem_NomMach,com_NomMach]
    -- Materiel : ()
         Mode : Mono
         Type : Materiel
         Askable : No
    -- Utilisation : ()
         Mode : Mono
         Type : Utilisation
         Askable : No
    -- REP : ()
; répertoire où on crée l'arborescence à installer
         Mode : Mono
         Type : String
         Default : "/manip/Install"
     >}

{InstClass : Materiel
 Super : Object
 <  -- Calculateur : ()
         Mode : Mono
         Type : Calculateur
         Askable : No
    -- Peripherie : ()
         Mode : Mono
         Type : Peripherie
         Askable : No
     >}
```

```
{InstClass : Calculateur
 Super : Object
 <  -- Marque : ()
         Mode : Mono
         Type : [Bull,Telmat]
         Askable : Yes
         Question : {Quelle est la marque de la machine sur laquelle
                       vous installez UNIX ?}
         WhenFilled : [com_MarCal]
    -- Modele : ()
         Mode : Mono
         Type : [SM90,|SPS7/50|,|SPS7/70|,|SPS7/300|,SPS9]
         Askable : Yes
         Question : {Quel est son modèle ?}
       WhenFilled : [dem_ModCal,com_ModCal]
    -- UniCen : ()
       Mode : Mono
       Type : UniteCentrale
       Askable : Yes
    -- NbMED : ()
         Mode : Mono
         Type : [0..2]
         Askable : Yes
         Question : {Combien de MED avez-vous dans votre machine ?}
       WhenFilled : [com_NbMED]
    -- MED : ()
         Mode : Multi
         Type : MED
       Askable : No
         MaxCard : 2
    -- NbMEV24 : ()
         Mode : Mono
         Type : [0..2]
         Askable : Yes
         Question : {Combien de MEV24 avez-vous dans votre machine ?}
       WhenFilled : [com_NbMEV]
    -- MEV24 : ()
         Mode : Multi
         Type : MEV
       Askable : No
         MaxCard : 2
    -- NbMETH : ()
         Mode : Mono
         Type : [0..2]
         Askable : Yes
         Question : {Combien de module Ethernet avez-vous dans votre machine ?}
         WhenFilled : [com_NbMETH,dem_NbMETH]
    -- METH : ()
         Mode : Multi
         Type : METH
       Askable : No
         MaxCard : 2
```

```
-- NbMEX25 : ()
     Mode : Mono
     Type : [0..2]
     Askable : Yes
     Question : {Combien de modules X25 avez-vous dans votre machine ?}
   WhenFilled : [dem_NbMEX25,com_NbMEX25]
-- MEX25 : ()
     Mode : Multi
     Type : MEX25
   Askable : No
     MaxCard : 2
-- NbV24 : ()
     Mode : Mono
     Type : Number
     Askable : Yes
   ToCompute : askV24
-- NbV24Libre : ()
     Mode : Mono
     Type : Number
     Askable : No
-- Bande : ()
     Mode : Mono
     Type : [oui,non]
     Askable : Yes
     Question : {Votre machine possède-t-elle un dérouleur de bande ?}
   WhenFilled : [dem_Deroule,com_Deroule]
-- NbRLOGIN : ()
     Mode : Mono
     Type : [0..32]
   Askable : Yes
     Question : {Combien d'utilisateurs pourront utiliser le rlogin ? }
     WhenFilled : [dem_NbRLOGIN, com_NbRLOGIN]
-- NbPAD : ()
     Mode : Mono
     Type : [0..16]
     Askable : Yes
     Question : {Combien de voies PAD voulez-vous créer ?}
     WhenFilled : [dem_NbPAD]
-- PAD : ()
     Mode : Multi
     Type : PAD
   Askable : No
     MaxCard : 16
 >}
```

```
{InstClass : UniteCentrale
 Super : Object
 <  -- CPU : ()
         Mode : Mono
         Type : [M68000,M68010,M68020,RISC]
         Askable : Yes
         Question : {Quel est le microprocesseur de l'unité de traitement ?}
       WhenFilled : [com_CPUUnCe]
    -- Memoire : ()
         Mode : Mono
         Type : Number
         Askable : Yes
         Question : {Quelle est la taille de la mémoire en Mo ?}
       WhenFilled : [com_MemUnCe]
    -- Console : ()
       Mode : Mono
       Type : Voie
       Askable : No
     >}

{InstClass : MED
 Super : Object
 <  -- Modele : ()
         Mode : Mono
         Type : [Nsc800,HacSasi,HacSmd]
         Askable : Yes
         Question : {Quel est le modèle de * ?}
         WhenFilled : [demMedMod]
    -- Numero : ()
         Mode : Mono
         Type : Number
         Askable : Yes
         Question : {Quel est le numéro de * ?}
       WhenFilled : [demMEDVM]
; Voie tty de maintenance geree par le MED.
    -- VM : ()
         Mode : Mono
         Type : Voie
         Askable : No
       WhenFilled : [dem_VMMED]
    -- Streamer : ()
         Mode : Mono
         Type : [oui,non]
         Askable : Yes
         Question : {Est-ce que * pilote un streamer ?}
         WhenFilled : [dem_Streamer,com_Streamer]
    -- NombreDisque : ()
         Mode : Mono
         Type : [0..6]
         Askable : Yes
         Question : {Combien de disques contrôle * ?}
       WhenFilled : [demDisk]
```

```
    -- Disque : ()
         Mode : Multi
         Type : Disk
       Askable : No
         MaxCard : 6
     >}

{InstClass : Disk
 Super : Object
 <  -- NumMED : ()
         Mode : Mono
         Type : Number
         Askable : No
    -- Numero : ()
         Mode : Mono
         Type : Number
         Askable : No
       WhenFilled : [dem_NumDisk]
    -- Mineur : ()
         Mode : Mono
         Type : Number
         Askable : No
    -- NumDiskPhys : ()
         Mode : Mono
         Type : String
         Askable : No
    -- Modele : ()
         Mode : Mono
         Askable : Yes
         Question : {Quel est le modèle du disque * ?}
         WhenFilled : [dem_GenreDisk]
    -- Genre : ()
         Mode : Mono
         Askable : Yes
         Question : {Quel est le genre du disque * ?}
         WhenFilled : [demModDisk,dem_FormDisk]
    -- SecFormat : ()
         Mode : Mono
         Type : [256,1024]
         Askable : Yes
         Question : {Quelle est la taille (en octets) du secteur
                     utilisée pour le formatage du disque * ?}
    -- ModFormat : ()
         Mode : Mono
         Askable : Yes
    -- Format : ()
         Mode : Mono
         Type : [oui,non,|ne sais pas|,-]
         Askable : Yes
         Question : {Le disque * est-il formaté ?}
         WhenFilled : [dem_FrmtDsk]
```

```
    -- Taille : ()
         Mode : Mono
         Type : Number
         Askable : Yes
         Question : {Quelle est la taille (en blocks de 512 octets)
                       du disque * ?}
    -- PartUnit : ()
         Mode : Mono
         Type : Number
         Askable : Yes
         Question : {Quelle est l'unité de découpage (nombre de blocs)
                     pour que les frontières des partitions
                     correspondent à des fins de pistes ?}
         WhenFilled : [com_Formatage]
    -- Partition : ()
         Mode : Mono
         Type : [oui,non]
         Askable : Yes
         Question : {Voulez-vous créer manuellement des partitions
                       sur le disque * ?}
         WhenFilled : [dem_Partition]
    -- ListPart : ()
         Mode : Multi
         Type : Partition
         Askable : No
         MaxCard : 32
     >}

{InstClass : Partition
 Super : Object
 <  -- Numero : ()
         Mode : Mono
         Type : Number
         Askable : No
    -- Nom : ()
         Mode : Mono
         Type : String
         Askable : Yes
         Question : {Quel est le nom de la partition * ?}
         WhenFilled : [dem_NomPar]
    -- NumDisk : ()
         Mode : Mono
         Type : Number
         Askable : Yes
         Question : {Quel est le numéro du disque où se trouve la
                       partition *?}
    -- NumDiskPhys : ()
         Mode : Mono
         Type : String
         Askable : No
    -- PartUnit : ()
         Mode : Mono
         Type : Number
         Askable : No
```

```
-- Base : ()
     Mode : Mono
     Type : Number
     Askable : Yes
     Question : {Quel est le bloc de base de la partition * ?}
-- Taille : ()
     Mode : Mono
     Type : Number
     Askable : Yes
     Question : {Quelle est la taille de la partition * ?}
     WhenFilled : [dem_PartSize]
-- SysFic : ()
     Mode : Mono
     Type : [oui,non]
     Askable : Yes
     Question : {Voulez-vous créer un système de fichiers sur la
                   partition * ?}
-- Verif : ()
     Mode : Mono
     Type : [oui,non,-]
     Askable : Yes
     Question : {Voulez-vous vérifier la cohérence de la partition
                    * lors du boot ?}
-- Montage : ()
     Mode : Mono
     Type : [oui,non,-]
     Askable : Yes
     Question : {Voulez-vous que la partition * soit montée
                   automatiquement lors du boot ?}
-- Repertoire : ()
     Mode : Mono
     Type : String
     Askable : Yes
     Question : {Répertoire d'accrochage de la partition * ?}
     WhenFilled : [dem_RepPart,com_RepPart]
 >}

{InstClass : MEV
 Super : Object
 <  -- Numero : ()
     Mode : Mono
     Type : Number
     Askable : Yes
     Question : {Quel est le numéro de * ?}
-- NombreVoie : ()
     Mode : Mono
     Type : Number
     Askable : Yes
     ToCompute : askNbVoie
     Question : {Combien de voies V24 gère * ?}
     WhenFilled : [demVoie]
```

```
    -- Voies : ()
         Mode : Multi
         Type : Voie
       Askable : No
         MaxCard : 31
       WhenFilled : [dem_VoieMEV,com_VoieMEV]
     >}

{InstClass : Voie
 Super : Object
 <  -- Indic : ()
; Cet attribut sert à savoir si la voie est gérée par un MEV
; ou par un autre module.
         Mode : Mono
       Type : [oui,non]
       Askable : Yes
         Question : {La voie * est-elle gérée par un MEV ?}
    -- Nom : ()
         Mode : Mono
       Type : String
       Askable : No
         WhenFilled : [demNomTTY]
    -- Genre : ()
         Mode : Mono
         Type : [VM,V24s,V24c]
         Askable : Yes
         Question : {Quel est le type de la * ?}
    -- Utilisation : ()
         Mode : Mono
         Type : [terminal,imprimante,machine,inutilisee]
         Askable : Yes
         Question : {Quelle est l 'utilisation de la * ?}
    -- Vitesse : ()
         Mode : Mono
         Type : [|19200|,|9600|,|4800|,|2400|,|1200|,|300|,|console|,|PAD|]
         Askable : Yes
         Question : {Quel est le modèle dans /etc/gettydefs à utiliser
                       pour la * ?}
    -- Terminal : ()
         Mode : Mono
         Type : String
         Askable : Yes
         Question : {Quel est le modèle de terminal connecté à la * ?}
    -- Inittab : ()
         Mode : Mono
         Type : [off,respawn]
         Askable : Yes
         Question : {Positionnement de l'entrée de la * dans inittab ?}
    -- NomLien : ()
         Mode : Mono
         Type : String
         Askable : Yes
       Question : {Nom du lien à associer à la voie * ? }
     >}
```

```
{InstClass : METH
 Super : Object
 <  -- Modele : ()
       Mode : Mono
       Type : [Hackers,Bull]
       Askable : Yes
       Question : {Quel est le modèle de * ? }
       WhenFilled : [demMethMod]
    -- Numero : ()
         Mode : Mono
         Type : Number
         Askable : Yes
         Question : {Quel est le numéro de * ?}
    -- Majeur : ()
         Mode : Mono
         Type : Number
         Askable : Yes
    -- AdrEthernet : ()
         Mode : Mono
         Type : String
         Askable : Yes
         Question : {Quelle est l'adresse Ethernet de cette machine ?}
       WhenFilled : [dem_AdrEth]
    -- AdrInternet : ()
         Mode : Mono
         Type : String
         Askable : Yes
         Question : {Quelle est l'adresse Internet de cette machine ?}
       WhenFilled : [dem_AdrInt]
    -- Voies : ()
         Mode : Multi
         Type : Voie
       Askable : No
         MaxCard : 6
       WhenFilled : [dem_VoieMETH,com_VoieMETH]
     >}

{InstClass : MEX25
 Super : Object
       Type : [Ost,Bull]
       Askable : Yes
       Question : {Quel est le modèle de * ? }
    -- Numero : ()
         Mode : Mono
         Type : Number
         Askable : Yes
         Question : {Quel est le numéro de * ?}
    -- Majeur : ()
         Mode : Mono
         Type : [|16|,-]
         Askable : Yes
```

```
    -- NbLiaisons : ()
         Mode : Mono
         Type : Number
         Askable : Yes
         Question : {Quel est le nombre de liaisons gérées
                       par le module * ?}
     >}

{InstClass : Peripherie
 Super : Object
 <  -- NbTerm : ()
       Mode : Mono
       Type : [0..32]
       Askable : Yes
       ToCompute : askNT
       Info : {Les micro-ordinateurs personnels ne sont pas à prendre
                en compte, ils apparaissent dans les attributs NbMach
                et MachV24. Les bitmaps seront également traités à part}
       WhenFilled : [demTerm]
    -- Term : ()
       Mode : Multi
       Type : Terminal
       Askable : No
       MaxCard : 32
    -- NbImpSer : ()
       Mode : Mono
       Type : [0..4]
       Askable : Yes
       ToCompute : askNIS
       WhenFilled : [demImpSer]
    -- ImpSer : ()
       Mode : Multi
       Type : ImpriSer
       Askable : No
       MaxCard : 4
    -- NbImpPar : ()
       Mode : Mono
       Type : [0..4]
       Askable : Yes
       Question : {Combien d'imprimantes parallèles sont connectées
                     sur votre machine ?}
       WhenFilled : [demImpPar]
    -- ImpPar : ()
       Mode : Multi
       Type : ImpriPar
       Askable : No
       MaxCard : 4
    -- DestDefSpool : ()
       Mode : Mono
       Type : [|pas de defaut|]
       Askable : Yes
```

```
    -- NbMachV24 : ()
       Mode : Mono
       Type : [0..16]
       Askable : Yes
       WhenFilled : [demMachV24]
       ToCompute : askNMV
    -- MachV24 : ()
       Mode : Multi
       Type : MachV24
       Askable : No
       MaxCard : 16

    >}

{InstClass : Terminal
 Super : Object
 <  -- Marque : ()
       Mode : Mono
       Type : String
       Askable : Yes
       Question : {Quelle est la marque du terminal * ?}
    -- Modele : ()
       Mode : Mono
       Type : String
       Askable : Yes
       Question : {Quel est le modèle du terminal * ?}
    -- Emul : ()
       Mode : Mono
       Type : String
       Askable : Yes
       Question : {Quelle est l'émulation utilisée par le terminal * ?}
    -- Vitesse : ()
       Mode : Mono
         Type : [|19200|,|9600|,|4800|,|2400|,|1200|,|300|,|console|,|PAD|]
       Askable : Yes
       Question : {A Quelle vitesse fonctionne le terminal * ?}
    -- Voie : ()
       Mode : Mono
       Type : Voie
       Askable : No
    >}

{InstClass : Imprimante
 Super : Object
 <  -- Marque : ()
       Mode : Mono
       Type : String
       Askable : Yes
       Question : {Quelle est la marque de l'imprimante * ?}
    -- Modele : ()
       Mode : Mono
       Type : String
       Askable : Yes
       Question : {Quel est le modèle de l'imprimante * ?}
```

```
    -- Nom : ()
       Mode : Mono
       Type : String
; longueur à limiter à 14 caractères car nom de fichier UNIX
       Askable : Yes
       Question : {Quel nom voulez-vous donner à l'imprimante * ?}
       WhenFilled :[demNomImp]
    -- ProgInter : ()
       Mode : Mono
       Type : String
       Askable : Yes
       ToCompute : askProg
       Info : {Si la réponse est un cheminom relatif, on suppose
                 que le programme d'interface se trouve dans le
                 répertoire /usr/spool/lp/model. Si le cheminom est
                 un cheminom absolu, on stocke basename dans
                 ProgInter et dirname dans RepProg.}
    -- RepProg : ()
       Mode : Mono
       Type : String
       Askable : No
    -- Classe : ()
       Mode : Mono
       Type : Number
       Askable : Yes
       Question : {A Quelle classe appartient l'imprimante * ?}
       WhenFilled :[demClasse]
    -- Interface : ()
       Mode : Mono
       Type : [serie,parallele]
       Askable : No
    >}

{InstClass : ImpriPar
 Super : Imprimante
 <  -- Interface : parallele
    >}

{InstClass : ImpriSer
 Super : Imprimante
 <  -- Interface : serie
    -- Vitesse : ()
       Mode : Mono
         Type : [|19200|,|9600|,|4800|,|2400|,|1200|,|300|,|console|,|PAD|]
       Askable : Yes
       Question : {A quelle vitesse travaille l'imprimante * ? }
    -- Parle : ()
       Mode : Mono
       Type : [oui,non]
       Askable : Yes
       Question : {Est-ce que l'imprimante * envoie des messages au
                      calculateur ?}
```

```
    -- Voie : ()
       Mode : Mono
       Type : Voie
       Askable : No
    >}

{InstClass : MachV24
 Super : Object
 <  -- Modele : ()
       Mode : Mono
       Type : String
       Askable : Yes
       Question : {Quel est le modèle de la machine * que vous voulez
                      connecter par une V24 sur votre machine ?}
    -- SysExp : ()
       Mode : Mono
       Type : String
       Askable : Yes
       Question : {Quel est le système d'exploitation de la machine * ?}
    -- LogCom : ()
       Mode : Multi
       Type : String
       Askable : Yes
       Question : {Quels logiciels de communication voulez-vous
                      utiliser entre votre machine et la machine * ?}
    -- Status : ()
       Mode : Mono
       Type : [maitre,esclave]
       Askable : Yes
       Question : {Quel est le comportement de la machine * vis à vis
                      de votre machine ?}
       WhenFilled : [demStatus]
       Info : {Par maître, nous entendons une machine qui prend
         l'initiative d'établir la connexion avec votre machine.
         Par esclave une machine qui attend d'être appelée par
         votre machine}
    -- TermEmu : ()
       Mode : Mono
       Type : String
       Askable : Yes
       Question : {Quel modèle de terminal la machine * émule-t-elle ?}
    -- Vitesse : ()
       Mode : Mono
         Type : [|19200|,|9600|,|4800|,|2400|,|1200|,|300|,|console|,|PAD|]
       Askable : Yes
       Question : {Quel est la vitesse de communication entre la
                      machine * et votre machine ?}
    -- Nom : ()
       Mode : Mono
       Type : String
       Askable : Yes
       Question : {Quel est le nom la machine * ?}
```

```
    -- Voie : ()
       Mode : Mono
       Type : Voie
       Askable : No
    >}

{InstClass : PAD
  Super : Object
  < -- Numero : ()
         Mode : Mono
         Type : [1..16]
         Askable : No
    -- EntSort : ()
         Mode : Mono
         Type : [entree, sortie, inutilisee]
         Askable : Yes
         Question : {Quelle est la fonction de * ?}
    -- Donnees : ()
         Mode : Mono
         Type : String
         Askable : Yes
         Question : {Quelles données d'appel voulez- vous pour * ?
                      Vous pouvez tapper """""" pour "vide".}
    -- Coord : ()
         Mode : Mono
         Type : String
         Askable : Yes
         Question : {Coordonées de la machine pour * }
    -- Majeur : ()
         Mode : Mono
         Type : Number
         ToCompute : getMajPAD
         Askable : No
    -- Mineur : ()
         Mode : Mono
         Type : Number
         ToCompute : getMinPAD
         Askable : No
     >}

{InstClass : Utilisation
 Super : Object
 <  -- NbMaxUtilisateurs : ()
         Mode : Mono
         Type : Number
         Askable : Yes
         Question : {Nombre maximum d'utilisateurs sur la machine ?}
   -- Utilisateurs : ()
         Mode : Multi
         Type : Utilisateurs
         Askable : No
       MaxCard : () ;le donner avec une règle en dépendance de
                    ;la machine ?
 >}
```

```
{InstClass : Utilisateurs
 Super : Object
 <  -- NomLogin : ()
         Mode : Mono
         Type : String
         Askable : Yes
         Question : {Nom de l'utilisateur pour se logger ?}
    -- Numero : ()
         Mode : Mono
         Type : Number
         Askable : Yes
         Question : {Quel est le numéro de l'utilisateur ?}
    -- NumGroupe : ()
         Mode : Mono
         Type : Number
         Askable : Yes
         Question : {Quel est son numéro de groupe ?}
    -- Shell : ()
         Mode : Mono
         Type : String
         Default : "/bin/sh"
         Askable : Yes
         Question : {Quel est son programme d'accueil ?}
    -- PATH : ()
         Mode : Mono
         Type : String
         Askable : Yes
         Question : {Quel est le répertoire d'acceuil ?}
    -- Donnees : ()
         Mode : Mono
         Type : String
         Askable : Yes
         Question : {Quelles sont données à retenir dans passwd ?}
 >}
```

Annexe B

LES REGLES EN CHAINAGE-AVANT

```
{RuleLayer : InstForwardRL}

{InstRule + : FR1
If
  Cal^NbMED = 1
Then
  (Do
      (Make *MED MED_1)
      (getObjectValue MED_1 1 2)
      (addValue Cal MED MED_1)
      (addOwner MED_1 InstIL)
  )
Extern
  {Il n'y a qu'un MED ALORS on demande ses attributs.}
}

{InstRule + : FR2
If
  Cal^NbMED = 2
Then
  (Do
      (Make *MED  MED_1)
      (getObjectValue MED_1 1 2)
      (addValue Cal MED MED_1)
      (addOwner MED_1 InstIL)
      (Make *MED  MED_2)
      (getObjectValue MED_2 1 2)
      (addValue Cal MED MED_2)
      (addOwner MED_2 InstIL)
  )
Extern
  {Il y a deux MED ALORS on demande leurs attributs.}
}
```

```
{InstRule + : FR3
If
  Cal^NbMEV24 = 1
Then
  (Do
      (Make *MEV MEV_1)
      (getObjectValue MEV_1 1 2)
      (addValue Cal MEV24 MEV_1)
      (addOwner MEV_1 InstIL)
  )
Extern
  {Il n'y a qu'un MEV ALORS on demande ses attributs.}
}

{InstRule + : FR4
If
  Cal^NbMEV24 = 2
Then
  (Do
      (Make *MEV  MEV_1)
      (getObjectValue MEV_1 1 2)
      (addValue Cal MEV24 MEV_1)
      (addOwner MEV_1 InstIL)
      (Make *MEV  MEV_2)
      (getObjectValue MEV_2 1 2)
      (addValue Cal MEV24 MEV_2)
      (addOwner MEV_2 InstIL)
  )
Extern
  {Il y a deux MEV ALORS on demande leurs attributs.}
}

{InstRule + : FR5
If
  Cal^NbMEX25 = 1
Then
  (Do
      (Make *MEX25 MEX25_1)
      (getObjectValue MEX25_1 1 2)
      (addValue Cal MEX25 MEX25_1)
      (addOwner MEX25_1 InstIL)
  )
Extern
  {Il n'y a qu'un MEX25 ALORS on demande ses attributs.}
}
```

```
{InstRule + : FR6
If
  Cal^NbMEX25 = 2
Then
  (Do
      (Make *MEX25  MEX25_1)
      (getObjectValue MEX25_1 1 2)
      (addValue Cal MEX25 MEX25_1)
      (addOwner MEX25_1 InstIL)
      (Make *MEX25  MEX25_2)
      (getObjectValue MEX25_2 1 2)
      (addValue Cal MEX25 MEX25_2)
      (addOwner MEX25_2 InstIL)
  )
Extern
  {Il y a deux MEX25 ALORS on demande leurs attributs.}
}

{InstRule + : FR7
If
  Cal^NbMETH = 1
Then
  (Do
      (Make *METH METH_1)
      (getObjectValue METH_1 1 2)
      (addValue Cal METH METH_1)
      (addOwner METH_1 InstIL)
  )
Extern
  {Il n'y a qu'un METH ALORS on demande ses attributs.}
}

{InstRule + : FR8
If
  Cal^NbMETH = 2
Then
  (Do
      (Make *METH  METH_1)
      (getObjectValue METH_1 1 2)
      (addValue Cal METH METH_1)
      (addOwner METH_1 InstIL)
      (Make *METH  METH_2)
      (getObjectValue METH_2 1 2)
      (addValue Cal METH METH_2)
      (addOwner METH_2 InstIL)
  )
Extern
  {Il y a deux METH ALORS on demande leurs attributs.}
}
```

```
{InstRule + : FR9
If
  *Voie^Utilisation = imprimante
Then
  *Voie^Terminal = "un"
  *Voie^Inittab = off
Extern
  {Une imprimante est connectée sur la voie
    ALORS l'entrée correspondante dans inittab est mise
    à "off" et le type de terminal est "un" }
}

{InstRule + : FR10
If
  *Voie^Utilisation = terminal
Then
  *Voie^Inittab = respawn
Extern
  {Un terminal est connecté sur la voie
    ALORS l'éntrée correspondante dans inittab est mise à respawn.}
}


{InstRule + : FR11
If
  Cal^Modele Diff SPS9
Then
  (putClassValue MEV NombreVoie Type '(oneOf 6 7 8 31))
Extern
  {La machine n'est pas un SPS9 ALORS
    le nombre de voies qu'elle peut gérer ne peut
    être que 6, 7, 8 ou 31.}
}
```

Annexe C

LES REGLES EN CHAINAGE-ARRIERE

```
{RuleLayer : InstBackwardRL}

{InstRule - : BR1
If
  Cal^Marque = Telmat
Then
  Cal^Modele = SM90
Extern
  {La machine est une Telmat ALORS c'est une SM90 (seul cas traité).}
}

{InstRule - : BR2
If
  Cal^Modele = SM90
Then
  UC^CPU = M68000
  (putClassValue Disk Modele Type '(oneOf xt1065 d570 dma d140 d160
atasi SMD160))
Extern
  {La machine est une SM90
    ALORS son microprocesseur est un 68000
    et ses disques peuvent être xt1065 d570 dma d140 d160 atasi SMD160}
}

{InstRule - : BR3
If
  Cal^Modele = |SPS7/50|
Then
  UC^CPU = M68010
  (putClassValue Disk Modele Type '(oneOf d570 dma d140 d160))
Extern
  {La machine est un SPS7/50
    ALORS son microprocesseur est un 68010
    et ses disques peuvent être d570 dma d140 d160}
}
```

```
{InstRule - : BR4
If
  Cal^Modele = |SPS7/70|
Then
  UC^CPU = M68020
  (putClassValue Disk Modele Type '(oneOf d570 MXT190))
Extern
  {La machine est un SPS7/70
    ALORS son microprocesseur est un 68020
    et ses disques peuvent être d570 Mxt190}
}

{InstRule - : BR5
If
  Cal^Modele = |SPS7/300|
Then
  UC^CPU = M68020
  (putClassValue Disk Modele Type '(oneOf d570 Mxt190 floppy))
Extern
  {La machine est un SPS7/300
    ALORS son microprocesseur est un 68020
    et ses disques peuvent être d570 Mxt190 floppy.}
}

{InstRule - : BR6
If
  Cal^Modele = SPS9
Then
  UC^CPU = RISC
  (putClassValue Disk Modele Type '(oneOf Eagle F2322 F2333 F2312 F2344
CDC9766 P154))
Extern
  {La machine est un SPS9
    ALORS c'est une machine à architecture RISC
    et ses disques possibles : Eagle F2322 F2333 F2312 F2344 CDC9766 P154}
}

{InstRule - : BR7
If
  *MED^Numero = *Disk^NumMED
  *MED^Modele = Nsc800
Then
  *Disk^SecFormat = 256
Extern
  {Le MED pilotant le disque est un Nsc800
    ALORS la taille de secteur est de 256 octets.}
}
```

```
{InstRule - : BR8
If
  *MED^Numero = *Disk^NumMED
  *MED^Modele = HacSmd
Then
  *Disk^Modele = SMD160
  *Disk^Genre = dur
  *Disk^SecFormat = 1024
  *Disk^ModFormat = SMD160
  *Disk^Format = |ne sais pas|
  *Disk^Taille = 294336
  *Disk^PartUnit = 16
Extern
  {Le MED pilotant le disque est un HacSmd
    ALORS ce disque est un SMD160 c'est un disque dur,
    sa taille est 294336, la taille de secteur est de 1024 octets,
    le modèle de formatage est le SMD160
    et les partitions doivent avoir une taille multiple de 16.}
-- Interest : 60
}

{InstRule - : BR9
If
  *MED^Numero = *Disk^NumMED
  *MED^Modele = HacSasi
Then
  *Disk^SecFormat = 1024
  *Disk^Modele = d570
  *Disk^ModFormat = d570k
  *Disk^Taille = 122976
  *Disk^PartUnit = 18
Extern
  {Le MED pilotant le disque est un HacSasi
    ALORS ce disque est un d570, le modèle de formatage est le d570k,
    la taille de secteur est de 1024 octets,
    la taille du disque est de 122976 blocs
    et les partitions doivent avoir une taille multiple de 18.}
-- Interest : 60
}

{InstRule - : BR10
If
  *MED^Streamer = non
  *MED^Numero = *Disk^NumMED
  *MED^Modele = Nsc800
  *Disk^Modele = d570
Then
  *Disk^ModFormat = d570
  *Disk^Taille = 109312
  *Disk^PartUnit = 16
Extern
  {La machine ne possède pas de streamer
    et le MED pilotant le disque d570 est un Nsc800
    ALORS le modèle de formatage est le d570, sa taille est 109312
    et les partitions doivent avoir une taille multiple de 16}
```

```
}

{InstRule - : BR11
If
  Cal^Modele Diff |SPS7/300|
  *Disk^Modele = d570
  *MED^Numero = *Disk^NumMED
  *MED^Modele = Nsc800
  *MED^Streamer = oui
Then
  *Disk^ModFormat = d570s
  *Disk^Taille = 112728
  *Disk^PartUnit = 66
Extern
  {La machine n'est pas un SPS7/300
    elle possède un streamer
    le disque est un d570 piloté par un MED Nsc800
    ALORS le modèle de formatage est le modèle d570s,
    sa taille est 112728,
    et les partitions doivent avoir une taille multiple de 66.}
}

{InstRule - : BR12
If
  Cal^Modele = |SPS7/300|
  *Disk^Modele = d570
  *MED^Numero = *Disk^NumMED
  *MED^Modele = Nsc800
Then
  *MED^Streamer = oui
  *Disk^ModFormat = d570s
  *Disk^Taille = 118568
  *Disk^PartUnit = 126
Extern
  {La machine est un SPS7/300
    et le disque est un d570 piloté par un MED Nsc800
    ALORS elle possède un streamer, sa taille est 118568,
    le modèle de formatage est d570s
    et les partitions doivent avoir une taille multiple de 126.}
}
```

```
{InstRule - : BR13
If
  *Disk^Modele = dma
  *Disk^Genre = mobile
Then
  *Disk^ModFormat = dmam
  *Disk^Taille = 9792
  *Disk^PartUnit = 16
  *Disk^Format = |-|
  *Disk^Partition = non
Extern
  {Le disque est la partie mobile d'un dma
    ALORS le modèle de formatage est le dmam, sa taille est 9792,
    les partitions doivent avoir une taille multiple de 16,
    on ne s'intéresse pas à son formatage
    et on ne crée pas de partitions.}
-- Interest : 60
}

{InstRule - : BR14
If
  *Disk^Modele = dma
  *Disk^Genre = fixe
Then
  *Disk^ModFormat = dmaf
  *Disk^Format = |ne sais pas|
  *Disk^Taille = 9792
  *Disk^PartUnit = 16
Extern
  {Le disque est la partie fixe d'un dma
    ALORS le modèle de formatage est le dmaf, sa taille est 9792,
    les partitions doivent avoir une taille multiple de 16}
-- Interest : 60
}

{InstRule - : BR15
If
  *Disk^Modele = d140
  *Disk^Genre = mobile
Then
  *Disk^ModFormat = d140m
  *Disk^Taille = 18768
  *Disk^PartUnit = 24
  *Disk^Format = |-|
  *Disk^Partition = non
Extern
  {Le disque est la partie mobile d'un d140
    ALORS le modèle de formatage est d140m, sa taille est 18768,
    les partitions doivent avoir une taille multiple de 24,
    on ne s'intéresse pas à son formatage,
    et on ne cree pas de partitions.}
-- Interest : 60
}
```

```
{InstRule - : BR16
If
  *Disk^Modele = d140
  *Disk^Genre = fixe
Then
  *Disk^ModFormat = d140f
  *Disk^Taille = 18768
  *Disk^PartUnit = 24
Extern
  {Le disque est la partie fixe d'un d140
    ALORS le modèle de formatage est d140f, sa taille est 18768,
    les partitions doivent avoir une taille multiple de 24}
-- Interest : 60
}

{InstRule - : BR17
If
  *Disk^Modele = d160
Then
  *Disk^Genre = dur
  *Disk^Taille = 225600
  *Disk^ModFormat = d168
  *Disk^PartUnit = 24
Extern
  {Le disque est un d160
    ALORS c'est un disque dur, sa taille est 225600,
    le modèle de formatage est le d168
    et les partitions doivent avoir une taille multiple de 24}
-- Interest : 60
}

{InstRule - : BR18
If
  *Disk^Modele = d570
Then
  *Disk^Genre = dur
  *Disk^Format = |ne sais pas|
Extern
  {Le disque est un d570 ALORS il ne peut être que dur.}
}
```

```
{InstRule - : BR19
If
  *Disk^Modele = xt1065
Then
  *Disk^Genre = dur
  *Disk^Taille = 102480
  *Disk^ModFormat = xt1065
  *Disk^Format = |ne sais pas|
  *Disk^PartUnit = 16
Extern
  {Le disque est un maxtor 1065
    ALORS c'est un disque dur, sa taille est 102480,
    le modèle de formatage est le xt1065
    et les partitions doivent avoir une taille multiple de 16}
-- Interest : 60
}

{InstRule - : BR20
If
  *Disk^Modele = MXT190
Then
  *Disk^Genre = dur
  *Disk^Taille = 321032
  *Disk^ModFormat = MXT190
  *Disk^Format = |ne sais pas|
  *Disk^PartUnit = 270
Extern
  {Le disque est un maxtor MXT190
    ALORS c'est un disque dur, sa taille est 321032,
    le modèle de formatage est le MXT190
    et les partitions doivent avoir une taille multiple de 270}
-- Interest : 60
}

{InstRule - : BR21
If
  *Disk^Modele = atasi
Then
  *Disk^Genre = dur
  *Disk^Taille = 71120
  *Disk^ModFormat = atasi
  *Disk^Format = |ne sais pas|
  *Disk^PartUnit = 16
Extern
  {Le disque est un atasi
    ALORS c'est un disque dur, sa taille est 71120,
    le modèle de formatage est le atasi
    et les partitions doivent avoir une taille multiple de 16}
-- Interest : 60
}
```

```
{InstRule - : BR22
If
  *Disk^Modele = floppy
Then
  *Disk^Taille = 0
  *Disk^PartUnit = 0
  *Disk^Format = |-|
  *Disk^Partition = non
Extern
  {Le disque est un floppy
    ALORS il existe 3 modèles avec les tailles suivantes 2400, 1200, 720.
    On ne s'interesse pas à son formatage
    et on ne crée pas de partitions.}
}

{InstRule - : BR23
If
  Cal^Modele Diff SPS9
  Cal^NbMED = 1
Then
  *MED^Numero = 3
Extern
  {La machine n'est pas un SPS9 et il n'y a qu'un MED
    ALORS son numero est 3. }
}

{InstRule - : BR23-a
If
  Cal^Modele = SPS9
  Cal^NbMED = 1
Then
  *MED^Numero = 0
Extern
  {La machine est un SPS9 et il n'y a qu'un MED
    ALORS son numero est 0. }
}

{InstRule - : BR24
If
  *Partition^SysFic = non
Then
  *Partition^Verif = |-|
  *Partition^Montage = |-|
  *Partition^Repertoire = "-"
Extern
  {On ne crée pas de système de fichiers sur cette partition
    ALORS c'est inutile de la verifier, de la monter au boot
    et d'y associer un répertoire.}
}
```

```
{InstRule - : BR25
If
  *Partition^Nom = "root"
Then
  *Partition^SysFic = oui
  *Partition^Verif = |-|
  *Partition^Montage = |-|
  *Partition^Repertoire = "/"
Extern
  {La partition est la partition root
    ALORS on y crée un système de fichier,
    on ne la place ni dans /etc/checklist ni dans /etc/mountlist
    et le répertoire associé est /}
}

{InstRule - : BR26
If
  *Partition^Nom = "swap"
Then
  *Partition^SysFic = non
Extern
  {La partition est la zone de swap
    ALORS on n'y crée pas un système de fichiers.}
}


{InstRule - : BR27
If
  (UnKnown *Voie Utilisation)
Then
  *Voie^Utilisation = inutilisee
  *Voie^Vitesse = |9600|
  *Voie^Terminal = "un"
  *Voie^Inittab = off
Extern
  {La voie est inutilisée
    ALORS la vitesse est par défaut de 9600,
    l'entrée correspondante dans /etc/inittab est mise à "off"
    et le type de terminal est "un".}
}


{InstRule - : BR28
If
  *METH^Modele = Bull
Then
  *METH^Majeur = 20
  *METH^AdrEthernet = " "
Extern
  {Le module Ethernet est un module BULL
    ALORS son majeur est 20 et son adresse Ethernet est inutile.}
}
```

```
{InstRule - : BR29
If
  *METH^Modele = Hackers
Then
  *METH^Majeur = 26
Extern
  {Le module Ethernet est un module HACKERS
    ALORS son majeur est 26. }
}

{InstRule - : BR30
If
  Cal^NbMEX25 = 1
Then
  *MEX25^Numero = 13
Extern
  {Il n'y a qu'un MEX25 ALORS son numéro est 13.}
}


{InstRule - : BR31
If
  *MEX25^Modele = Bull
Then
  *MEX25^Majeur = |-|
  *MEX25^NbLiaisons = 2
Extern
  {Le module X25 est un module BULL
    ALORS son majeur est -  et il gère 2 liaisons X25.}
}

{InstRule - : BR32
If
  *MEX25^Modele = Ost
Then
  *MEX25^Majeur = |16|
  *MEX25^NbLiaisons = 2
Extern
  {Le module X25 est un module OST
    ALORS son majeur est 16 et il gère 2 liaisons X25.}
}

{InstRule - : BR33
If
  Cal^NbMEV24 = 1
Then
  *MEV^Numero = 1
Extern
  {Il n'y a qu'un MEV ALORS son numéro est 1. }
}
```

```
{InstRule - : BR34
If
  Cal^NbMETH = 1
Then
  *METH^Numero = 15
Extern
  {Il n'y a qu'un METH ALORS son numéro est 15.}
}


{InstRule - : BR35
If
  Cal^Modele Diff SPS9
Then
  Perif^NbImpPar = 0
Extern
  {La machine n'est pas un SPS9
    ALORS on considère qu'il n'y a pas d'imprimante parallèle.}
}

{InstRule - : BR36
If
  *Terminal^Marque = Ampex
  *Terminal^Modele = 210
Then
  *Terminal^Emul = tvi925
Extern
  {Le terminal est un Ampex 210
    ALORS nous conseillons de l'utiliser en émulation tvi925.}
}

{InstRule - : BR37
If
  *Terminal^Marque = Bull
  *Terminal^Modele = dku5029
Then
  *Terminal^Emul = tvi950
Extern
  {Le terminal est un Bull avec clavier qwerty
    ALORS nous conseillons de l'utiliser en émulation tvi950.}
}
```

```
{InstRule - : BR38
If
  *MachV24^Modele = "PC"
Then
  *MachV24^SysExp = "MS-DOS"
  *MachV24:LogCom <-+ ["kermit"]
  *MachV24^Status = maitre
  *MachV24^TermEmu = "h19"
Extern
  {La machine est un PC
    ALORS son système d'exploitation est MS-DOS,
    la communication se fera par kermit,
    les connexions seront réalisées depuis le PC
    et on utilise le h19 comme émulation de terminal.}
}

{InstRule - : BR39
If
  *MachV24^Modele = "atari"
Then
  *MachV24^SysExp = "atari"
  *MachV24^Status = maitre
  *MachV24^TermEmu = "vt100"
Extern
  {La machine est un atari
    ALORS son système d'exploitation est atari,
    les connexions seront réalisées depuis l'atari,
    on utilise le vt100 comme émulation de terminal.}
}

{InstRule - : BR40
If
  *MachV24^Modele = "mac"
Then
  *MachV24^SysExp = "mac"
  *MachV24^Status = maitre
  *MachV24^TermEmu = "vt100"
Extern
  {La machine est un mac
    ALORS son système d'exploitation est mac,
    les connexions seront réalisées depuis le mac
    et on utilise le vt100 comme émulation de terminal.}
}

{InstRule - : BR41
If
  Perif^NbImpSer = 0
  Perif^NbImpPar = 0
Then
  Perif^DestDefSpool = |pas de defaut|
Extern
  {Il n'y a ni imprimantes série ni imprimantes parallèles
    ALORS le spool n'a pas de destination par défaut.}
}
```

```
{InstRule - : BR42
If
  (UnKnown *Voie Terminal)
  (UnKnown *Voie Utilisation)
  *Voie^Genre = V24c
  *MachV24^Status = maitre
Then
  *MachV24^Voie = *Voie
  /*Voie^Terminal = *MachV24^TermEmu
  /*Voie^Utilisation = machine
  /*Voie^Inittab = respawn
  /*Voie^NomLien = (catenate "tty" *MachV24^Nom)
  /*Voie^Vitesse = *MachV24^Vitesse
Extern
  {Si on trouve une V24 complète inutilisée
    ALORS on l'affecte à la machine
    et on positionne les attributs de cette voie.}
-- Interest : 100
}

{InstRule - : BR43
If
  (UnKnown *Voie Terminal)
  (UnKnown *Voie Utilisation)
  *Voie^Genre = V24c
  *MachV24^Status = esclave
Then
  *MachV24^Voie = *Voie
  /*Voie^Terminal = "un"
  /*Voie^Utilisation = machine
  /*Voie^Inittab = off
  /*Voie^NomLien = (catenate "tty" *MachV24^Nom)
  /*Voie^Vitesse = *MachV24^Vitesse
Extern
  {Si on trouve une V24 complète inutilisée
    ALORS on l'affecte à la machine
    et on positionne les attributs de cette voie.}
-- Interest : 100
}
```

```
{InstRule - : BR44
If
  (UnKnown *Voie Terminal)
  (UnKnown *Voie Utilisation)
  *Voie^Genre Diff V24c
  *MachV24^Status = maitre
Then
  *MachV24^Voie = *Voie
  /*Voie^Terminal = *MachV24^TermEmu
  /*Voie^Utilisation = machine
  /*Voie^Inittab = respawn
  /*Voie^NomLien = (catenate "tty" *MachV24^Nom)
  /*Voie^Vitesse = *MachV24^Vitesse
Extern
  {ATTENTION toutes les V24 complètes sont déjà employées
    j'utilise une V24 simplifiée ou une voie de maintenance
    pour connecter cette machine cela peut être source de PROBLEMES.}
}

{InstRule - : BR46
If
  (UnKnown *Voie Terminal)
  (UnKnown *Voie Utilisation)
  *Voie^Genre Diff V24c
  *MachV24^Status = esclave
Then
  *MachV24^Voie = *Voie
  /*Voie^Terminal = "un"
  /*Voie^Utilisation = machine
  /*Voie^Inittab = off
  /*Voie^NomLien = (catenate "tty" *MachV24^Nom)
  /*Voie^Vitesse = *MachV24^Vitesse
Extern
  {ATTENTION toutes les V24 complètes sont déjà employées
    j'utilise une V24 simplifiée ou une voie de maintenance
    pour connecter cette machine cela peut être source de PROBLEMES.}
}

{InstRule - : BR47
If
  (UnKnown *Voie Terminal)
  (UnKnown *Voie Utilisation)
  *Voie^Genre = V24c
  *ImpriSer^Parle = oui
Then
  *ImpriSer^Voie = *Voie
  /*Voie^Utilisation = imprimante
  /*Voie^NomLien = *ImpriSer^Nom
  /*Voie^Vitesse = *ImpriSer^Vitesse
Extern
  {Si on trouve une V24 complète inutilisée on l'affecte
    à l'imprimante qui dialogue avec votre machine.}
-- Interest : 100
}
```

```
{InstRule - : BR48
If
  (UnKnown *Voie Terminal)
  (UnKnown *Voie Utilisation)
  *Voie^Genre Diff V24c
  *ImpriSer^Parle = oui
Then
  *ImpriSer^Voie = *Voie
  /*Voie^Utilisation = imprimante
  /*Voie^NomLien = *ImpriSer^Nom
  /*Voie^Vitesse = *ImpriSer^Vitesse
Extern
  {ATTENTION toutes les V24 complètes sont déjà employées
    j'utilise une V24 simplifiée ou une voie de maintenance
    pour connecter cette imprimante cela peut être source de PROBLEMES.}
}

{InstRule - : BR49
If
  (UnKnown *Voie Terminal)
  (UnKnown *Voie Utilisation)
  *Voie^Genre = V24c
  *ImpriSer^Parle = non
Then
  *ImpriSer^Voie = *Voie
  /*Voie^Utilisation = imprimante
  /*Voie^NomLien = *ImpriSer^Nom
  /*Voie^Vitesse = *ImpriSer^Vitesse
Extern
  {L'imprimante est muette, toutes les V24 simplifiées ou
    de maintenance sont déjà utilisées alors on essaie de
    la connecter sur une V24 complète.}
}

{InstRule - : BR50
If
  (UnKnown *Voie Terminal)
  (UnKnown *Voie Utilisation)
  *Voie^Genre Diff V24c
  *ImpriSer^Parle = non
Then
  *ImpriSer^Voie = *Voie
  /*Voie^Utilisation = imprimante
  /*Voie^NomLien = *ImpriSer^Nom
  /*Voie^Vitesse = *ImpriSer^Vitesse
Extern
  {L'imprimante est muette on essaie de la connecter sur
    une V24 simplifiée ou une voie de maintenance.}
-- Interest : 100
}
```

```
{InstRule - : BR51
If
  (UnKnown *Voie Terminal)
  (UnKnown *Voie Utilisation)
  *Voie^Genre Diff VM
Then
  *Terminal^Voie = *Voie
  /*Voie^Terminal = *Terminal^Emul
  /*Voie^Utilisation = terminal
  /*Voie^Vitesse = *Terminal^Vitesse
Extern
  {On essaie d'affecter une V24 simplifiée ou complète pour ce terminal.}
-- Interest : 100
}

{InstRule - : BR52
If
  (UnKnown *Voie Terminal)
  (UnKnown *Voie Utilisation)
  *Voie^Genre = VM
Then
  *Terminal^Voie = *Voie
  /*Voie^Terminal = *Terminal^Emul
  /*Voie^Utilisation = terminal
  /*Voie^Vitesse = *Terminal^Vitesse
Extern
  {Il ne reste plus de V24 complètes ou simplifiées disponibles
    ALORS on utilise une voie de maintenance pour connecter ce terminal,
    ATTENTION cela va peut être créer des PROBLEMES.}
}


{InstRule - : BR53
If
  *PAD^EntSort = |entree|
Then
  *PAD^Coord = ""
Extern
  {Le Pad est en entrée
    ALORS les coordonnées de la machine cible sont égales à la chaîne vide.}
}


{InstRule - : BR54
If
  *PAD^EntSort = |inutilisee|
Then
  *PAD^Coord = "-"
  *PAD^Donnees = "-"
Extern
  {Le Pad est inutilisée
    ALORS il n'y a pas de machine cible,
    donc ses coordonnées et ses données sont égales à la chaîne vide.}
}
```

```
{InstRule - : BR55
If
  Cal^NbMETH = 0
Then
  Cal^NbRLOGIN = 0
Extern
  {Il n'y a pas de module d'échange Ethernet
    ALORS il n'y a pas non plus de possibilité de rlogin.}
}

{InstRule - : BR56
If
  Cal^NbMETH = 0
  Cal^NbMEX25 = 0
Then
  Cal^NbPAD = 0
Extern
  {Il n'y a pas de module d'échange Ethernet, ni X25 |
    ALORS il n'y  a pas non plus de voie PAD.}
}
```

Annexe D

UN EXEMPLE D'INSTALLATION

1 LA CONFIGURATION A INSTALLER

Nous allons présenter les objets qui modélisent un SPS7/300 appelé *smia* et les fichiers d'installation et d'administration créés par notre outil. Ce SPS7/300 dispose de :

- 4 Mo de mémoire
- un disque MXT190 partitionné de la manière suivante :
  - une partition système de 81000 blocs
  - une zone de swap de 19440 blocs
  - une partition *local* de 50220 blocs
  - une partition *users* de 50220 blocs
  - une partition dédiée à une base de données de 119880 blocs
- un terminal Bull dku5029 comme console système
- un module d'échange MEV24 qui gère six voies asynchrones. Quatre de ces voies sont utilisées pour connecter :
  - deux terminaux Ampex 210
  - une imprimante Bull pru1030
  - un micro-ordinateur compatible PC
- un module d'échange Ethernet pour lequel on définit quatre pseudo-terminaux

2 LES OBJETS

Cette configuration sera représentée par les objets suivants qui apparaissent dans leur ordre de création :

```
{Voie : Csl
  -- Indic : MT68
  -- Nom : "console"
  -- Genre : VM
  -- Utilisation : terminal
  -- Vitesse : console
  -- Terminal : "tvi950"
  -- Inittab : respawn
  -- NomLien : ()
  }

{UniteCentrale : UC
  -- CPU : M68020
  -- Memoire : 4
  -- Console : Csl
  }

{Calculateur : Cal
  -- Marque : Bull
  -- Modele : SPS7/300
  -- UniCen : UC
  -- NbMED : 1
  -- MED : [MED_1]
  -- NbMEV24 : 1
  -- MEV24 : [MEV_1]
  -- NbMETH : 1
  -- METH : [METH_1]
  -- NbMEX25 : 0
  -- MEX25 : []
  -- NbV24 : 8
  -- NbV24Libre : 3
  -- Bande : non
  -- NbRLOGIN : 4
  -- NbPAD : 0
  -- PAD : []
  }

{Peripherie : Perif
  -- NbTerm : 3
  -- Term : [term_3,term_2,term_1]
  -- NbImpSer : 1
  -- ImpSer : [impr_1]
  -- NbImpPar : 0
  -- ImpPar : []
  -- DestDefSpool : "pr1"
       Type : [|pas de defaut|,"pr1",1]
  -- NbMachV24 : 1
  -- MachV24 : [mach_1]
  }

{Materiel : Mat
  -- Calculateur : Cal
  -- Peripherie : Perif
  }
```

```
{Machine : Mai
  -- Nom : "smia"
  -- Materiel : Mat
  -- Utilisation : Uti
  -- REP : "/manip/Install"
  }
{Utilisation : Uti
  -- NbMaxUtilisateurs : ()
  -- Utilisateurs : []
  }

{Voie : tty30
  -- Indic : MED
  -- Nom : "tty30"
  -- Genre : VM
  -- Utilisation : inutilisee
  -- Vitesse : 9600
  -- Terminal : "un"
  -- Inittab : off
  -- NomLien : ()
  }

{Disk : Disque_31
  -- NumMED : 3
  -- Numero : 300
  -- Mineur : 96
  -- NumDiskPhys : "c3d0"
  -- Modele : MXT190
  -- Genre : dur
       Type : [dur,floppy]
  -- SecFormat : 256
  -- ModFormat : MXT190
  -- Format : |ne sais pas|
  -- Taille : 321032.
  -- PartUnit : 270
  -- Partition : oui
  -- ListPart : [Par_300_4,Par_300_3,Par_300_2,Par_300_1,Par_300_0]
  }

{Partition : Par_300_0
  -- Numero : 0
  -- Nom : "root"
  -- NumDisk : 300
  -- NumDiskPhys : "c3d0"
  -- PartUnit : 270
  -- Base : 0
  -- Taille : 81000.
  -- SysFic : oui
  -- Verif : -
  -- Montage : -
  -- Repertoire : "/"
  }
```

```
{Partition : Par_300_1
  -- Numero : 1
  -- Nom : "swap"
  -- NumDisk : 300
  -- NumDiskPhys : "c3d0"
  -- PartUnit : 270
  -- Base : 81000.
  -- Taille : 19440
  -- SysFic : non
  -- Verif : -
  -- Montage : -
  -- Repertoire : "-"
  }

{Partition : Par_300_2
  -- Numero : 2
  -- Nom : "local"
  -- NumDisk : 300
  -- NumDiskPhys : "c3d0"
  -- PartUnit : 270
  -- Base : 100440.
  -- Taille : 50220.
  -- SysFic : oui
  -- Verif : oui
  -- Montage : oui
  -- Repertoire : "/usr/local"
  }

{Partition : Par_300_3
  -- Numero : 3
  -- Nom : "users"
  -- NumDisk : 300
  -- NumDiskPhys : "c3d0"
  -- PartUnit : 270
  -- Base : 150660.
  -- Taille : 50220.
  -- SysFic : oui
  -- Verif : oui
  -- Montage : oui
  -- Repertoire : "/users"
  }

{Partition : Par_300_4
  -- Numero : 4
  -- Nom : "bdd"
  -- NumDisk : 300
  -- NumDiskPhys : "c3d0"
  -- PartUnit : 270
  -- Base : 200880.
  -- Taille : 119880.
  -- SysFic : non
  -- Verif : -
  -- Montage : -
  -- Repertoire : "-"
  }
```

```
{MED : MED_1
  -- Modele : Nsc800
  -- Numero : 3
  -- VM : tty30
  -- Streamer : oui
  -- NombreDisque : 1
  -- Disque : [Disque_31]
  }

{Voie : tty11
  -- Indic : MEV24
  -- Nom : "tty11"
  -- Genre : VM
  -- Utilisation : inutilisee
  -- Vitesse : 9600
  -- Terminal : "un"
  -- Inittab : off
  -- NomLien : ()
  }

{Voie : tty12
  -- Indic : MEV24
  -- Nom : "tty12"
  -- Genre : V24s
  -- Utilisation : inutilisee
  -- Vitesse : 9600
  -- Terminal : "un"
  -- Inittab : off
  -- NomLien : ()
  }

{Voie : tty13
  -- Indic : MEV24
  -- Nom : "tty13"
  -- Genre : V24s
  -- Utilisation : imprimante
  -- Vitesse : 4800
  -- Terminal : "un"
  -- Inittab : off
  -- NomLien : "prl"
  }

{Voie : tty14
  -- Indic : MEV24
  -- Nom : "tty14"
  -- Genre : V24c
  -- Utilisation : terminal
  -- Vitesse : 9600
  -- Terminal : "tvi925"
  -- Inittab : respawn
  -- NomLien : ()
  }
```

```
{Voie : tty15
  -- Indic : MEV24
  -- Nom : "tty15"
  -- Genre : V24c
  -- Utilisation : terminal
  -- Vitesse : 9600
  -- Terminal : "tvi925"
  -- Inittab : respawn
  -- NomLien : ()
  }

{Voie : tty16
  -- Indic : MEV24
  -- Nom : "tty16"
  -- Genre : V24c
  -- Utilisation : machine
  -- Vitesse : 9600
  -- Terminal : "h19"
  -- Inittab : respawn
  -- NomLien : "ttypc1"
  }

{MEV : MEV_1
  -- Numero : 1
  -- Majeur : 1
  -- BaseMineur : 16
  -- NombreVoie : 6
  -- Voies : [tty16,tty15,tty14,tty13,tty12,tty11]
  }

{METH : METH_1
  -- Modele : Bull
  -- Numero : 15
  -- Majeur : 20
  -- AdrEthernet : " "
  -- AdrInternet : "192.31.211.10"
  -- Voies : []
  }

{MachV24 : mach_1
  -- Modele : "PC"
  -- SysExp : "MS-DOS"
  -- LogCom : ["kermit"]
  -- Status : maitre
  -- TermEmu : "h19"
  -- Vitesse : 9600
  -- Nom : "pc1"
  -- Voie : tty16
       Askable : Yes
  }
```

```
{ImpriSer : impr_1
  -- Marque : "Bull"
  -- Modele : "pru1030"
  -- Nom : "pr1"
  -- ProgInter : "dumb"
  -- RepProg : "/usr/spool/lp/model"
  -- Classe : 1
  -- Interface : serie
  -- Vitesse : 4800
  -- Parle : non
  -- Voie : tty13
       Askable : Yes
  }

{Terminal : term_1
  -- Marque : "Bull"
  -- Modele : "dku5029"
  -- Emul : "tvi950"
  -- Vitesse : console
  -- Voie : Cs1
       Askable : Yes
  }

{Terminal : term_2
  -- Marque : "Ampex"
  -- Modele : "210"
  -- Emul : "tvi925"
  -- Vitesse : 9600
  -- Voie : tty15
       Askable : Yes
  }

{Terminal : term_3
  -- Marque : "Ampex"
  -- Modele : "210"
  -- Emul : "tvi925"
  -- Vitesse : 9600
  -- Voie : tty14
       Askable : Yes
  }
```

3 LES FICHIERS CREES

A partir de ces informations, notre logiciel construit trois sortes de fichiers :

- les fichiers *shl**, fichiers de commandes qui assurent la génération de la partition système destinée à la machine à installer

- les fichiers *config**, fichiers de commandes qui réalisent l'installation de la machine

- les fichiers *init**, fichiers d'administration du futur système.

Pour le SPS7/300 *smia*, notre logiciel génère les fichiers :

- *shlFichSpe* pour créer les fichiers spéciaux

```
# shell-script qui crée les fichiers spéciaux et les liens

/etc/mknod /manip/Install/dev/mem  c 3 0
/etc/mknod /manip/Install/dev/kmem c 3 1
/etc/mknod /manip/Install/dev/null c 3 2
/bin/chmod 440  /manip/Install/dev/mem /manip/Install/dev/kmem
/bin/chmod 666 /manip/Install/dev/null

/etc/mknod /manip/Install/dev/console c 0 0
/bin/ln /manip/Install/dev/console /manip/Install/dev/systty
/bin/ln /manip/Install/dev/console /manip/Install/dev/syscon
/etc/mknod /manip/Install/dev/tty30 c 1 48

echo Je crée les répertoires et les fichiers spéciaux pour le streamer
/etc/mkdir /manip/Install/dev/ct
/etc/mkdir /manip/Install/dev/rct
/etc/mknod /manip/Install/dev/ct/0  b 7 176
/etc/mknod /manip/Install/dev/ct/0n b 7 240
/etc/mknod /manip/Install/dev/rct/0  c 7 48
/etc/mknod /manip/Install/dev/rct/0n c 7 112
chmod 666 /manip/Install/dev/ct/* /manip/Install/dev/rct/*
echo Je crée les fichiers spéciaux et les liens pour le disque 300.
/etc/mknod /manip/Install/dev/dsk/c3d0 b 0 96
/etc/mknod /manip/Install/dev/rdsk/c3d0 c 4 96
/bin/ln /manip/Install/dev/dsk/c3d0 /manip/Install/dev/pd300
/bin/ln /manip/Install/dev/rdsk/c3d0 /manip/Install/dev/rpd300
echo
echo Je fais les fichiers spéciaux et les liens pour les partitions
echo du disque 300, MXT190.
echo
etc/mknod /manip/Install/dev/dsk/c3d0s0 b 0 128
etc/mknod /manip/Install/dev/rdsk/c3d0s0 c 4 128
echo
etc/mknod /manip/Install/dev/dsk/c3d0s1 b 0 129
etc/mknod /manip/Install/dev/rdsk/c3d0s1 c 4 129
echo
etc/mknod /manip/Install/dev/dsk/c3d0s2 b 0 130
etc/mknod /manip/Install/dev/rdsk/c3d0s2 c 4 130
/bin/ln /manip/Install/dev/vdlocal /manip/Install/dev/dsk/c3d0s2
/bin/ln /manip/Install/dev/rvdlocal /manip/Install/dev/rdsk/c3d0s2
echo
etc/mknod /manip/Install/dev/dsk/c3d0s3 b 0 131
etc/mknod /manip/Install/dev/rdsk/c3d0s3 c 4 131
/bin/ln /manip/Install/dev/vdusers /manip/Install/dev/dsk/c3d0s3
/bin/ln /manip/Install/dev/rvdusers /manip/Install/dev/rdsk/c3d0s3
echo
etc/mknod /manip/Install/dev/dsk/c3d0s4 b 0 132
etc/mknod /manip/Install/dev/rdsk/c3d0s4 c 4 132
/bin/ln /manip/Install/dev/vdbdd /manip/Install/dev/dsk/c3d0s4
/bin/ln /manip/Install/dev/rvdbdd /manip/Install/dev/rdsk/c3d0s4
```

```
/etc/mknod /manip/Install/dev/tty11 c 1 17
/etc/mknod /manip/Install/dev/tty12 c 1 18
/etc/mknod /manip/Install/dev/tty13 c 1 19
/etc/mknod /manip/Install/dev/tty14 c 1 20
/etc/mknod /manip/Install/dev/tty15 c 1 21
/etc/mknod /manip/Install/dev/tty16 c 1 22

echo je crée les liens existant sur les tty
/bin/ln /manip/Install/dev/tty16 /manip/Install/dev/ttypc1
/bin/ln /manip/Install/dev/tty13 /manip/Install/dev/pr1
#
echo Je crée des fichiers spéciaux pour la carte ethernet
/etc/mknod /manip/Install/dev/ethernet c 20 15
echo Je crée les fichiers spéciaux pour les 4 pseudo-terminaux
/etc/mknod /manip/Install/dev/ptyp0 c 12 0
/etc/mknod /manip/Install/dev/ttyp0 c 11 0
/etc/mknod /manip/Install/dev/ptyp1 c 12 1
/etc/mknod /manip/Install/dev/ttyp1 c 11 1
/etc/mknod /manip/Install/dev/ptyp2 c 12 2
/etc/mknod /manip/Install/dev/ttyp2 c 11 2
/etc/mknod /manip/Install/dev/ptyp3 c 12 3
/etc/mknod /manip/Install/dev/ttyp3 c 11 3
chmod 666 /manip/Install/dev/ptyp* /manip/Install/dev/ttyp*
```

- *shlFormat* pour définir le modèle de formatage des disques

```
#********* Crée les liens de formatage des disques ******************

echo
echo Je teste si /manip/Install/etc/format/MXT190 et
echo /manip/Install/etc/format/300 existent
utlCreFic /manip/Install/etc/format/MXT190
utlRmvFic /manip/Install/etc/format/rpd300
echo
echo Je fais le lien pour le formatage du disque 300, MXT190.
/bin/ln /manip/Install/etc/format/MXT190 /manip/Install/etc/format/rpd300
```

- *shlMkafs* pour créer le système de fichiers de la partition d'installation

```
# shlMkafs fichier de commandes pour créer la partition d'installation
##################################
echo
echo creation de la partition d'installation ou va etre
echo construite la partition systeme de la machine a installer
echo Je cree le systeme des fichiers pour la partition d'installation
etc/mkfs /dev/rvdinstall  81000
```

- *shlPlanAct* pour créer la partition système de la machine à installer

```
#*********** shell-script à lancer à la fin du kool ******************
# contient l'ordonnancement des autres shl-
$REP=/manip/Install;export REP
shlMkafs
echo on monte la partition d'installation sur /manip/Install
/etc/mount /dev/vdinstall $REP
echo creation des repertoires de la partition d'installation
utlRep
echo construction de la partition d'installation par des cpio
echo copie des fichiers d'installation et d'administration
utlCopi
echo creation des fichiers speciaux
/manip/InstallshlFichSpe
echo creation des liens de formatage
/manip/InstallshlFormat
echo creation du support d'installation
echo copie du boot sur le support
echo copie du boot du futur disque systeme
echo copie de la partition d'installation par dd
```

- *init_machconf* le fichier */etc/machconf* du système à installer

```
rm -f /dev/syst /dev/rsyst
rm -f /dev/swap /dev/rswap

set ""`/etc/devnm /`
DBOOT=`/etc/devnm / | sed -e "s/ .*$//"`
RDBOOT=`echo $DBOOT | sed -e "s?/dev/?/dev/r?"`
SBOOT=`echo $DBOOT | sed -e "s/.$/1/"`
RSBOOT=`echo $SBOOT | sed -e "s?/dev/?/dev/r?"`
/bin/ln $DBOOT /dev/syst
/bin/ln $RDBOOT /dev/rsyst
/bin/ln $DBOOT /dev/root
/bin/ln $RDBOOT /dev/rroot
/bin/ln $SBOOT /dev/swap
/bin/ln $RSBOOT /dev/rswap
/etc/vdset /dev/r$1 128
/etc/swapzone

/bin/uname -Nsmia
```

- *init_inittab* le fichier */etc/inittab* du système à installer

```
cf::bootwait:/etc/machconf 1>/dev/syscon 2>&1          system configuration
bl::bootwait:/etc/bcheckrc</dev/syscon> /dev/syscon 2>&1 # bootlog
bc::bootwait:/etc/brc 1>/dev/syscon 2>&1              # bootrun command
rc:123456:bootwait:/etc/rc 1>/dev/syscon 2>&1         # run command
lp:123456:wait:/etc/initlp 1>/dev/syscon 2>&1         # printer
sl::wait:/etc/initcons 1>/dev/syscon 2>&1          # console init
pf::powerfail:/etc/powerfail 1>/dev/syscon 2>&1       # power fail routines
16:23:respawn :/etc/getty -Th19 -sp tty16 9600 # MEV24 machine ttypcl
```

```
13:23:off :/etc/getty -Tun -sp tty13 4800 # MEV24 imprimante pr1
co::respawn :/etc/getty -Ttvi950 -sp console console # MT68 terminal
15:23:respawn :/etc/getty -Ttvi925 -sp tty15 9600 # MEV24 terminal
14:23:respawn :/etc/getty -Ttvi925 -sp tty14 9600 # MEV24 terminal
12:23:off :/etc/getty -Tun -sp tty12 9600 # MEV24 inutilisee
11:23:off :/etc/getty -Tun -sp tty11 9600 # MEV24 inutilisee
30:23:off :/etc/getty -Tun -sp tty30 9600 # MED inutilisee
```

- *init_checklist* le fichier */etc/checklist* du système à installer

```
/dev/rvdlocal
/dev/rvdusers
```

- *init_mountlist* le fichier */etc/mountlist* du système à installer

```
/etc/mount /dev/vdlocal /usr/local || echo refus de monter /dev/vdlocal
/etc/mount /dev/vdusers /users || echo refus de monter /dev/vdusers
```

- *init_hosts* l'amorce du fichier */etc/hosts* du système à installer

```
# ce fichier est à expédier aux autres machines du réseau
# et à concaténer à leur propre fichier "/etc/hosts"

192.31.211.10     smia

# il faudra le compléter par les autres machines du réseau
```

- *configFormat* pour réaliser le formatage des disques

```
# **********  FORMATAGE des Disques  **********
# **********  Disque_31  **********
# ********   MXT190    ********

if /etc/diskform -e /dev/rpd300 -g
then
     echo Le disque Disque_31 est formate.
else
    if /etc/diskform -f /dev/rpd300
    then
        echo Le disque Disque_31 est formate.
    else
        echo Formatage impossible. On arrete tout!
        exit 1
    fi
fi
```

- *configMkapar* pour définir le découpage des disques en partitions

```
#**************** Partitionnement des disques physiques **************

# Ce shell-script, construit pendant la session KOOL, ne s'exécute
# qu'une fois, lors de l'installation du système.

echo
echo J'enregistre le partitionnement du disk 300, MXT190.
etc/mkpar /dev/rdsk/c3d0 0 81000 81000 19440 100440 50220 150660 50220
200880 119880
```

- *configMkafs* pour définir les systèmes de fichiers des partitions

```
/etc/vdset /dev/rdsk/c3d0 128
echo
echo Je cree le systeme des fichiers pour la partition local du disque c3d0
etc/mkfs /dev/rdsk/c3d0s2 50220
/bin/mkdir /usr/local
etc/mount /dev/dsk/c3d0s2 /usr/local
echo
echo Je cree le systeme des fichiers pour la partition users du disque c3d0
etc/mkfs /dev/rdsk/c3d0s3 50220
/bin/mkdir /users
etc/mount /dev/dsk/c3d0s3 /users
```

- *configPlanAct* pour contrôler l'installation

```
# configPlanAct
# Plan d'actions à faire lors de l'installation.

if configFormat
then echo formatage termine
else echo on arrete l'installation
     exit 1
fi
echo On effectue le partitionnement
configMkapar
echo Partitionnement effectue
echo On installe les systemes de fichiers
echo On cree les repertoires
echo On monte les partitions
configMkafs
echo Systemes de fichiers installes
echo Installation des options
echo sauvegarde des disques
echo Installation terminee relancer le systeme en multi-utilisateurs
```

4 L'INSTALLATION

L'installation du SPS7/300 *smia* avec notre outil se déroule de la manière suivante :

- consultation, sur la machine dédiée, du système expert qui génère les fichiers d'installation et d'administration à partir des caractéristiques du SPS7/300 *smia*

- exécution, sur la machine dédiée, en mode mono-utilisateur, du fichier de commandes *shlPlanAct* qui assure :

    - la création de la partition où va être construite la partition système destinée au SPS7/300 *smia* (fichier *shlMkafs*)

    - le montage de cette partition

    - la création des principaux répertoires de l'arborescence UNIX (fichier *utlRep*)

    - la copie des fichiers d'administration et d'installation (fichier *utlCopi*)

    - la création des fichiers spéciaux (fichier *shlFichSpe*)

    - la création des liens de formatage (fichier *shlFormat*)

    - la sauvegarde de cette partition sur une bande ou une cassette

- boot du SPS7/300 *smia* sur cette bande et copie sur le disque système

- boot du SPS7/300 *smia* sur le disque système nouvellement créé

- exécution, en mode mono-utilisateur, du fichier de commandes *configPlanAct* qui réalise successivement :

    - le formatage des disques (fichier *configFormat*)

    - le partitionnement des disques (fichier *configMkapar*)

    - la création des systèmes de fichiers et des répertoires, le montage des partitions (fichier *configMkafs*)

    - l'installation des options

    - la sauvegarde des partitions

ECOLE NATIONALE SUPERIEURE
DES MINES DE SAINT-ETIENNE

N° d'ordre : 22 ID
1988

Philippe VIAL

Un outil d'aide à l'installation d'UNIX fondé sur les connaissances

Spécialité : Informatique, Image, Intelligence Artificielle et Algorithmique

Mots clés :

Représentation des connaissances, Programmation orientée vers les objets

Système expert, Génie logiciel, UNIX, aide à l'installation

Résumé :

Le succès d'UNIX est, pour une grande part, lié à la diffusion massive des stations de travail. Si UNIX fournit un bon environnement de travail, son installation et son administration constituent des tâches difficiles pour les utilisateurs finaux qui ne possèdent pas toujours les compétences nécessaires. Pour permettre à un non-spécialiste de la réaliser sans effort important, nous avons donc décidé de développer un outil d'aide à l'installation d'UNIX. A partir de la description d'une machine et de son utilisation, notre outil déduit la configuration du système UNIX à installer et génère les commandes pour la mettre en place.

Nous présentons, tout d'abord, les principaux aspects de l'administration et de l'installation d'un système UNIX sur une machine. Après le rappel des caractéristiques de trois des principaux paradigmes de programmation (procédures, objets, règles) et du mode de fonctionnement d'un moteur d'inférences, nous montrons comment les caractéristiques des nombreuses connaissances (matériels, logiciels, commandes UNIX...) à représenter nous ont conduits à développer un système expert avec un environnement de développement qui marie les trois paradigmes de programmation cités précédemment. Enfin nous présentons l'état actuel de notre prototype et plusieurs extensions envisageables.