CONSERVATOIRE NATIONAL DES ARTS ET METIERS

Centre agréé de GRENOBLE (CUEFA)

MEMOIRE

présenté en vue d'obtenir le

DIPLOME D'INGENIEUR CNAM

en INFORMATIQUE

par

Aldo MAZZILLI

Etude de la migration et de la portabilité des applications informatiques d'Unix vers Windows NT

Soutenu le 31 janvier 2000

JURY	Président	Mme Véronique DONZEAU-GOUGE
	Membres	M. Christian CARREZ M. Jacques COURTIN M. Hubert GARAVEL M. Radu MATEESCU

CONSERVATOIRE NATIONAL DES ARTS ET METIERS

Centre agréé de GRENOBLE (CUEFA)

MEMOIRE

présenté en vue d'obtenir le

DIPLOME D'INGENIEUR CNAM

en INFORMATIQUE

par

Aldo MAZZILLI

Etude de la migration et de la portabilité des applications informatiques d'Unix vers Windows NT

Les travaux relatifs à ce mémoire ont été effectués au sein de l'action VASY de l'INRIA sous la direction de M. Hubert Garavel.

Remerciements

Je tiens à remercier :

- Madame Véronique Donzeau–Gouge, Professeur au Conservatoire National des Arts et Métiers, pour avoir bien voulu présider le jury de ce mémoire ;
- Monsieur Jacques Courtin, Professeur à l'Université Pierre Mendès–France de Grenoble et responsable du cycle ingénieur CNAM en informatique de Grenoble ;
- Monsieur Christian Carrez, Professeur au Conservatoire National des Arts et Métiers, pour sa participation au jury ;
- Monsieur Hubert Garavel, Chargé de Recherche à l'Institut National de Recherche en Informatique et en Automatique, qui n'a pas compté son temps pour diriger mon travail et me conseiller dans les décisions stratégiques ; je le remercie également pour le choix de ce sujet et pour son accueil sympathique au sein de son équipe ;
- Monsieur Alain Landelle, Responsable du Département Formations Supérieures au Centre Universitaire d'Education et de Formation des Adultes de Grenoble, pour la confiance qu'il m'a accordée dans la poursuite du cycle ingénieur CNAM ;
- Monsieur Radu Mateescu, Chargé de Recherche à l'Institut National de Recherche en Informatique et en Automatique, qui a bien voulu examiner mon travail ;
- L'ensemble de l'équipe VASY avec laquelle j'ai eu de nombreux échanges fructueux ;
- La société Global Automotive GBM S.A., pour m'avoir accordé un Congé Individuel de Formation qui m'a permis d'effectuer ce stage ;
- Enfin, Dorothée mon épouse, pour la patience et le soutien dont elle a su faire preuve à mon égard.

Enfin, je souhaite remercier toutes les personnes qui, de près ou de loin, ont contribué à la réalisation de ce mémoire. A tous, j'adresse ma sincère reconnaissance.

Table des matières

Chapitre 1

Introduction

1.1 Situation technique

Le marché de l'informatique impose de plus en plus aux applications logicielles de fonctionner sur une grande variété de systèmes d'exploitation.

Les concepteurs de logiciels doivent fournir des applications fonctionnant à la fois sous UNIX et WINDOWS NT de MICROSOFT. Ils sont concernés par cette nouvelle contrainte du marché pour des raisons aussi variées que leur vécu professionnel, leurs objectifs commerciaux et la prise en compte des nouvelles tendances. Toutes ces raisons influencent leur situation présente et leurs directions futures.

La maintenance d'autant de codes sources que d'architectures potentielles génère de nombreux problèmes financiers et organisationnels, notamment lorsqu'il s'agit de la formation des programmeurs.

La meilleure solution consiste à ne maintenir qu'un seul code source fonctionnant sur toutes les architectures proposées. Cette solution implique que le code soit entièrement portable et nécessite, donc, un certain nombre d'aménagements qui constituent le travail de portage.

La réussite d'un projet de migration repose sur une très bonne compréhension du système hôte et du système cible (respectivement UNIX et WINDOWS NT, dans notre cas), une planification minutieuse et une bonne connaissance des environnements de programmation.

1.1.1 Le système hôte : UNIX

Le système UNIX est un système d'exploitation né au laboratoires Bell d'AT&T sous l'impulsion de Ken Thompson et d'un petit groupe de chercheurs, d'abord pour un usage interne. Rapidement, deux grandes familles de systèmes UNIX se sont diffusées : SYSTEM V d'AT&T et BSD de l'Université de Berkeley [Rif91].

Le développement du langage C[1] [Rit84] a permis l'écriture du système UNIX dans ce langage

[1] par Kernighan et Ritchie

et l'a rendu portable sur tout type de machine disposant d'un compilateur C.

Un certain nombre de versions du système ont été développées à partir des sources fournis par AT&T. Certaines d'entre elles prennent en compte les standards définis par le comité POSIX de l'IEEE[2] [IEE90].

Le système UNIX est un système multi-utilisateurs et multi-tâches. En tant que système d'exploitation, son rôle principal est d'assurer aux différentes tâches et différents utilisateurs, une bonne répartition des ressources de l'ordinateur (mémoire centrale, espace de stockage disque, processeur, imprimante, etc). Lorsque les demandes sont trop importantes, le système est conçu pour ne pas bloquer l'accès aux ressources en réduisant le temps d'utilisation par tâche et/ou utilisateur (il s'agit du principe de *Time-Sharing*). L'application de ce principe conduit à ralentir (plus ou moins) le système.

UNIX est par ailleurs un système de développement et ses utilisateurs ont à leur disposition un certain nombre d'outils d'édition de texte, de compilation et de mise au point leur permettant de développer et documenter leurs programmes.

Le système UNIX est composé de :

- un noyau assurant la gestion de la mémoire interne, des entrées/sorties et des processus,
- un ou plusieurs interpréteurs de langages de commande appelés *Shells*. Les plus répandus de ces Shells sont le Bourne Shell, le Korn Shell (du nom de leurs auteurs) et le C-Shell développé à l'Université de Berkeley,
- un ensemble de protocoles : TCP/IP,
- un grand nombre de programmes utilitaires tels que des compilateurs (langages C, Fortran), des éditeurs de texte (emacs, vi), des logiciels de communication (ftp, rsh, rcp), des générateurs d'analyseurs lexicaux et syntaxiques (lex, yacc), etc.

1.1.2 Le système cible : WINDOWS NT

MICROSOFT WINDOWS NT [Sol98] [Cus93] est un système d'exploitation réseau. Son objectif est de mettre à disposition des utilisateurs, par le biais de leurs postes de travail, les différentes ressources informatiques nécessaires à la réalisation de leur travail. MICROSOFT WINDOWS NT a été conçu pour supporter à la fois les fonctions de partage de ressources en réseau (fichiers, imprimantes, outils de sauvegardes, fax, etc), de serveur d'applications (client-serveur) et de serveur Internet/Intranet.

Conceptuellement, WINDOWS NT consiste en la combinaison de différents modules.

- un module client/serveur qui fournit les différents environnements de système d'exploitation, également nommés sous-systèmes protégés,
- un module exécutif (*NT executive*) pour gérer uniformément les ressources système de façon transparente pour l'utilisateur,

[2] The Institute of Electrical and Electronics Engineers, Inc. `www.ieee.org`

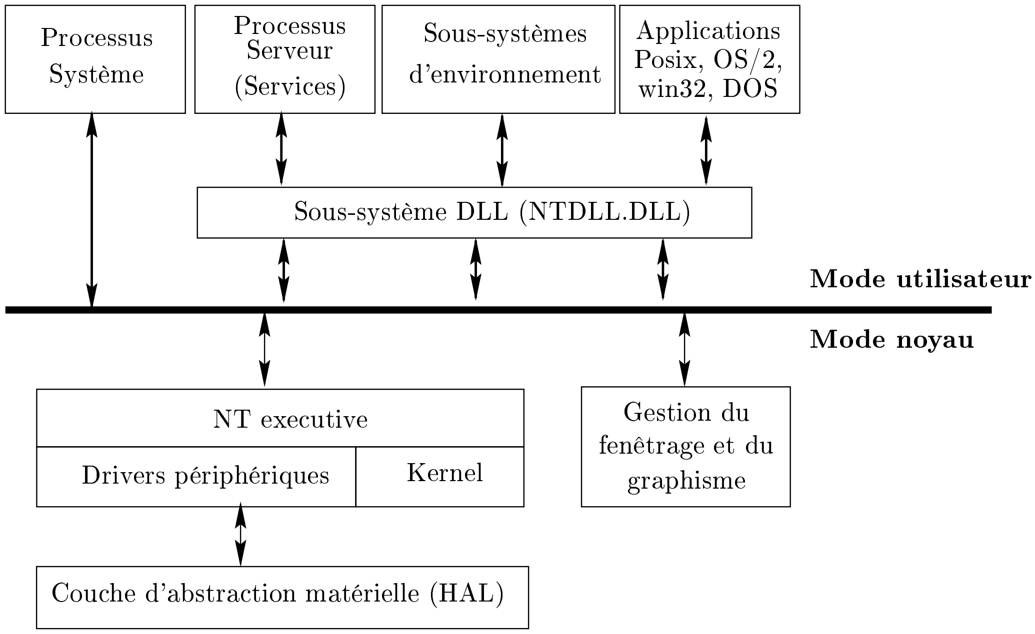

Figure 1.1: Architecture de WINDOWS NT

- un module de multitraitement symétrique (SMP) pour exploiter les systèmes multi-processeurs.

De plus, afin de protéger l'accès par l'utilisateur à des données critiques du système, WINDOWS NT utilise deux modes bien distincts :

- Le mode utilisateur (*User mode*) : mode dans lequel fonctionnent les programmes de l'utilisateur (privilèges restreints).
- Le mode noyau (*Kernel mode*) : mode dans lequel fonctionnent les programmes du système (niveau élevé de privilèges).

Dans la suite, nous présenterons les différentes composantes du système WINDOWS NT agissant dans chacun de ces deux modes (voir figure 1.1).

Le modèle client/serveur : La structure de WINDOWS NT est basée sur un modèle client/serveur en couches. Dans cette configuration, le système d'exploitation est divisé en différents processus, appelés serveurs, dont chacun met en œuvre un ensemble unique de services tels que les services de gestion mémoire, de création de processus et d'ordonnancement des processus.

Chaque serveur fonctionne en mode utilisateur, exécutant une boucle qui contrôle si un client a requis l'un de ses services. Un client est un autre composant du système d'exploitation ou une application utilisateur qui établit une requête de service en postant un message au serveur. Le noyau du système (fonctionnant en mode noyau) délivre le message au serveur qui exécute l'opération, attend la fin du service demandé et rend ensuite le résultat au client par le biais d'un autre message. Parce que chaque serveur fonctionne dans un mode utilisateur séparé et partitionné (il gère sa propre mémoire), l'échec d'un processus serveur ne peut pas corrompre le reste du système d'exploitation.

WINDOWS NT utilise ce modèle client/serveur essentiellement pour fournir les API[3] du système. En plaçant chaque API dans un serveur séparé, WINDOWS NT élimine les conflits et permet l'implémentation aisée de nouvelles API. Ces processus serveurs sont appelés des sous-systèmes protégés car leur mémoire de traitement est protégée des autres processus grâce au système de mémoire virtuelle géré par l'exécutif NT.

Exécutif NT : Les flots de données en mode noyau sont gérés par l'exécutif NT. Ce dernier est composée de différents éléments qui mettent en œuvre la gestion des entrées/sorties et des systèmes de fichiers, la communication inter-processus et le système de sécurité. Ces composants communiquent entre eux de façon modulaire à travers des routines clairement spécifiées dans un système de messagerie interne.

La partie la plus basse du mode noyau est la couche d'abstraction matérielle (*Hardware Abstraction Layer* ou HAL). Il s'agit d'une couche de code (DLL[4]) qui protège le noyau et le reste de l'exécutif NT des différences matérielles et des spécificités des plate-formes. La couche HAL manipule directement le matériel, ce qui rend WINDOWS NT portable sur diverses plate-formes.

Multiprocessing et multi-tâches : WINDOWS NT est un système multi-utilisateurs et multi-tâches, incluant un ordonnancement préemptif et avec priorités des processus. WINDOWS NT inclut la gestion de tâches[5] dans un processus.

Dans le mode utilisateur, WINDOWS NT gère un ensemble d'environnements par le biais de sous-systèmes protégés, afin de fournir à l'utilisateur et/ou programmeur les caractéristiques systèmes principales. Dans la suite, nous voyons quels sont ces sous-systèmes, nous détaillons particulièrement l'un d'eux (le sous-système WIN32).

Les sous-systèmes protégés : WINDOWS NT met en œuvre deux types de sous-systèmes protégés :

- Les sous-systèmes d'environnement :
  * Le sous-système WIN32 : il est le plus important des sous-systèmes d'environnement. Il fournit toute l'API 32 bits, l'interface graphique et controle toutes les entrées utilisateurs et les sorties applicatives.
  * Le sous-système OS/2 : il permet le fonctionnement d'application OS/2 en mode caractère uniquement.

[3] Application Program Interface
[4] Dynamic Link Library
[5] threads

  * Le sous-système POSIX : il met en œuvre un environnement UNIX basé sur la norme POSIX 1003.1 (POSIX.1) [IEE90]. Nous verrons un peu plus loin, et dans le cadre d'une stratégie possible de migration, pourquoi son implémentation n'est pas assez complète pour réaliser un portage réaliste d'applications UNIX.
  * Le sous-système MS-DOS : il offre la compatibilité binaire des applications MS-DOS à travers un émulateur appelé machine virtuelle DOS (VDM[6]).
- Les sous-systèmes intégrés :
  * Le sous-système sécurité : il maintient une base de données sur les utilisateurs et effectue les traitements de connexion et d'authentification.
  * Le sous-système réseaux : WINDOWS NT met en œuvre une gestion de réseaux basée sur des sockets et un environnement de traitement distribué d'appels de procédures distantes (RPC[7]). Il supporte plusieurs protocoles, dont TCP/IP.

L'API WIN32 : WIN32 est l'API principale de WINDOWS NT. Elle supporte la gestion du modèle 32 bits. WIN32 est une extension de l'API 16 bits utilisée pour développer des applications graphiques (GUI[8]) WINDOWS 3.1 basées sur la couche MS-DOS. Les applications WINDOWS en mode 16 bits continuent de fonctionner sous WINDOWS NT.

Les DLL : Une DLL (bibliothèque dynamique) reprend l'idée d'une bibliothèque statique avec une dimension supplémentaire.

Une bibliothèque statique consiste en un ensemble de fonctions qu'un groupe de programmes est susceptible d'utiliser. Cela signifie qu'un programmeur écrit le code pour une certaine tâche, une seule fois et peut ensuite utiliser la même fonction dans d'autres programmes qui ont besoin de réaliser la même tâche. Cependant, avec une bibliothèque statique, l'éditeur de liens inclut dans le programme exécutable tous les objets (contenant fonctions et données) nécessaires à son exécution. Chaque programme inclut donc sa propre copie de toutes les bibliothèques qu'il est susceptible d'utiliser.

Une extension naturelle à l'idée de bibliothèque statique est de placer le code correspondant aux fonctions dans une bibliothèque dynamique faisant en sorte que le lien s'établisse à l'exécution et non durant l'édition de liens (d'où le terme dynamique). Une DLL est un code objet binaire qui, au lieu d'être exécutable par un utilisateur pour une certaine tâche, est constitué de fonctions "exportables" de façon à ce que d'autres programmes puissent les appeler. Ce concept comporte plusieurs avantages. En premier lieu, il y a une seule copie de la DLL, utilisable par toutes les applications qui ont besoin de partager les mêmes fonctions ; chaque application étant ainsi plus économe en place disque. Egalement, s'il existe une erreur dans une DLL, il suffit de corriger cette DLL pour que chaque application qui l'utilise bénéficie de la correction.

[6]Virtual DOS Machine
[7]Remote Procedure Call
[8]Graphic User Interface

1.2 Cadre institutionnel

1.2.1 Le projet VASY

Créée au 1er janvier 1997 à l'INRIA Rhône–Alpes, l'action VASY "Recherche et Applications" s'inscrit dans la problématique de la conception de systèmes sûrs par l'utilisation de méthodes formelles. Plus précisément, VASY s'intéresse à tout système (matériel, logiciel, protocoles de télécommunication) faisant intervenir du parallélisme asynchrone, une modélisation du parallélisme basée sur la sémantique d'entrelacement (*interleaving semantics*) et bien adaptée à la description de systèmes répartis.

Pour la conception de systèmes sûrs, VASY préconise l'utilisation des techniques de description formelle, complétées par des outils informatiques adaptés, offrant des fonctionnalités de simulation, prototypage rapide, vérification et génération de tests. Parmi les différentes approches existantes pour la vérification, VASY concentre ses efforts sur la vérification "basée sur les modèles" (*model-checking*) qui recouvre un grand nombre de techniques spécialisées (vérification énumérative, à la volée, symbolique, etc). Ces techniques, bien que moins générales que les approches par preuves (*theorem proving*), possèdent pourtant l'avantage de permettre une détection automatique, rapide et économique des erreurs de conception dans des systèmes complexes.

Les travaux de VASY se situent au confluent de deux grandes approches en méthodes formelles : l'approche basée sur des modèles (automates, réseaux de Petri) et l'approche basée sur des langages. Il est indispensable de s'appuyer sur des formalismes de plus haut niveau (c'est-à-dire des langages) permettant de décrire des problèmes réels et complexes sous forme de programmes. Ces programmes sont ensuite analysés et traduits automatiquement vers des modèles sur lesquels opèrent les algorithmes de vérification. Pour mener à bien la vérification de systèmes complexes (de taille "industrielle"), il semble nécessaire de maîtriser simultanément la technologie des modèles et celle des langages.

1.2.2 Les domaines d'application de l'équipe VASY

Les modèles théoriques utilisés (automates, algèbres de processus, bisimulations, logiques temporelles) et les logiciels développés sont suffisamment généraux pour ne pas dépendre trop étroitement d'un seul secteur applicatif. Les méthodes de VASY peuvent s'appliquer à tout système ou protocole composé d'agents distribués communiquant par messages. Ce cadre conceptuel trouve de nombreuses incarnations dans le domaine du logiciel, du matériel et des télécommunications. Les études de cas récemment conduites avec les outils du projet VASY illustrent bien cette diversité applicative :

- architectures multiprocesseur : arbitrage de bus, cohérence de caches ;
- bases de données : protocoles transactionnels, bases de connaissances distribuées, gestion de stocks ;

- électronique de consommation : télécommandes audiovisuelles, vidéo à la demande, bus FIREWIRE ;
- protocoles de sécurité : commerce électronique, distribution de clés cryptographiques ;
- systèmes embarqués : communications entre avions et tours de contrôle ;
- systèmes répartis : mémoire virtuelle, systèmes de fichiers répartis, ingénierie concourante, algorithmes d'élection ;
- télécommunications : réseaux à haut débit, administration de réseaux, interactions de services téléphoniques ;
- interactions homme-machine : interfaces graphiques, visualisation de données biomédicales, etc.

1.2.3 La boîte à outils CADP

En collaboration avec le laboratoire Verimag, Vasy développe la boîte à outils Cadp (Caesar/Aldebaran Development Package) pour l'ingénierie des protocoles et des systèmes distribués (voir `http:/www.inrialpes.fr/vasy/cadp`).

Au sein de cette boîte à outils, Vasy a en charge les logiciels suivants :

- Cæsar est un compilateur qui produit, à partir d'un programme Lotos (langage de spécification normalisé [ISO88]), du code exécutable ou des modèles sur lesquels différentes méthodes de vérification peuvent être appliquées. Le programme source Lotos est traduit successivement en une algèbre de processus simplifiée, un réseau de Pétri étendu avec des variables et des transitions atomiques, et, finalement, un système de transitions étiquetées obtenu par simulation exhaustive du réseau de Pétri.
- Cæsar.adt est un compilateur qui traduit les définitions de type abstraits Lotos vers des bibliothèques de types et de fonctions en langage C. La traduction met en œuvre un algorithme de compilation par filtrage et des techniques pour la reconnaissance des classes de types usuels (nombres entiers, énumérations, tuples, listes, etc.) qui sont identifiées automatiquement et implémentées de manière optimale.
- Bcg (Binary Coded Graphs) est un format qui utilise des techniques efficaces de compression permettant de stocker des graphes (représentés sous forme explicite) sur disque de manière très compactée. Ce format est indépendant du langage source et des outils de vérification. En outre, il contient suffisamment d'informations pour que les outils qui l'exploitent puissent fournir à l'utilisateur des diagnostics précis dans les termes du programme source. Pour exploiter ce format, un environnement logiciel est disponible, qui se compose de bibliothèques C et de plusieurs outils, notamment : Bcg_Io (qui effectue des conversions de format), Bcg_Open (qui permet d'appliquer à des graphes BCG les outils de l'environnement Open/Cæsar pour la vérification à la volée), Bcg_Draw (qui permet d'afficher en PostScript une représentation 2D

d'un graphe) et BCG_EDIT (qui permet de modifier interactivement la représentation graphique produite par BCG_DRAW).

- OPEN/CÆSAR est un environnement extensible permettant de développer des outils de simulation, de vérification et de génération de tests sur des graphes représentés sous forme implicite. Ces outils peuvent être réalisés de manière simple, modulaire et indépendante du langage utilisé pour décrire les systèmes à valider. L'environnement OPEN/CÆSAR comprend un ensemble de bibliothèques avec leurs interfaces de programmation, ainsi que divers outils pour la simulation pas à pas, l'exécution aléatoire, la recherche de blocages, la recherche de séquences satisfaisant un certain critère, etc.

- XTL (eXecutable Temporal Langage) est un méta-langage adapté à l'expression des algorithmes d'évaluation et de diagnostic pour les formules de logiques temporelles. D'inspiration fonctionnelle, ce méta-langage offre des primitives d'accès à toutes les informations contenues dans les graphes BCG : états, étiquettes des transitions, fonctions successeurs et prédécesseurs, ainsi qu'aux types et fonctions du programme source. Il permet la définition de fonctions récursives servant à calculer des prédicats de base et des modalités temporelles portant sur les ensembles d'états et de transitions.

Tous ces outils ainsi que d'autres développés par les projets MEIJE (INRIA Sophia-Antipolis) et PAMPA (INRIA Rennes) et par les Universités de Liège et d'Ottawa sont intégrés au sein de l'interface graphique EUCALYPTUS (développée en TCL/TK) qui offre un accès facile et uniforme aux différents outils, en cachant à l'utilisateur les conventions d'appel et les formats spécifiques à chaque outil.

Depuis les années 80 et le début des années 90, les méthodes formelles ont connu un grand essor : de multiples langages, outils et méthodologies ont été proposés. Cette phase d'expansion semble sur le point de s'achever et une phase de sélection s'apprête à lui succéder. Les langages inadaptés et les prototypes immatures seront délaissés, au profit d'outils qui auront fait leurs preuves sur des exemples industriels et pour lesquels l'existence d'une communauté importante d'utilisateurs permettra d'assurer les développements futurs.

La boîte à outils CADP est bien placée dans cette compétition. Elle s'appuie sur un langage normalisé (langage LOTOS de l'ISO), comporte des outils robustes et regroupe un nombre important d'utilisateurs.

1.3 Travail effectué

L'essentiel du travail effectué dans le cadre de ce mémoire, traite de la migration des applications développées par l'équipe VASY sur une plate-forme UNIX vers l'architecture WINDOWS NT.

Concrètement, cette tâche consiste à apporter les méthodes et outils qui permettront à l'équipe VASY de développer une version pour WINDOWS NT de la boîte à outils CADP.

CADP est un logiciel volumineux (plusieurs centaines de milliers de lignes de code écrites dans différents langages de programmation : C, Tcl/Tk, shells Unix, Makefiles, Lex/Yacc, Syntax/Fnc-2, Lotos, manuels au format Nroff, documentation en LaTeX, etc).

De plus, Cadp utilise plusieurs caractéristiques système d'Unix : lancement de sous-processus, bibliothèques de code binaire, édition de liens, variables d'environnement, etc. On peut donc supposer que Cadp constitue un exemple représentatif assez complet des problèmes posés par la migration des applications Unix vers Windows NT.

Plus généralement, une des motivations du projet est de tirer les leçons de l'expérience acquise lors de la migration de Cadp. Il s'agira de définir une méthodologie pour l'écriture d'applications portables, capables de fonctionner aussi bien sous Unix que sous Windows NT.

Cette étude a pour objectif de :

- recenser les problèmes potentiels liés au portage,
- de réaliser un comparatif des solutions commerciales existantes,
- de mettre en œuvre une technique (organisationnelle et/ou logicielle) permettant de réaliser le portage des applications.

1.4 Organisation du document

Ce mémoire est organisé comme suit :

- Le chapitre 2 (*Objectifs et contraintes*) présente les objectifs de notre travail et l'environnement dans lequel nous avons œuvré pour les atteindre.
- Le chapitre 3 (*Solutions et choix*) détaille plus en profondeur chacun des sous-objectifs et donne les éléments de choix qui nous ont conduit à opter pour les différentes solutions retenues.
- Le chapitre 4 (*Réalisation*), plus technique que les précédents, donne une vision des méthodes employées et du travail réalisé.
- Enfin, le chapitre 5 (*Conclusion*) effectue un retour sur les chapitres précédents pour en dégager le bilan et les perspectives.

Chapitre 2

Objectifs et contraintes

Ce chapitre présente les objectifs à atteindre (section 2.1) ainsi que l'existant et les contraintes à respecter. Après avoir démontré la diversité de l'existant (section 2.2), il en détaillera le contenu en distinguant les environnements de développement (section 2.3), d'une part, et les outils (section 2.4), d'autre part. Il dresse également une liste des problèmes potentiels liés au portage en présentant les différences essentielles entre WINDOWS NT et UNIX (section 2.5).

2.1 Objectifs à atteindre

Dans cette section, il sera question des motivations qui ont conduit les responsables de l'action VASY à réaliser le portage de leurs applications vers WINDOWS NT.

A l'initiative du projet, il était indispensable de prendre en considération une demande croissante du public à utiliser l'environnement WINDOWS NT compte tenu des nombreux avantages qu'il proposait :

- un coût faible,
- un large éventail d'applications existantes,
- une multitude d'experts,
- une interface graphique attrayante.

Dans cette optique, il devenait indispensable de proposer une solution fiable aux utilisateurs de CADP désireux d'exploiter le système WINDOWS NT, sans perdre de vue les objectifs suivants :

- réaliser un minimum de changement dans le code source,
- conserver des temps de réponse cohérents,
- maîtriser l'environnement WINDOWS NT.

2.2 Diversité de l'existant

La difficulté de la migration des applications réside dans la diversité de l'existant. En effet, CADP se compose de nombreux programmes (plusieurs centaines de milliers de lignes de code) écrits dans plusieurs langages de programmation. Bien qu'une majorité de ces programmes soit écrite en langage C, la boîte à outils CADP utilise également toute la puissance de l'environnement de programmation UNIX, notamment :

- les Makefiles,
- les programmes écrits en Lex : analyseur lexical,
- les programmes écrits en Yacc : analyseur syntaxique.

De plus, CADP est composé de nombreux scripts Shell (Bourne Shell et C-Shell).

Enfin, pour parfaire l'ergonomie de la boîte à outils, tous les programmes sont accessibles à travers deux interfaces graphiques écrites en TCL et qui utilisent également les extensions TK, TIX et EXPECT :

- INSTALLATOR est le programme qui permet d'assister l'utilisateur dans l'installation de la boîte à outils CADP. La figure 2.1 montre l'architecture de cet outil installé par un programme auto-extractible nommé "`installator.shar`"

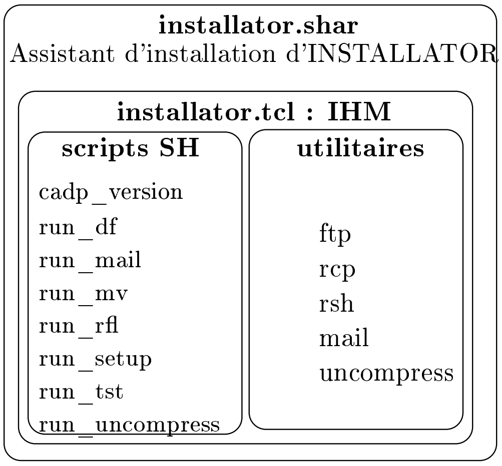

Figure 2.1: INSTALLATOR

- EUCALYPTUS est l'interface graphique qui fédère l'utilisation des programmes de la boîte à outils CADP. Il évite de frapper des commandes parfois fastidieuses en les rendant accessibles à travers des menus et des fenêtres développées grâce au langage TCL et son extension TK. Le lancement d'EUCALYPTUS est réalisé par le script `xeuca` dont le but est de positionner les variables d'environnement nécessaires au bon fonctionnement de CADP.

La figure 2.2 montre l'architecture d'EUCALYPTUS et présente les outils qui le composent. EUCALYPTUS invoque un certain nombre de scripts shell (`sh`) et de binaires CADP qui eux-mêmes appellent d'autres binaires (binaires invoqués indirectement). De plus, EUCALYPTUS permet à l'utilisateur d'accéder à un certain nombre d'utilitaires tels que `ghostview` pour la consultation de documents au format Postscript ou `netscape` pour la consultation de documents au format HTML.

xeuca : script SH

eucalyptus.tcl : TCL/TK

scripts SH

xeuca_convert
xeuca_echo
xeuca_fc2info
xeuca_info
xeuca_mail
xeuca_man
xeuca_ps
xeuca_version

binaires CADP

aldebaran	caesar.indent	fc2compare
apero-syn	caesar	fc2generate
apero-lib	xeludo	atg
bcg_draw	lotos	caesar.open
bcg_edit	bridge	exp.open
bcg_labels	fc2implicit	bcg_open
caesar.adt	fc2reduce	xtl

utilitaires

ghostview lpr mail man netscape sh vi xterm

binaires invoqués indirectement

bcg_expand	exp2c
bcg_io	exp2fc2
bcg_lib	hostinfo
des2aut	xtl_expand

Figure 2.2: EUCALYPTUS

Tous ces éléments donnent, de par leur hétérogénéité, un caractère complexe au présent projet de migration.

2.3 Langages des environnements de développement

2.3.1 Les scripts Shell

Un Shell (interpréteur Shell) [Qui97] est un programme spécial utilisé comme interface entre l'utilisateur et le cœur du système UNIX, appelé le Kernel. Le Kernel est chargé en mémoire au moment du boot et il gère le système jusqu'à son arrêt. Il crée et contrôle les processus, gère la mémoire, les systèmes de fichiers, les communications, etc.

Le Shell est un programme utilitaire qui démarre à la connexion de l'utilisateur. Il autorise les utilisateurs à interagir avec le Kernel en interprétant des commandes tapées sur la ligne de commande ou dans un script. Les trois principaux Shell supportés sur les systèmes UNIX sont :

- le Bourne Shell (AT&T) (`sh`),
- le C Shell (Berkeley) (`csh`),
- le Korn Shell (sur-ensemble du Bourne Shell) (`ksh`).

Tous trois mettent à disposition les mêmes caractéristiques mais ont quelques différences de syntaxe et de performance lorsqu'ils sont utilisés comme langages de script.

CADP est composé de nombreux scripts Shell écrits pour la plupart en Bourne Shell (`sh`). Quelques scripts sont écrits en C Shell (`csh`).

Voyons, maintenant, quelles sont les commandes Shell utilisées et les propriétés attendues dans la boîte à outils CADP.

Commandes Shell utilisées

Le tableau suivant recense les commandes utilisées dans les divers scripts de la boîte à outils CADP, ainsi que les options avec lesquelles elles sont associées {entre accolades}.

awk	basename	cat	cp
cpp {-M}	cut {-d -f}	date	dirname
echo	egrep {-v}	fgrep	file
gcc {-M}	grep {-v -s}	hostname	lex
lpr	ls {-lg}	mail	make
man	mkdir	more	mv
pr	ps {-x}	read	rcp
rm {-rf}	rmdir	rsh	sed {-e}
sort {-f -u}	tail	test	tr
trap	uname {-msr}	uncompress	wc {-w}
whoami	yacc		

Dans le présent projet de portage, il faudra garantir une solution fidèle à l'utilisation de chacune de ces commandes.

Propriétés attendues

Outre les différentes commandes citées ci-dessus, afin que le fonctionnement de CADP soit assuré, le shell devra présenter les propriétés (ou règles syntaxiques) suivantes :

Les métacaractères du Shell. Ils permettent :

- de construire des chaînes de caractères génériques. Exemple : `*` désigne une chaîne de caractères quelconque, `?` désigne un caractère quelconque.
- de modifier l'interprétation d'une commande. Exemple : `;` sépare deux commandes sur une même ligne, `‘cmd‘` capture la sortie standard de la commande “cmd” pour former un nouvel argument ou une nouvelle commande.

Les expressions régulières : on appelle “expression régulière”, une chaîne de caractère composée de caractères normaux et de caractères spéciaux appelés “métacaractères des expressions régulières” (à ne pas confondre avec les métacaractères du Shell décrits plus haut). Elles servent à extraire un motif de chaîne d'une chaîne de caractère quelconque.

Les redirections de l'entrée et de la sortie standard : utilisation des caractères >, <, >>, <<, | (tube).

Les redirections de la sortie standard des erreurs : utilisation des caractères `2>` et `2>>`.

Les scripts et le “Shebang” : un script est un ensemble de commandes UNIX rassemblées dans un fichier. Lorsque les deux premiers octets de ce fichier sont `#!`, cela indique que le chemin qui suit est celui du shell qui va interpréter le présent script. Exemple : `#!/bin/sh` indique que le script sera interprété par le Bourne Shell (`sh`). On appelle cette notation le “Shebang”.

Les paramètres positionnels : `$0, $1 ... $9` désignent les paramètres positionnels, à l'appel d'un script. `$0` désigne le nom du script, `$#` désigne le nombre de paramètres et `$*` la liste des paramètres positionnels. Enfin, `$?` contient le code de retour de la dernière commande exécutée.

L'opérateur `&&` : dans la commande “`cmd1 && cmd2`”, l'exécution de “cmd2” est réalisée à condition que “cmd1” ait le code de retour 0.

L'opérateur `||` : dans la commande “`cmd1 || cmd2`” l'exécution de “cmd2” est réalisée à condition que “cmd1” ait un code de retour différent de 0.

La structure de contrôle de condition :

```
if condition1
then
        liste commandes 1
elif condition2
then
        liste commandes 2
else
        liste commandes 3
fi
```

La structure de contrôle d'aiguillage à choix multiples :

```
case variable in
cas1 )
        liste commandes 1 ;;
cas2 )
        liste commandes 2 ;;
cas3 | cas4 )
        liste commandes 3 ;;
* )
        liste commandes 4 ;;
esac
```

La structure de contrôle de boucle "For" :

```
for variable in liste_de_cas
do
        liste commandes
done
```

La structure de contrôle de boucle "While" :

```
while condition
do
        liste commandes
done
```

La structure de contrôle de boucle "Until" :

```
until condition
        do
        liste commandes
done
```

Le lancement d'un shell fils : toute séquence de commandes placée entre parenthèses est exécutée par un "sous shell" ou "shell fils" avec les propriétés suivantes :

- Tout déplacement dans l'arborescence des répertoires et toute modification de la valeur d'une variable ne sont plus effectifs dès le retour dans le shell père.
- L'entrée et la sortie standard d'un shell fils peuvent être redirigées au moment de l'appel. Ceci redirige alors implicitement toutes les entrées et sorties standard des commandes appelées dans le shell fils.
- Une variable X du shell père ne sera connue dans le shell fils, qu'après lancement de la commande `export X` mais les modifications de la valeur de X effectuées dans le shell fils ne seront pas répercutées dans le shell père.

Les fichiers spéciaux : `/dev/null` doit être déclaré comme un fichier en mode caractère. La commande `ls -l /dev/null` doit renvoyer un résultat de la forme :

```
crw-rw-rw-   1  Admin None    000,000 Mar 30 10:15  null
```

Le répertoire contenant les binaires : `/bin` doit exister et contenir les commandes du Shell.

Divers : La notation `${parameter:-word}` permet d'attribuer la valeur "`word`" à "`parameter`" si celui-ci n'existe pas ou est égal à `NULL`.

2.3.2 Les fichiers Makefiles

Les Makefiles sont des fichiers décrivant les dépendances entre les différents fichiers qui composent un programme, ainsi que les différentes commandes nécessaires pour générer une version à jour de chacun de ces fichiers. Ainsi, il devient inutile de se souvenir de tous les modules qui ont été modifiés depuis la dernière compilation et de tout recompiler chaque fois qu'un module a changé. Un fichier Makefile est interprété par la commande "Make".

Exemple de fichier Makefile :

```
myprog: myprog.o
        cc -o myprog myprog.o
myprog.o: myprog.c myprog.h
        cc -c myprog.c
```

Dans cet exemple, les termes placés à gauche du caractère '`:`' (`myprog` et `myprog.o`) sont appelés les cibles (*target*). A droite, il s'agit des dépendances (`myprog.o` et `myprog.c myprog.h`) à partir desquelles la cible sera construite. L'ensemble de la cible et de ses dépendances forme la règle.

Le bloc de lignes qui suit directement la règle est un groupe de une ou plusieurs instructions qui s'exécutent lorsque la cible est atteinte.

2.3.3 L'outil Make-makefile

"Make-makefile" est l'outil standard utilisé pour automatiser la compilation et le portage des outils de CADP sur différentes architectures. Il a été mis au point par Hubert Garavel. A la date du 1er octobre 1998, tous les outils de CADP (ceux gérés par VASY et ceux gérés par VERIMAG) sont fabriqués avec Make-Makefile.

Le package Make-makefile se compose des fichiers suivants :

- `arch` : un script Shell qui détermine l'architecture de la machine sur laquelle il est exécuté.
- `Make-makefile` : un script Shell pour générer des Makefile.
- `make.Make-makefile` : un script Shell qui joue le rôle de "make".

Note : tous les scripts Shell sont écrits en Bourne Shell. Make-makefile prend en entrée le répertoire courant, un fichier "Userfile" contenant des paramètres spécifiques à l'exécutable à produire et le résultat de la commande "arch" ($ARCH). Il produit en sortie un fichier Makefile.$ARCH dépendant de l'architecture.

Pour exécuter ce Makefile.$ARCH, il suffit d'exécuter la commande "make.Make-makefile" qui lit un fichier de configuration de la commande "Make" (Usermake) et exécute les règles nécessaires pour produire l'exécutable défini dans le fichier Userfile.

Le synoptique de fonctionnement de "Make-makefile" est illustré sur la figure 2.3.

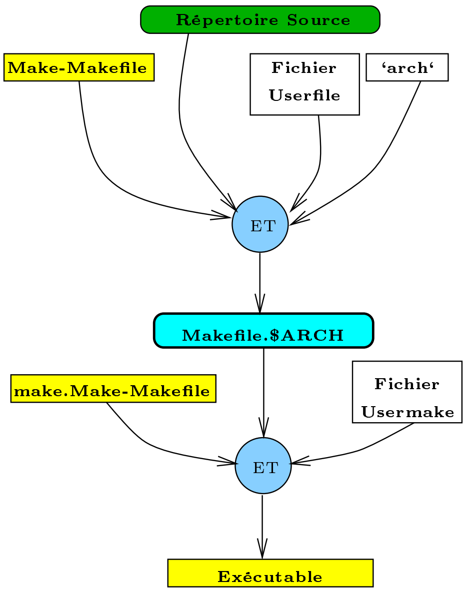

Figure 2.3: Architecture du Make-makefile

2.3.4 Les pages de manuel

La documentation de CADP est basée sur les pages de manuel UNIX (*manpages*). Ces pages, codées à la main au format NROFF [EP87], peuvent être ensuite formatées et consultées à l'écran ou à l'imprimante.

ROFF et ses descendants comme NROFF (destiné à l'affichage à l'écran) et TROFF (spécialisé dans l'impression de documents) sont des outils de formatage issus des laboratoires Bell d'AT&T dans les années 1970. Le langage de commande de ROFF servait à commander des machines de composition graphique. Les commandes sont donc de bas niveau, et la syntaxe peu amène.

2.4 Langages utilisés dans les outils CADP

2.4.1 Langage C

C est le langage le plus utilisé dans les sources de la boîte à outils CADP. Les programmes ont été écrits de la façon la plus portable possible, en se référant de la manière la plus fidèle à la norme ANSI.

Portabilité du C et norme ANSI

Un programme est portable s'il peut être transféré d'un système informatique à un autre. Le degré de portabilité est une mesure de la facilité avec laquelle un tel transfert peut être réalisé.

Pour les programmes écrits en langage C, il existe plusieurs dimensions de portabilité. Les principales barrières à la portabilité se situent à la frontière entre le code ANSI C [Sch93] et le "vieux" C [Rit84], d'une part, entre les environnements DOS et UNIX, d'autre part.

Avant la normalisation ANSI du langage C, les développeurs de compilateurs C étaient libres de faire ce qu'ils voulaient. Cependant, le langage implémenté par les compilateurs pré-ANSI est généralement un sous-ensemble strict de la spécification ANSI.

Le principal problème lié a cette hétérogénéité de compilateurs est le pré-processeur. La plupart des pré-processeurs pré-ANSI a un comportement différent de celui spécifié dans la norme ANSI, surtout sur les environnements basés sur le système d'exploitation UNIX.

La spécification ANSI implique une utilisation très rigoureuse des bibliothèques standard. Dans les environnements pré-ANSI, celles-ci n'étaient pas complètement standard : des différences pouvaient être rencontrées dans les fonctionnalités fournies et dans les fichiers `.h`, par exemple sur l'utilisation des commandes `printf()` et `scanf()`.

Bien sûr, pour une portabilité totale, il ne faut pas utiliser des bibliothèques de fonctions qui ne font pas partie du standard ANSI.

La norme ANSI ne spécifie pas totalement le langage C. Plusieurs secteurs sont laissés à la libre appréciation du compilateur afin qu'il réalise la tâche la plus appropriée. Cette liberté de la norme est placée sous la bannière du comportement de l'implémentation spécifique.

Le comportement d'implémentation spécifique le plus significatif concerne la taille et la nature des types de données de base. La norme ANSI présente une liste de plusieurs pages des comportements d'implémentation spécifique.

Les principaux sont les suivants :

- Dans la mesure du possible, déclarer les variables entières comme `short`, `int` ou `long int`. Ne pas reposer sur les valeurs par défaut.

- Prendre en compte les contraintes sur les identificateurs (voir tableau ci-dessous).

- Etudier avec soin les caractéristiques peu utilisées du langage telles que les énumérations et les champs de bits, qui sont vraisemblablement les plus problématiques dans un processus de portage.

- Attention au types de données union et aux effets de bords des contraintes d'alignement.

- Ne pas négliger les effets de bord possibles sur l'ordre d'évaluation des arguments d'une fonction.

- Concernant les types de données caractères, toujours préférer des variables de type `signed char` ou `unsigned char` au profit d'un simple `char`.

Par ailleurs, il est indispensable de prendre en compte les limites du compilateur si l'on souhaite avoir le code le plus portable possible. Le tableau suivant dresse la liste des limites à respecter (ces limites sont souvent dépassées par de nombreux compilateurs exceptés ceux qui respectent la spécification ANSI).

31	arguments par appel de fonction
32767	octets dans un objet
257	labels `case` dans un `switch`
509	caractères sur une ligne de source
509	caractères dans une constante chaîne
127	constantes dans une énumération
15	niveaux de `do`, `while` et `for`.
12	niveaux de référencement dans une déclaration
32	niveaux d'expressions dans une expression
511	identificateurs externes dans un module
8	niveaux de fichiers `.h`
127	identificateurs locaux dans un bloc
15	niveaux de `if` et `switch`
8	niveaux de `#if`, `#ifdef` et `#ifndef`
1024	macros dans un module
127	membres d'une union ou d'une structure
31	paramètres dans la définition d'une fonction
31	paramètres dans la définition d'une macro
6	caractères significatifs dans un identificateur externe
31	caractères significatifs dans un identificateur interne
15	niveaux de définition de structure et d'union

2.4.2 Tcl/Tk/Tix/Expect

L'interface homme-machine (IHM) de CADP est développée grâce au langage TCL et ses différentes extensions TK, TIX et EXPECT [Wel95] [Ous94].

Dans les années 90, la convivialité d'utilisation des applications a pris toute son importance. On ne peut plus écrire d'applications sans IHM graphique. Le confort d'exécution est augmenté par l'utilisation de la souris, la sélection de champs dans des listes qui évite bien des erreurs de frappe, l'utilisation de boutons, etc.

TCL/TK a été développé en réponse à ces préoccupations. Il a l'avantage d'être simplement un langage interprété et d'être gratuit. De surcroît, il bénéficie d'un forum de discussion très actif et d'une documentation croissante sur le réseau Internet.

TCL/TK est disponible pour toutes les plate-formes UNIX et le portage par l'auteur pour Macintosh et PC est disponible depuis peu.

La partie TK de cet ensemble a suscité beaucoup d'enthousiasme, étant donné la simplicité avec laquelle on peut construire la partie graphique d'une application.

Pourquoi TCL/TK ?

TCL et TK forment ensemble un système de programmation pour des applications avec interfaces graphiques (GUI : *graphical user interface*). TCL et TK ont été conçus et implémentés par John Ousterhout, de l'université de Berkeley (Californie).

TCL (*Tool Command Language*) est un langage de commandes au même titre que Bourne shell, C shell, Korn shell et Perl. Ce langage a l'avantage d'être extensible. Une bibliothèque contient l'ensemble des commandes, écrites en C, qui constitue le noyau de TCL ainsi que l'interpréteur de commandes.

TK (*Toolkit*) est l'extension de TCL pour la partie application graphique.

A l'aide de l'exécutable "wish" (*Windowing Shell*), il est possible d'écrire des applications graphiques sous la simple forme de scripts. Les avantages de cet ensemble TCL/TK peuvent être résumés comme suit :

- syntaxe simple et richesse des commandes de base,
- pas de compilation nécessaire (scripts interprétés),
- outils de haut niveau pour la partie graphique (de nombreux objets graphiques sont inclus dans TK),
- extensibilité : il est possible d'écrire de nouvelles commandes en C et de les ajouter à la bibliothèque,
- il existe déjà de nombreuses extensions disponibles sur le réseau.

TK contient un grand nombre de widgets (*Window gadgets*) destinés à enrichir l'interface graphique et lui donner toujours plus de convivialité.

TIX (*Tk Interface Extension*) est une extension de TCL/TK visant à fournir des "méga-widgets", c'est-à-dire des ensembles de widgets regroupés dans un seul objet manipulable comme n'importe quel simple widget. Il permet de gagner beaucoup de temps dans la réalisation d'interfaces graphiques en mettant à disposition des objets tels que des boîtes de dialogues pré-paramétrées, des navigateurs, etc.

EXPECT, enfin, est un utilitaire, développé en script TCL, destiné à contrôler les programmes interactifs. Par exemple, '`ftp`' ne peut être exécuté sans l'intervention d'un utilisateur. EXPECT permet d'écrire des scripts qui décrivent le dialogue avec l'utilisateur et le rendent plus convivial.

Eucalyptus

EUCALYPTUS est le nom de l'interface graphique (voir figure 2.4) de la boîte à outils CADP développée à l'aide de TCL et de l'extension TK.

Le principe de fonctionnement d'EUCALYPTUS réside dans un mécanisme de communication entre deux processus : un Bourne Shell et l'interpréteur WISH des langages TCL et TK. Cette communication est établie à l'aide de 2 tubes[9] qui vont permettre de mettre en relation la

[9] pipes

sortie standard de `wish` avec l'entrée standard du Shell et la sortie standard du Shell avec l'entrée standard de `wish`.

- La partie gauche de la fenêtre d'EUCALYPTUS (fenêtre des icônes) permet de composer des commandes à l'aide de l'interface graphique. Ce sont ces commandes qui sont ensuite envoyées vers le shell exécuté en parallèle.
- La partie droite de la fenêtre d'EUCALYPTUS (fenêtre des résultats) permet d'afficher les résultats des commandes préalablement préparées.

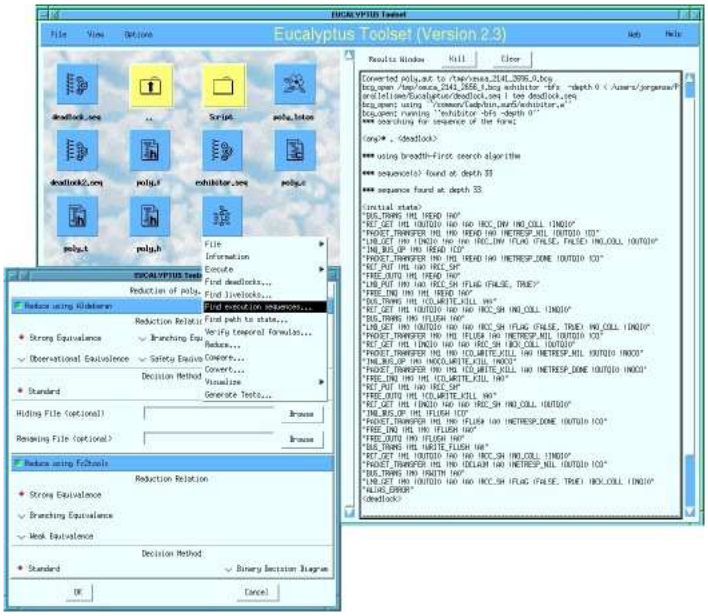

Figure 2.4: Interface graphique EUCALYPTUS

Ce mécanisme est réalisé en C par le programme "`duplex`" qui réalise les opérations suivantes :

- création des 2 tubes à l'aide de la commande `pipe()`,
- duplication du processus en cours grâce à la commande `fork()`,
- redirection des extrémités des tubes pour croiser l'entrée standard (`stdin`) et la sortie standard (`stdout`) du processus père et du processus fils (commandes `close()` puis `dup()`),
- lancement de `wish` par le père,
- lancement du shell par le fils.

Ce mécanisme est illustré sur la figure 2.5.

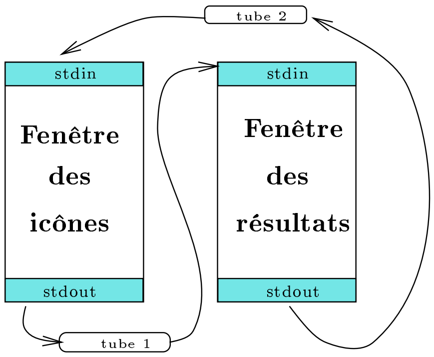

Figure 2.5: Principe de la communication par tube dans EUCALYPTUS

Installator

INSTALLATOR (figure 2.6) est l'assistant d'installation de CADP. Depuis septembre 1997, CADP est distribué de façon électronique par Internet en utilisant les protocoles FTP [PR85] et HTTP [RFBL99] ; le support magnétique n'est, depuis ce jour, plus supporté.

INSTALLATOR permet de simplifier l'installation de CADP parce qu'il réalise automatiquement la connexion sur le site FTP du projet VASY, vérifie la dernière version en vigueur et télécharge CADP pour l'architecture désirée.

Figure 2.6: INSTALLATOR : l'assistant d'installation de CADP

INSTALLATOR est, comme EUCALYPTUS, développé en TCL et TK. Il utilise, en plus, l'extension EXPECT afin de communiquer avec le protocole FTP. Cet échange est réalisé de la façon suivante :

```
spawn ftp ftp.inrialpes.fr

expect {*Name *}
exp_send "anonymous\n"

expect {*Password:*}
exp_send "[exec logname]@\n"

expect {*ftp>*}
exp_send "cd /pub/vasy/cadp\n"
expect {*ftp>*}
exp_send "get $REMOTE_FILE $LOCAL_FILE\n"
expect {
    "*No*file*ftp>" {
        return 1
    }
    "*ermission*denied" {
        return 2
    }
    "*ftp>*" {
        return 0
    }
}

exp_send "quit\n"
```

Dans cette portion de code TCL utilisant EXPECT :

- `spawn` permet d'exécuter un programme interactif (la connexion avec le serveur FTP),
- chaque commande `expect` permet de vérifier la présence d'une chaîne de caractères dans la sortie du processus courant (programme interactif),
- chaque commande `exp_send` envoie ses arguments sur l'entrée standard du processus courant. Les arguments en question sont des commandes destinées au programme lancé par la commande `spawn` (ici : `ftp`).

2.4.3 Lex et Yacc

Lex

LEX est un générateur d'analyseurs lexicaux qui exploite un code source contenant des instructions en langage C complété par un ensemble de règles de reconnaissance lexicale pour produire un programme C [Mic88].

Un programme LEX est composé de trois parties :

- La première (délimitée par les chaînes de caractères `%{` et `}%` en première colonne) contient les déclarations C communes aux actions LEX et au programme principal.
- La seconde (encadrée par des marques `%%` en première colonne) contient les règles LEX auxquelles sont associées des instructions C.
- La dernière contient le programme principal (`main`).

Pour exemple, le programme suivant remplace dans un texte donné en entrée, tous les caractères “<” par des retours à la ligne :

```
%{
#include "stdio.h"
%}
%%
"<"     printf("\n");
%%
main()
{
        yylex();
}
```

Le programme principal contient l'appel de la fonction `yylex()` qui, à l'exécution, lancera l'analyse du flot de données arrivant sur l'entrée standard (`stdin`).

Yacc

YACC (*Yet Another Compiler Compiler*) [Mic88] est un programme destiné à compiler une grammaire du type LALR (*Look Ahead Left Root*) [ASU86] et à produire le texte source d'un analyseur syntaxique du langage engendré par cette grammaire. Il est aussi possible, en plus de la vérification de la syntaxe de la grammaire, de faire effectuer à l'analyseur des actions sémantiques. De la même manière que pour un fichier LEX, un fichier YACC se compose de trois parties articulées comme suit :

```
declarations
%%
productions
%%
code additionnel
```

Seuls le premier séparateur `%%` et la deuxième partie (`productions`) sont obligatoires.

2.5 Différences entre WINDOWS NT et UNIX : problèmes potentiels

Cette section illustre les diffèrences entre les deux systèmes UNIX et WINDOWS NT sous divers angles :

- sous l'angle du système d'exploitation : pour les problèmes généaux d'administration,
- sous l'angle de la bibliothèque C : pour les aspects liés à la programmation,
- sous l'angle des commandes utilisateurs : pour la manipulation de commandes diverses.

2.5.1 Système d'exploitation

WINDOWS NT est un système d'exploitation fonctionnant sur des processeurs 32 bits. Il possède toutes les caractéristiques UNIX standard telles que la gestion des réseaux et de l'interface graphique. Les deux systèmes utilisent de nombreux concepts communs, mais il existe certaines différences que les développeurs doivent prendre en compte pour migrer leurs applications d'UNIX vers WINDOWS NT.

Les principales différences sont les suivantes [Cor96] :

(voir `http://www.performancecomputing.com/unixintegration/9710/9710f1.htm`)

- Les deux systèmes d'exploitation ont des API complètement différentes. Dans certains cas, il n'existe pas de traduction simple du code UNIX vers WINDOWS NT. Certains programmes sont simples à porter, mais il existe des situations où le code source nécessite une restructuration significative.
- Il y a des différences subtiles dans la relation entre les processus ou dans le traitement des signaux.
- Le concept UNIX de la relation père-fils entre processus n'existe pas sous WINDOWS NT, ce qui nécessite que chaque processus garde une trace de l'identité de chaque processus qu'il a lancé (PID ou handle).
- WINDOWS NT utilise un modèle de gestion des processus différent de celui d'UNIX, et, de ce fait, ne dispose d'aucun équivalent à la commande `fork()`. La séquence `fork()/exec()` fréquemment utilisée dans les programmes UNIX pour créer un nouveau processus exécutant un nouveau programme peut être implémentée sous WINDOWS NT par la fonction `CreateProcess()`.
- Contrairement à UNIX, WINDOWS NT autorise le programmeur à créer différentes piles et gère la mémoire de façon indépendante dans chacune des piles. Cela simplifie la programmation et rend les applications plus performantes.

- Windows NT fournit le support au format Unicode, un ensemble de caractères codés sur 16 bits qui permet l'utilisation de caractères internationaux. Unix utilise pour sa part l'ensemble standard de caractères 8 bits Ansi et le format XPG4 spécifiant un ensemble de caractères étendus.

- Unix ne dispose pas d'équivalent à la gestion OLE (*Object Linking and Embedding*) proposée par Windows NT.

- Sensibilité de casse alphabétique : l'Api Win32 ne fournit aucun support de sensibilité de casse. Cela pose un problème puisque les fichiers suivants, par exemple, ne pourront pas coexister dans le même répertoire sous Windows NT : `makefile`, `Makefile`, `MAKEFILE`. Dans le même registre, les variables d'environnement `$path` et `$PATH` ne représentent qu'une seule variable, sous Windows NT.

- Windows NT ne sait pas gérer les liens symboliques.

- Concernant le nommage des fichiers, Windows NT, contrairement à Unix, autorise les espaces, mais interdit certains caractères réservés tels que ":".

- Sous Windows NT, il n'est pas possible de nommer un fichier "com1", "lpt1" ou "aux" sous peine d'arrêt du processus courant.

- Sur Windows NT, les options de la ligne de commande sont précédées du caractère slash (/) alors que Unix utilise le tiret (-).

- L'utilisateur doit nommer les exécutables Windows NT avec une extension ".exe" car Windows NT ne peut reconnaître un exécutable sans extension.

- Les bibliothèque partagées Unix se nomment DLL sous Windows NT.

- Les systèmes Unix varient très largement et présentent des Api très différentes selon les constructeurs. Le portage des applications entre différents systèmes Unix (ou même entre différentes versions d'un même système) est souvent coûteux.

  Le système Windows NT présente la même Api quelque soit le processeur ou l'architecture machine et les codes sources fonctionnent sous n'importe quel Windows NT.

2.5.2 Bibliothèque C

Dans de nombreux cas, il y a peu, voire pas, de différences entre l'implémentation Unix de la bibliothèque C et sa contrepartie sous Windows NT. Certaines fonctions de la bibliothèque C Windows NT sont précédées d'un caractère "_" (souligné). Le tableau suivant montre les fonctions nécessitant d'être remplacées dans le cadre d'un portage vers Windows NT.

Fonction de la bibliothèque C UNIX	Equivalent WINDOWS NT
strdup()	_strdup()
strcasecmp()	_strcmp()
strcmp()	_strcmp()
isascii()	_isascii()

D'autres différences viennent s'ajouter à ce tableau :

Fichiers textes/binaires : WINDOWS NT utilise la convention DOS de 2 caractères `<cr><nl>` pour signifier la fin de chaque ligne dans un fichier texte. UNIX, lui, utilise uniquement le caractère `<nl>` pour la même opération.

Il en résulte que le traitement de fichiers est plus complexe que sous UNIX car il existe deux modes séparés de gestion de fichiers :

- le mode binaire
- le mode texte

Le mode binaire traite un fichier comme une séquence d'octets. Le mode texte supprime un `<cr>` en face de chaque `<nl>` en lecture et insère un `<cr>` en face de chaque `<nl>` en écriture. Parce que le nombre de caractères lus n'indique pas la position physique dans le fichier, les programmes qui lisent des ensembles de caractères et utilisent la commande `lseek()` (par exemple) ont de grandes chances de ne pas fonctionner.

Fonctions non disponibles : Les fonctions suivantes ne sont pas disponibles dans la librarie C de WINDOWS NT :

- `bcmp()`
- `bzero()`
- `bcopy()`
- `index()`
- `rindex()`

2.5.3 Commandes utilisateur

Ftp : `ftp` est l'interface utilisateur du protocole Internet FTP (*File Transfer Protocol*). Cette commande permet de transférer des fichiers entre machines distantes. "`ftp`" est un programme interactif qui requiert des commandes utilisateurs aux différents prompts affichés et renvoie les résultats de ces commandes. WINDOWS NT fournit une commande `ftp` dont le fonctionnement est très proche de celui de son équivalent sous UNIX.

Une fois exécutée la commande `ftp` suivie du nom de machine distante, il suffit de s'identifier (nom d'utilisateur et mot de passe). Cette identification se fait :

- sous UNIX : au prompt "`Name:`"

– sous WINDOWS NT : au prompt “`User:`”

Cette différence de prompt pose un problème potentiel de portage, notamment lorsqu'on utilise un utilitaire tel que EXPECT (voir section 4.4).

Make : `make` est la commande qui permet de mettre à jour et regénérer des programmes et des fichiers à partir de règles associées à des cibles (voir section 2.3.2). La syntaxe des fichiers Makefiles varie sensiblement selon l'architecture sur laquelle ils sont exécutés. Ils nécessitent donc les modifications suivantes [Cor96] :

– Les macros qui définissent des chemins de répertoires doivent prendre en compte la différence de séparateur : “/” pour UNIX et “\” pour WIN32.

– Le nom et l'emplacement des fichiers `.h` peuvent varier d'une architecture à une autre.

– Certaines macros définissant des commandes dépendantes de l'architecture doivent être modifiées. Exemple : `$(CC)`.

– Certaines options de compilation varient selon l'architecture. Ainsi, on utilisera l'option `-c` sous UNIX et `/nolink` sous WIN32 pour ne pas réaliser l'étape d'édition de liens.

– Les extensions des binaires produits sont différentes. Les objets sont suffixés de l'extension `.o` sous UNIX et `.OBJ` sous WIN32. Les exécutables WIN32 doivent être suffixés `.EXE` sous WIN32 alors qu'une telle contrainte est inexistante sous UNIX.

Rcp : RCP est l'utilitaire qui permet de copier des fichiers entre des machines distantes. Sa syntaxe est la suivante :

```
rcp [-r] hostname1:filename1 hostname2:filename2
```

RCP est disponible et son fonctionnement est très proche sur les deux environnements UNIX et WINDOWS NT. La version WINDOWS NT de RCP propose quelques options supplémentaires fort intéressantes :

– `-a` : cette option spécifie le mode de transfert ASCII. Ce mode convertit les séquences “`\r\n`” en “`\n`” pour les fichiers sortants, et le caractère “`\n`” en séquence “`\r\n`” pour les fichiers entrants.

– `-b` : cette option spécifie le mode de transfert BINARY. Aucune conversion de fin de ligne n'est effectuée dans ce mode.

Ces options sont à prendre en compte dans le présent processus de portage afin d'éviter les problèmes dans les scripts qui utilisent la commande RCP.

Rsh : RSH est le programme qui permet d'exécuter une commande sur une machine distante. Sa syntaxe est la suivante :

```
rsh  -l username hostname command
```

Il suffit simplement de préciser le nom de l'utilisateur (`username`) qui réalise la demande de connexion et le nom de la machine (`hostname`) sur laquelle sera exécutée la commande.

La différence de fonctionnement entre UNIX et WINDOWS NT réside dans le paramétrage du service sur le serveur RSH.

- sous UNIX : il est nécessaire de configurer et lancer un processus `inetd` qui "écoute" les ports TCP/IP associés au services présents dans le fichier de configuration "`/etc/inetd.conf`". Seuls les utilisateurs authentifiés dans le fichier "`/etc/.rhosts`" pourront exécuter des commandes sur le serveur distant.
- sous WINDOWS NT : l'installation du service RSH se fait par la commande `rshsetup`. Comme pour UNIX, le fichier `.rhosts` (positionné dans le répertoire `%Systemroot% \System32\Drivers\etc`) assure la sécurité locale des connexions [Gro93].

  La commande requise est soumise à l'interpréteur WIN32 `cmd.exe` et non à un shell comme pour UNIX. Cela signifie que, dans le cadre du portage, il sera nécessaire de faire précéder la commande envoyée vers un serveur WIN32 par la séquence "`sh -c`".

Après avoir établi les différences essentielles existant entre les environnements UNIX et WINDOWS NT, voyons, dans le chapitre qui suit, quelles sont les solutions qui permettent de les prendre en compte et comment se sont orientés nos choix.

Chapitre 3

Solutions et choix

Ce chapitre présente les solutions de portage possibles à travers les différentes stratégies envisageables et les différents logiciels du commerce permettant de réaliser la migration d'applications UNIX vers WINDOWS NT. Pour chacune des solutions éventuelles, nous réaliserons une évaluation qui permettra d'aboutir au choix de la solution répondant au mieux aux exigences du portage des programmes de la boîte à outils CADP.

3.1 Stratégies possibles

3.1.1 Simple recompilation

Dans la plupart des cas, le portage d'applications UNIX vers WINDOWS NT ne nécessite qu'une simple recompilation. La bibliothèque C est similaire à celle que l'on peut trouver dans le SYSTEM V. Ainsi, si le code ne comporte pas une des caractéristiques suivantes, il peut fonctionner immédiatement :

- utilisation d'appels système absents de la bibliothèque C,
- utilisation de quelques rares paradigmes non implémentés sous WINDOWS NT, tels que `fork()`,
- dépendance forte avec l'architecture (ordre des octets ou compression de données),
- utilisation d'une interface graphique.

Les applications écrites en ANSI C et n'utilisant pas d'appels systèmes UNIX se compilent aisément et fonctionnent généralement. Pour de nombreuses personnes qui désirent porter leurs applications, le premier pas consiste à voir leur programme fonctionner. La volonté d'optimiser et inclure les caractéristiques avancées de WINDOWS NT ne vient que dans un second temps.

C'est pourquoi, une des premières stratégies à explorer consiste à se procurer un compilateur WIN32 (voir section 3.2), compiler les applications et voir si tout fonctionne ...

3.1.2 Couche POSIX fournie avec WINDOWS NT

WINDOWS NT inclut un sous-système POSIX dans son architecture.

POSIX et WINDOWS NT : POSIX signifie Portable Operating System Interface. POSIX a démarré comme un effort de la part de la communauté IEEE de promouvoir la portabilité des applications à travers les différents environnements UNIX.

WINDOWS NT est conforme à la norme POSIX.1 qui assure la conformité de l'API système à travers le langage C.

POSIX et le système de fichiers : Le sous-système POSIX propose un certain nombre de fonctionnalités conformes à la norme telles que la possibilité d'administrer des liens symboliques ou des noms sensibles à la casse alphabétique.

Communication avec les autres sous-systèmes : Le sous-système POSIX ne peut communiquer avec les autres sous-systèmes que par l'intermédiaire de tubes qui permettent la redirection de la sortie standard d'une commande POSIX vers l'entrée standard d'une commande WIN32, ou l'inverse. Cela autorise, par exemple, la commande "`ls | more`". Cependant, il n'existe aucune autre solution pour accéder à des fonctionnalités extérieures au sous-système, telles que la gestion graphique, la gestion réseau, l'échange dynamique de données ou les fichiers mappés en mémoire, qui sont des fonctionnalités avancées du sous-système WIN32.

Fabrication d'un exécutable POSIX : Pour compiler et lier une application POSIX sur WINDOWS NT, il faut les fichiers `.h` et les bibliothèques POSIX livrées avec le système WINDOWS NT[10]. Les exécutables POSIX sont liés dynamiquement (lors de l'exécution) avec la DLL `PSXDLL.DLL`.

WINDOWS NT disposant d'une couche POSIX, une des premières réflexions menait naturellement à son utilisation puisqu'elle met en œuvre la plupart des appels systèmes standard UNIX. Pour ces raisons et parce qu'elle évitait de lourds investissements, nous avons d'abord pensé que la couche POSIX suffisait à elle-seule pour porter les outils CADP vers WINDOWS NT.

3.1.3 Autres bibliothèques POSIX du marché

Une autre méthode de migration consiste à utiliser une bibliothèque POSIX qui permet "d'émuler" l'environnement UNIX sur une machine pourvue du système WINDOWS NT. Cette bibliothèque aura pour objectif principal d'apporter une solution à chacun des appels UNIX manquants sur WINDOWS NT. Il existe différentes solutions commerciales, plus ou moins complètes, basées sur ce concept. Pour certaines, la solution est présentée sous forme d'une bibliothèque dynamique (DLL) fonctionnant dans le sous-système WIN32 et avec laquelle il faut se lier afin que le système puisse charger la routine nécessaire au moment de son exécution.

[10] *NT Resource Kit*

Pour d'autres, il s'agit d'un sous-système (un exécutable) qui vient, purement et simplement, en remplacement du sous-système POSIX livré par MICROSOFT.

Dans tous les cas, ces méthodes sont des solutions attrayantes car elles présentent l'avantage de minimiser l'effort de portage. Le principal inconvénient des palliatifs POSIX est que l'on rajoute une "couche" au système, ce qui a pour effet de rendre l'application moins performante.

La section 3.3 est entièrement consacrée à l'étude de ces solutions.

3.1.4 Compilation croisée

La compilation croisée (*cross-compilation*) consiste à générer, par compilation du code source sur la machine hôte (UNIX), des exécutables dans le format de la machine cible (WINDOWS NT).

Un environnement de compilation croisée comprend :

- un compilateur permettant de définir l'architecture cible WIN32,
- des fichiers `.h` de l'architecture cible WIN32,
- des bibliothèques de l'architecture cible WIN32.

Actuellement, deux compilateurs permettent la compilation croisée : `gcc` et `egcs`. Ces compilateurs peuvent s'associer avec l'une des deux solutions WIN32 (fichiers `.h` et bibliothèques correspondantes) suivantes :

- MINGW32
- CYGWIN

3.1.5 Réécriture au format WIN32

Cette option de portage consiste à réécrire les applications directement dans l'API WIN32 et mettre à profit toutes les caractéristiques avancées de WINDOWS NT, telles que l'interface graphique, l'échange dynamique de données et les fichiers mappés en mémoire.

3.2 Environnements UNIX de compilation WIN32

gcc : `gcc` est un compilateur C livré sous la GPL[11] qui requiert que tous les programmes dérivés soient distribués avec le code source. Il est issu du projet de la FSF (Free Software Foundation) dont l'objectif est d'offrir, à tous, les outils les plus performants. `gcc` appartient donc aux meilleurs outils de son domaine. Il permet de générer du code exécutable pour bon nombre de plate-formes dont les environnements WIN32.

[11] General Public License

egcs : egcs est une étape expérimentale visant à favoriser le développement du compilateur gcc. Il est basé sur la version 2.8 du compilateur gcc et ne cesse d'évoluer (contrairement à gcc). Placé sous la tutelle de la FSF, il améliore gcc en lui ajoutant de nouvelles optimisations, de nouvelles machines cibles, de nouvelles bibliothèques. egcs inclut également un certain nombre de corrections de bogues présents dans gcc-2.8.

djgpp : djgpp est un portage du compilateur et des outils de développement GNU[12] C/C++ pour des processeurs 32 bits fonctionnant en mode protégé sous Ms-Dos. Il fonctionne également sous Windows mais ne produit que des exécutables Ms-Dos.

cl : cl est le compilateur C de Microsoft. Il dispose de nombreuses bibliothèques mettant en œuvre toutes les fonctionnalités avancées de Windows NT.

bcc : bcc est le compilateur C de Borland. Comme cl, il est un compilateur C natif Win32 largement utilisé dans la communauté Windows.

3.3 Environnements UNIX fonctionnant sous WINDOWS

Cette section est destinée à présenter les différentes solutions "d'émulation" Unix sous Windows NT.

3.3.1 Cygwin

http://sourceware.cygnus.com/cygwin/

Cygwin est l'environnement Unix GNU porté sous Windows NT par la société Cygnus. Complet, il permet de développer des applications Win32 en mode console ou graphique (GUI) mettant à profit l'Api standard de Microsoft. Il est composé de :

- nombreuses bibliothèques et fichiers .h,
- une DLL d'émulation : cygwin.dll,
- un interpréteur Shell : bash (Bourne Shell amélioré),
- nombreuses commandes Shell,
- un compilateur C : gcc,
- un environnement de production : (gmake, flex, bison).

L'ensemble de la solution Cygwin est basé sur le sous-système Win32.

Cygwin est un logiciel libre composé d'outils GNU, d'outils du domaine public ou développés par Cygnus et placés sous la GPL.

[12] GNU is not Unix, Cf http://www.gnu.org

3.3.2 Mingw32

`http://agnes.dida.physik.uni-essen.de/~janjaap/mingw32/index.html`

MINGW32 est un ensemble "free" de fichiers `.h` et de bibliothèques qui permettent à un compilateur GNU de lier des programmes avec l'une des bibliothèques dynamiques C livrées par MICROSOFT. Par défaut, il utilise `CRTDLL.DLL` (*C-RunTime Library*) présente sur tous les systèmes WIN32 et n'a recours à aucune autre DLL. Cela signifie que les exécutables sont plus petits, indépendants et raisonnablement plus rapides. Il s'agit d'une bonne option pour les programmeurs qui souhaitent générer des exécutables natifs WIN32. Toutefois, MINGW32 signifie *Minimalist GNU win32*, ce qui indique que tous les appels de la bibliothèque C standard ne sont pas implémentés.

3.3.3 Uwin

`http://www.research.att.com/sw/tools/uwin`

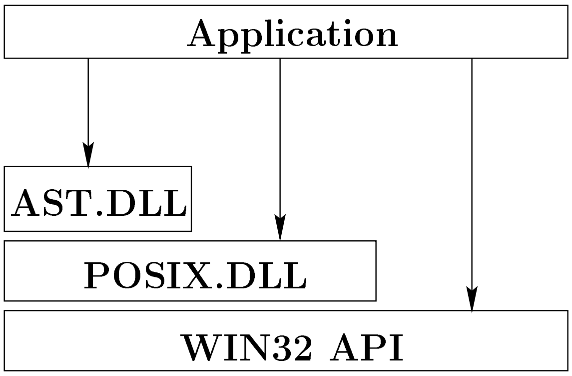

Figure 3.1: Architecture de Uwin

Créé par les laboratoires AT&T, UWIN est un ensemble de commandes, bibliothèques et fichiers `.h` qui fournissent un environnement UNIX pour WINDOWS NT. Développé par David Korn, il inclut notamment un Korn Shell et implémente de nombreuses fonctionnalités UNIX telles que l'appel `fork()`. UWIN fonctionne à l'aide de deux DLL basées sur l'utilisation du sous-système WIN32, comme le montre la figure 3.1.

3.3.4 Interix

`http://www.interix.com`

INTERIX est un sous-système (au sens WINDOWS du terme), créé par la société Softway Systems, Inc, qui vient en remplacement de la couche POSIX (livrée en standard). Le sous-système INTERIX contient de nombreuses améliorations au sous-système POSIX, notamment des langages de script (`ksh` et `csh`) et de nombreuses commandes associées, la gestion des sockets, la communication inter-processus et une implémentation `telnet` et `rlogin`.

INTERIX est conforme à la norme POSIX 1003.2.

3.3.5 Nutcracker

http://www.datafocus.com/products/nutc/

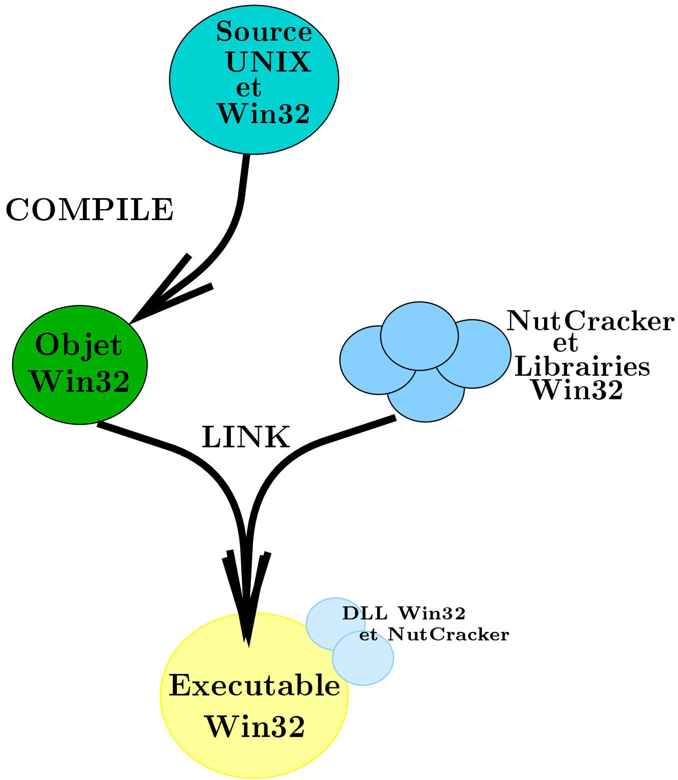

Figure 3.2: Processus de portage sous NUTCRACKER

NUTCRACKER (Datafocus) est un environnement de développement et d'exécution d'applications UNIX sous WINDOWS. Il inclut de nombreuses commandes Shell et utilitaires. La démarche de portage proposée est la suivante :

- transfert du code source sous WINDOWS,
- examen des Makefiles, des fichiers `.h` et du code source pour les problèmes de portage,
- compilation et édition de liens,
- correction d'erreurs de programmation,
- intégration avec WINDOWS,
- distribution de l'application.

Cette démarche est illustrée par le schéma 3.2.

3.3.6 Mks Toolkit

http://www.mks.com

Mks Toolkit consiste en une suite de plus de 220 utilitaires de programmation Unix et Windows pour l'environnement PC (Ms-Dos, Windows 3.1, Windows 95 et Windows NT) et pour OS/2.

Ce sont des utilitaires écrits en natif Api Win32 et, à ce titre, ils sont entièrement indépendants du sous-système Posix livré avec Windows NT. Outre des outils standards Unix (`ksh, awk, grep` ...), MKS fournit de nombreux utilitaires mettant en œuvre les caractéristiques avancées de l'Api Win32 telles que les liens DDE[13] ou le GUI[14].

3.4 Synoptique des solutions

Au titre d'une synthèse, la figure 3.3 présente la liste des différentes solutions de portage sous forme d'un synoptique décrivant la démarche de migration des applications Unix vers Windows NT.

Note : MKS Toolkit ne figure pas sur le synoptique car il ne constitue pas une solution de portage des applications écrites en C.

3.5 Evaluation des différentes solutions

Dans cette section, nous évaluons les solutions selon deux aspects :

- la stratégie globale à adopter, d'une part,
- l'environnement Unix fonctionnant sous Windows, d'autre part.

3.5.1 Stratégie à adopter

Simple recompilation

La boîte à outils Cadp est composée de nombreux éléments qui rendent une simple recompilation bien insuffisante pour obtenir un résultat significatif.

Cela est illustré par le fait que les programmes de la boîte à outils Cadp utilisent des appels systèmes tels que `fork()`, et, parce qu'ils sont inclus dans une interface graphique très volumineuse écrite en Tcl/Tk dont il faut tenir compte dans le processus de portage.

[13]Dynamic Data Exchange : Echange dynamique de données
[14]Graphical User Interface : Interface graphique utilisateur

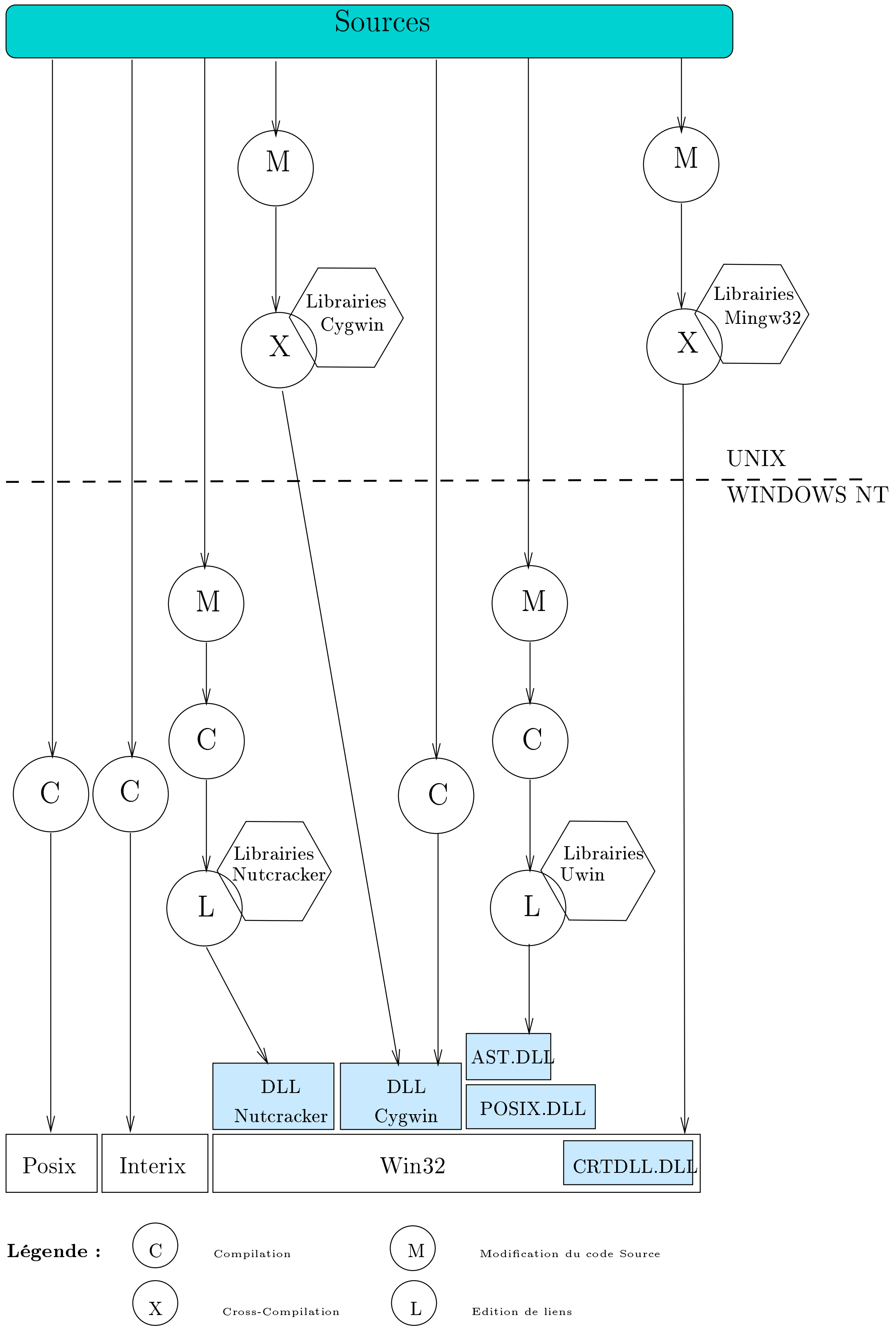

Figure 3.3: Synoptique des solutions possibles

Sous-système POSIX

Le sous-système Posix n'est pas une solution très viable dans le cadre du présent projet de migration car Microsoft a réalisé un sous-système Posix aussi fermé que possible. Il n'y a aucune solution pour accéder à des fonctionnalités extérieures au sous-système. Ainsi, il est impossible d'appeler le compilateur C Microsoft depuis le sous-système Posix. Cela signifie, entre autres, qu'un programme exécutable Posix ne peut créer une fenêtre ou utiliser des appels de procédures distantes (Rpc[15]) ou des sockets. Donc, il ne peut fonctionner qu'en mode console.

Par ailleurs, de par sa conformité à la norme POSIX.1, il ne permet l'appel qu'à seulement 110 fonctions de la bibliothèque C standard (composée d'environ 450 appels[16]).

De plus, il ne propose que les commandes Shell suivantes : `cat, chmod, chown, cp, find, grep, ln, ls, mkdir, mv, rm , rmdir, sh, touch, vi` et `wc`. Ce qui représente une couverture des besoins Cadp de l'ordre de 38%. Cela le rend bien incomplet pour satisfaire les attentes des outils de Cadp.

Enfin, du fait de sa capacité à gérer des liens symboliques ou la sensibilité de casse alphabétique, il nécessite de fonctionner sur un système de fichiers Ntfs, seul à pouvoir gérer ces caractéristiques (contrairement au système de fichiers de type Fat[17] ou HPFS[18]).

En définitive, le sous-système Posix a été implémenté par Microsoft en grande partie pour satisfaire les contrats du gouvernement américain pour lequel Posix est une requête très forte.

Compilation croisée

La solution de compilation croisée semble très attrayante car elle offre un avantage énorme : la standardisation des procédures de fabrication des binaires de Cadp à destination de toutes les architectures supportées et ce, à partir de la même architecture hôte. En clair, on lance un script Shell qui produit à partir du même code source, les exécutables Cadp pour : Sun Os 4.x, Sun Os 5.x, Linux et Windows NT. Cela signifie que l'on peut produire les programmes Cadp pour Windows sans avoir de PC Windows dans son parc.

La compilation croisée à destination de Windows ne peut être réalisée qu'à l'aide des bibliothèques Cygwin et Mingw32 (se reporter à la section 3.5.2 pour une étude détaillée de ces solutions).

Réécriture de l'application

Compte-tenu de l'interface graphique Eucalyptus de la boîte à outils Cadp et du volume des programmes à porter, il semble que l'effort de redéveloppement serait trop lourd et

[15]Remote Procedure Call
[16]Ce nombre est variable selon les systèmes : System V ou Bsd
[17]File Allocation Table : table pour fichiers au format MS-DOS
[18]pour gérer les fichiers dans le sous-système OS/2

pénalisant pour les utilisateurs s'il fallait tout réécrire directement dans le format WIN32. Bien que cela ne soit pas à exclure dans les années à venir et notamment parce que certains langages, tels que JAVA, permettent de réaliser des IHM de façon efficace, l'effort de portage doit rester minimum.

3.5.2 Environnements UNIX fonctionnant sous WINDOWS

Cygwin

Le Shell : CYGWIN est livré avec le shell `bash` qui répond aux propriétés attendues par CADP (voir section 2.3.1) à l'exception de la notion de Shebang. Parmi les commandes Shell proposées par CYGWIN et nécessaires au fonctionnement de CADP, il manque `more`, `which`, `file`, `mail`, `man`.

CYGWIN propose un mécanisme de gestion des liens symboliques par un en-tête magique contenant le mot-clé `symlink`.

L'environnement de programmation : CYGWIN propose un compilateur `gcc` ainsi que les générateurs d'analyseurs lexicaux et syntaxiques FLEX (pour LEX) et BYACC (pour YACC). Il est également pourvu de l'outil `make` pour exécuter les Makefiles.

Les bibliothèques du C : Les bibliothèques C proposées par CYGWIN sont très complètes et permettent d'apporter une solution à quasiment tous les problèmes potentiels de portage tels que la gestion des fichiers binaires/textes, la commande `fork()`, la gestion des signaux ou les fonctions réseau.

Utilisée dans le processus de compilation croisée, la solution CYGWIN présente l'inconvénient de rendre les exécutables dépendants de la DLL `cygwin.dll`, ce qui impose de souscrire une licence GNUPro auprès de Cygnus (coût 7495 dollars jusqu'à 5 utilisateurs et 24995 dollars jusqu'à 25 utilisateurs).

Divers : CYGWIN ne propose aucune possibilité de consulter des pages au format NROFF (pas de commande `man` notamment).

Mingw32

MINGW32 "n'est qu'un" ensemble de bibliothèques et de fichiers `.h` et ne dispose d'aucun élément de Shell (interpréteur ou commandes). A ce titre, il ne peut être utilisé qu'associé au compilateur GNU `gcc` (ou `egcs`) pour de la compilation native sous WINDOWS NT ou croisée depuis une machine UNIX. Il contient les fichiers `.h` suivants :

`alloc.h`	`errno.h`	`prsht.h`	`sys`	`winerror.h`
`assert.h`	`excpt.h`	`rcs`	`tchar.h`	`wingdi.h`
`cderr.h`	`fcntl.h`	`setjmp.h`	`time.h`	`winkernel.h`
`commctrl.h`	`float.h`	`share.h`	`unistd.h`	`winnt.h`
`commdlg.h`	`io.h`	`shellapi.h`	`values.h`	`winreg.h`

```
conio.h     limits.h    shlobj.h    varargs.h    winsock.h
ctype.h     locale.h    signal.h    wchar.h      wintypes.h
ddeml.h     malloc.h    sqlext.h    winadvapi.h  winuser.h
dir.h       math.h      stdarg.h    winbase.h    winversion.h
direct.h    mem.h       stddef.h    wincon.h
dirent.h    memory.h    stdio.h     windef.h
dlgs.h      mmsystem.h  stdlib.h    windows.h
dos.h       process.h   string.h    windowsx.h
```

Il présente une gestion minimale des signaux. Seuls les signaux suivants sont implémentés :

```
#define SIGINT        2     /* Interactive attention */
#define SIGILL        4     /* Illegal instruction */
#define SIGFPE        8     /* Floating point error */
#define SIGSEGV       11    /* Segmentation violation */
#define SIGTERM       15    /* Termination request */
#define SIGBREAK      21    /* Control-break */
#define SIGABRT       22    /* Abnormal termination (abort) */
```

La commande `kill()` n'existe pas, mais la commande `raise()` est implémentée.

La bibliothèque `libm.a` (bibliothèque mathématique) n'existe pas, mais les fonctions mathématiques sont tout de même implémentées dans la bibliothèque C générale `libmingw32.a`.

Par ailleurs, un problème de substitution de variable est à déplorer (contribution auprès de Colin Peters[19]). En effet, la commande (par exemple) :

```
system("$VAR/mycmd")
```

ne réalise pas la substitution de la variable `VAR`. Cela implique de réécrire la commande ci-dessus de la façon suivante:

```
var = getenv(VAR) ;
sprintf(command,"%s/mycmd",var) ;
system(command) ;
```

En revanche, Mingw32 met en œuvre des modes de gestion de fichiers textes et binaires avancés et propose des fonctionnalités purement Windows telles que l'accès à la base de registres.

Mingw32 est gratuit.

[19]concepteur de la solution Mingw32

Uwin

UWIN présente deux versions complémentaires :

- UWIN BASE : composé du KornShell (`ksh`) et de 115 commandes Shell.
- UWIN SDK : contenant un environnement de développement UNIX : `cc, cpp, gmake, ld, lex, yacc ....`

Le KornShell est un sur-ensemble du Bourne Shell, ce qui lui donne toutes les caractéristiques attendues par CADP.
UWIN :

- est pourvu d'une commande `man` qui réalise une conversion d'un fichier au format NROFF vers le format HTML puis lance un navigateur Internet pour afficher le résultat,
- sait gérer les liens symboliques comme sous UNIX (`ln -s`). Même les raccourcis créés sous WINDOWS sont reconnus comme des liens symboliques sous UWIN,
- supporte les fichiers spéciaux tels que `/dev/null`,
- propose une solution pour accéder à la base de registres WINDOWS en montant un système de fichiers `/reg` dont l'arborescence reflète la structure des clés et des valeurs de registre.

La bibliothèque C de UWIN est très complète [Kor97] :

- le mécanisme `fork()/exec()` est implémenté,
- la gestion des fichiers textes/binaires est effectuée en proposant des options O_BINARY et O_TEXT à la commande `open()`,
- la gestion des signaux est assurée par le lancement, dans chaque processus, d'une tâche (*thread*) dont le rôle est de scruter les signaux qui sont envoyés au processus courant,
- la commande `select()` proposée par UWIN sait gérer tous les types de descripteurs de fichiers, contrairement à la commande `select()` présente dans la bibliothèque `libwsock32.a` de MICROSOFT qui ne sait gérer que les descripteurs de fichiers associés à des sockets.

UWIN ne dispose pas d'un compilateur mais il contient des commandes `cc` et `ld` réalisant les appels appropriés à `cl` et `link` de MICROSOFT. De plus, la commande `cpp` est disponible.

Cette configuration impose que l'environnent de programmation C de MICROSOFT soit présent sur la machine. UWIN localise automatiquement `cl` et `link` sur le PC, mais il est possible de forcer le chemin à l'aide de la commande suivante :

```
export PACKAGE_cc="<chemin du compilateur Microsoft>"
```

UWIN devrait bientôt proposer une version permettant de réaliser une interface de compilation avec n'importe quel compilateur WIN32.

Dans le cadre d'un usage à but académique (éducation ou recherche), UWIN peut être utilisé librement et gratuitement. Lorsqu'il est utilisé à des fins commerciales, le prix de UWIN s'élève à 499 dollars pour la version UWIN BASE et 499 dollars pour la version UWIN SDK (voir `http://www.gtlinc.com/products/uwin/uwin.html`).

L'environnement UWIN comprend également les bibliothèques et interpréteurs TCL dont la version (8.0) est incompatible avec la communication par tube implémentée dans EUCALYPTUS et INSTALLATOR (voir section 4.2.6).

Interix

INTERIX est un sous-système (au sens WINDOWS du terme) qui vient en remplacement du sous-système POSIX livré par MICROSOFT. A ce titre, dans le cadre du processus de portage, il conviendrait simplement de porter le code source C vers un système INTERIX et recompiler [Int98a].

INTERIX est plus proche du système BSD bien que la gestion du terminal se rapproche davantage du SYSTEM V.

INTERIX ne supporte pas :

- les liens symboliques
- les membres `st_rdev` et `st_blocks` de la structure `stat`
- le montage de systèmes de fichiers

Pour isoler les fonctionnalités manquantes dans le code C, il est conseillé d'utiliser la directive `#ifdef __OPENNT`[20]

Comme pour la plupart des environnements UNIX fonctionnant sous WINDOWS, il arrive parfois qu'il manque certains fichiers `.h`. Il faut alors se poser la question de l'utilité de ce fichier et voir si la fonctionnalité à mettre en œuvre n'est pas spécifiée dans un autre fichier `.h` déjà existant. C'est là toute l'essence du portage.

INTERIX propose une commande "`cc`" permettant l'accès à un compilateur WIN32 standard du marché (`cl, bcc`). Par défaut, il utilise `cl`. Pour utiliser `cl`, il suffit simplement d'indiquer le chemin du compilateur et des DLL associées dans la variable `PATH`.

Pour utiliser un autre compilateur, il suffit de positionner les variables d'environnement suivantes :

[20] OpenNT est l'ancien nom de la solution INTERIX

- C89_COMPILER : chemin complet du compilateur
- C89_LINKER : chemin complet de l'éditeur de liens.

et d'indiquer le chemin du compilateur et des DLL associées dans la variable PATH.

La bibliothèque C d'INTERIX implémente de façon quasi-exhaustive la totalité des fonctions UNIX.

Concernant le portage des scripts shell, INTERIX propose un interpréteur ksh et de nombreuses commandes associées. Une des modifications principales à apporter dans les scripts provient du fait que INTERIX stocke ses binaires dans le répertoire $INTERIX_ROOT/bin et $INTERIX_ROOT/usr/contrib. Ainsi, tous les scripts qui positionnent la variable PATH à la valeur /bin:/usr/bin échoueront irrémédiablement [Int98b].

La communication par tube implémentée dans EUCALYPTUS et INSTALLATOR (voir section 4.2.6) semble ne pas fonctionner dans le cas du shell ksh proposé par INTERIX.

Nutcracker

NUTCRACKER [Sup98] est un environnement complet permettant de convertir des applications UNIX en applications WIN32 par simple recompilation de code grâce à des bibliothèques (DLL) fournies avec le produit. Il est ensuite possible d'optimiser les applications ainsi portées en utilisant des appels directs WIN32. Un des avantages pour les utilisateurs est donc de pouvoir maintenir leurs applications sous UNIX et WINDOWS à partir d'un code source commun.

Concernant l'environnement de développement, NUTCRACKER propose une interface avec le compilateur cl de MICROSOFT et inclut l'utilitaire gmake de GNU.

Sa bibliothèque C implémente la primitive fork() et offre une gestion des signaux très complète en supportant à la fois les signaux synchrones (violation d'accès) et asynchrones (tels que le ctrl-c). Il est possible d'utiliser la commande sigaction() pour spécifier la disposition du gestionnaire de signaux après la manipulation d'un signal.

De plus, NUTCRACKER "mappe" les permissions de fichiers UNIX vers WIN32. En revanche, il ne sait pas gérer les liens symboliques.

NUTCRACKER est la solution pour laquelle l'évaluation a été la moins détaillée du fait du prix peu attractif qu'elle propose : la licence de développement NUTCRACKER coûte entre 40000 et 50000 francs (HT) selon la version. En sus, il faut acquitter le prix du *runtime* nécessaire pour déployer l'application sur les postes clients, soit de 1000 à 3500 francs (HT) environ.

Mks Toolkit

MKS TOOLKIT n'est pas une solution de portage (construction des binaires au format WIN32) des applications écrites en C mais uniquement une batterie de commandes UNIX portées sous WINDOWS.

Néanmoins, MKS TOOLKIT propose une commande "cc" réalisant le lien avec un compilateur WIN32 du marché afin de compiler de façon native des programmes écrits en C. Ce lien est realisé grâce à un fichier de configuration : $ROOTDIR/etc/compiler.ccg .

Il est possible de définir un autre emplacement de ce fichier à l'aide de la variable d'environnement `CCG`. Il ne faut pas oublier d'indiquer le chemin du compilateur utilisé dans le `PATH`.

3.6 Solution adoptée

De nombreux éléments techniques et financiers ont conduit à réaliser le choix le plus proche des attentes de CADP. Il a été question de choisir la meilleure stratégie de portage et définir quels étaient les meilleurs outils associés, tout en respectant les objectifs de l'équipe VASY, d'une part, et des utilisateurs, d'autre part.

3.6.1 Pour la construction des binaires de CADP

Le projet VASY recherche, dans son environnement UNIX fonctionnant sous WINDOWS, les caractéristiques nécessaires à la production de son application CADP.

- La compilation croisée : c'est la stratégie choisie pour générer les exécutables de CADP. L'objectif visé par cette méthode est de ne maintenir qu'un seul code source destiné à être compilé de façon native (pour une utilisation sous UNIX) ou croisée (pour une utilisation sous WINDOWS NT). De plus, cette méthode offre l'avantage d'uniformiser la procédure de production des exécutables en n'apportant que quelques légères modifications dans l'outil Make-makefile de CADP. En outre, elle permet de générer des programmes au format WIN32 sans forcément disposer de machine WINDOWS ; ce qui évite bien des manipulations de transfert de code source vers d'autres architectures.

  Dans l'annexe A, nous verrons, en détail, la démarche de construction d'un compilateur croisé (*cross-compiler*).

- Les bibliothèques MINGW32 en compilation croisée : MINGW32 est un environnement (ensemble de bibliothèques et de fichiers `.h`) permettant de produire des exécutables au format WIN32 de façon croisée et en toute liberté : les binaires produits peuvent être distribués librement sans contrainte financière et technique puisque MINGW32 ne nécessite aucune DLL additionnelle. Ainsi, lorsqu'un programme est compilé de façon croisée à l'aide de MINGW32, l'exécutable "`.exe`" produit fonctionne directement en mode console sous WIN32 : cela signifie qu'il peut indifféremment être utilisé dans une console de commande MS-DOS ou dans l'interface graphique WINDOWS. De plus, MINGW32 est la solution choisie pour construire le compilateur croisé et les binaires WIN32 car ses bibliothèques permettent de couvrir la quasi-totalité des besoins du code C de CADP. La portion manquante nécessitera la modification du code afin d'en rétablir la portabilité.

CADP met en œuvre différentes caractéristiques non implémentées dans MINGW32 :

- la commande `fork()` : le chapitre suivant montrera comment nous avons pu nous passer de cette fonction utilisée pour la communication inter-processus dans l'IHM EUCALYPTUS.
- la commande `kill()` : cette fonction est, dans CADP, systématiquement associée à la fonction `getpid()` afin d'envoyer un signal au processus courant de la façon suivante :

  ```
  kill (getpid(), 15) ;
  ```

  MINGW32 implémente, dans sa gestion des signaux, la commande `raise()` dont l'objectif est d'envoyer un signal au processus courant. Cela convient parfaitement. Ainsi, il suffira de remplacer la commande ci-dessus par la commande suivante :

  ```
  raise(15) ;
  ```

- l'initialisation de variables globales par des valeurs non constantes : CADP utilise des commandes d'initialisation de variables globales de la forme suivante :

  ```
  FILE    *mystdin = {stdin} ;
  ```

  Bien que "`cc`" et "`gcc`" acceptent ce genre de programmation, elle ne correspond pas à la norme ANSI et, à ce titre, MINGW32 ne la prend pas en compte, considérant `stdin` comme une valeur non constante (message : `initialiser is not constant`). C'est pourquoi, on lui préfèrera la notation suivante :

  ```
  FILE    *mystdin ;
  ...
  main () {
          mystdin = {stdin} ;
  }
  ```

  Cela engendrera quelques modifications dans le code source.
- Les fonctions suivantes (prototypées dans le fichier `strings.h`) sont absentes de MINGW32 : `bcmp` (comparaison de 2 chaînes), `bzero` (initialisation à une chaîne contenant des zéro), `bcopy` (copie d'octets d'une chaîne vers une autre), `index` et `rindex` (renvoi de pointeurs dans une chaîne de caractères). Ces fonctions peuvent aisément être remplacées par des fonctions prototypées dans le fichier `string.h`, comme le montre le tableau suivant :

Ancienne formulation	**Nouvelle formulation**
`bcmp (s1, s2, n)`	`memcmp (s1, s2, n)`
`bzero (s, n)`	`memset (s, 0, n)`
`bcopy (s1, s2, n)`	`memcpy (s2, s1, n)`
`index (s, c)`	`strchr (s, c)`
`rindex (s, c)`	`strrchr (s, c)`

Parce que chaque problème rencontré n'est pas resté sans solution, MINGW32 s'est avéré la solution idéale pour la production des binaires de CADP.

CYGWIN répond également aux exigences du code source présent dans la boîte à outils CADP, mais les binaires produits nécessitent la DLL `cygwin.dll` pour fonctionner. Cependant, les fonctionnalités mises en œuvre dans ses bibliothèques sont plus nombreuses, et à ce titre, nous avons choisi de construire également un compilateur croisé en utilisant CYGWIN afin de répondre, dans l'avenir, aux éventuelles nouvelles fonctionnalités mises en œuvre dans CADP et non implémentées dans MINGW32, pour lesquelles aucune solution basique ne pourrait être envisagée.

- EGCS pour la compilation croisée : seuls `gcc` et `egcs` peuvent être associés à MINGW32 pour la construction du compilateur croisé du fait des macros programmées dans les fichiers `.h` (MINGW32 a été développé pour `egcs` ou `gcc`). Pour cette raison, et parce que les compilateurs `cl` et `bcc` présentent certaines limitations en ce qui concerne le pré-processeur (pas de génération de dépendances possible : équivalent de `cpp -M` ou `gcc -M`, par exemple), notre choix s'est naturellement dirigé vers `gcc` et `egcs`.

  En 1992, la version 1 de `gcc` a atteint un point de stabilité permettant d'assurer un bon fonctionnement vers les architectures cibles qu'elle pouvait supporter. Cette version présentait certaines limitations inhérentes à sa conception et difficiles à résoudre ; c'est pourquoi un effort important fût fait et la version 2 de `gcc` en a été le résultat. Lorsque `gcc2` a atteint un état utile et satisfaisant, le développement de la version 1 a été stoppé. C'est exactement ce qui se passe aujourd'hui avec `egcs` basé sur `gcc2.8` et devant remplacer `gcc` dans un avenir très proche.

  De plus, depuis 18 mois de nombreuses entités ont contribué à l'amélioration et la correction d'erreurs sur la version 2 de `gcc`. En bref, les principales différences entre `gcc` et `egcs` résident essentiellement dans les macros d'optimisation du binaire produit et dans le mode de contribution dans lequel se sont engagées de grandes institutions pour assurer à `egcs` une grande pérennité.

  Toutes ces raisons ont conduit à choisir `egcs` comme compilateur associé à MINGW32 pour construire le compilateur croisé.

3.6.2 Pour l'environnement utilisateur de CADP

Les utilisateurs de CADP ont, pour leur part, des besoins qui sont liés à l'usage de la boîte à outils CADP. Dans cette optique, nous retiendrons les solutions CYGWIN et MKS TOOLKIT.

- CYGWIN :

  D'une part, CADP est écrit en grande partie en langage C, d'autre part, les applications qui le constituent génèrent du code C, qu'elles compilent et exécutent, à la volée.

  C'est pourquoi l'environnement utilisateur de CADP doit fournir un ensemble complet de commandes UNIX (et un interpréteur Shell) et un environnement de compilation de programmes C.

Cygwin semble la solution répondant le mieux à ces contraintes parce qu'il apporte, gratuitement (outils GNU) :

- un Shell : `bash`,
- de nombreuses commandes Unix,
- des bibliothèques C très complètes,
- un environnement de compilation exhaustif : `gcc`, `make`, etc.

Parmi les commandes Shell proposées par Cygwin et utilisées par Cadp, il manque néanmoins : `more`, `which` et `mail`. En remplacement de `more` et `which`, Cygwin propose respectivement les commandes `less` et `type`. Pour ce qui est de la commande `mail`, il a été choisi d'avoir recours à une solution externe (`blat`) comme cela est précisé dans le chapitre 4.

- Mks Toolkit :

  Mks Toolkit répond, de par son interpréteur Shell (`ksh`) et ses nombreuses commandes Unix associées, aux exigences des contraintes de l'utilisation de la boîte à outils Cadp. Cependant, son environnement de compilation se limite à la simple commande "`cc`" qui assure le lien avec un réel compilateur Win32 du marché. Par conséquent, Mks Toolkit ne peut être utilisé pour Cadp que s'il est accompagné de l'environnement Microsoft C ou Borland C.

Note : bien que seuls Cygwin et Mks Toolkit proposent une solution complètement adaptée à Cadp, les utilisateurs pourront choisir Uwin ou Interix pour une utilisation limitée. En effet, ils ne pourront pas mettre à profit les interfaces graphiques Eucalyptus et Installator qui mettent en œuvre un mécanisme de communication par tubes que Uwin et Interix ne savent pas prendre en compte.

3.6.3 Synthèse

Pour récapituler, les solutions préconisées pour le portage et l'utilisation de Cadp sont les suivantes :

- la compilation croisée à l'aide du compilateur Egcs et des bibliothèques Mingw32 pour produire les exécutables de la boîte à outils Cadp,
- Cygwin ou Mks Toolkit (associé à un compilateur natif Win32) pour l'utilisation de Cadp.

Chapitre 4

Réalisation

Le but de ce chapitre est de donner une vision précise du travail réalisé. Plus technique que les précédents, il donne des détails sur la méthode employée pour le portage réel des applications de la boîte à outils CADP.

Dans un premier temps, nous verrons quels sont les aspects du portage des composantes de l'environnement de développement des outils CADP que sont les scripts shell, les fichiers Makefile. Nous aborderons cet aspect en présentant également le compilateur croisé et l'analyseur `scrutator`, des outils que nous avons réalisé pour faciliter le portage des applications écrites en langage C.

Dans un second temps, nous nous placerons du côté de l'utilisateur pour étudier le portage des interfaces graphiques composant la boîte à outils CADP à travers EUCALYPTUS et INSTALLATOR.

Puis, nous étudierons la migration des commandes purement système suivantes :

- envoi de mail,
- consultation de pages de manuel
- visualisation d'un fichier Postscript,
- consultation des propriétés d'un fichier,
- appel à un éditeur de texte,
- impression d'un document,
- appel à un navigateur Internet,
- ouverture d'un interpréteur shell en mode console.

Ensuite, nous détaillerons le portage d' INSTALLATOR sous l'angle du protocole FTP.

Une section sera également consacrée à diverses autres améliorations apportées aux interfcaes graphiques.

Enfin, à travers les programmes intégrés dans la boîte à outils CADP, nous présenterons les aspects de la migration du code C et des langages d'analyse lexicale et syntaxique LEX et YACC.

4.1 Portage de l'environnement de développement

4.1.1 Portage des scripts shell

A l'exception de deux scripts écrits en `csh` que nous avons choisi de convertir en Bourne Shell (`sh`), la première ligne des scripts shell (*shebang*) de la boîte à outils CADP est la suivante : `#!/bin/sh`. Cela signifie qu'ils sont interprétés par le Bourne Shell (`sh`). Cependant, cette ligne est inopérante sous WINDOWS ; un script shell est toujours interprété par le shell courant qui l'exécute.

Tous les scripts shell de CADP sont écrits en utilisant le Bourne Shell, ce qui leur garantit la plus grande capacité de portabilité. Les solutions envisagées sous WINDOWS NT proposent deux types de shell :

- le Bash Shell
- le Korn Shell

Tous deux assurent une compatibilité descendante avec le Bourne Shell. Cependant, certaines commandes ont une réaction sensiblement différente selon le shell qui les exécute. C'est pourquoi un certain nombre d'aménagements ont dû être consentis afin d'assurer la stabilité des applications, comme le montre le tableau suivant :

Script	Modification apportée

`xeuca_info`	Le découpage “`cut -f`” réalisé à partir du résultat de la commande “`ls -lg`” a été revu car il peut être perturbé par un nom de groupe d'utilisateurs comportant des espaces, ajoutant ainsi des colonnes supplémentaires au résultat du “`ls`”.
	Exemple avec 9 colonnes (correct) :
	`-rwxrwxrwx 1 0 everybody 3545 Jun 25 1999 Monfichier`
	Exemple avec 11 colonnes (incorrect) :
	`-rwxrwxrwx 1 0 Tout le Monde 3545 Jun 25 1999 Monfichier`

`xeuca_ps`	Ce script affiche la liste des processus attachés à l'utilisateur courant à l'aide de la commande 'ps -x'. Or, l'option "`-x`" n'existe pas sous CYGWIN. Il a donc été décidé de supprimer cette option qui affichait également les processus sans contrôle terminal.

Script	Modification apportée
`run_uncompress`	Présent dans INSTALLATOR, ce script permet d'extraire par décompression, les fichiers nécessaires au fonctionnement de CADP, en mettant en œuvre la commande `uncompress`. Or, cette commande est absente de la distribution CYGWIN, ce qui a conduit à utiliser la commande "`gzip -d`" dans le cas d'une utilisation de la solution de Cygnus.
`rfl`	MKS TOOLKIT proposant une commande `rsh` signifiant "*Restricted Shell*" représentant une source de conflit possible avec l'utilitaire TCP/IP `rsh` (*Remote Shell*) , il est important de contrôler la variable d'environnement `$PATH` afin de positionner le chemin du système WINDOWS NT(`$SYSTEMROOT/system32`) avant le répertoire des binaires MKS TOOLKIT.
`tst`	Ce script est destiné à contrôler la présence des commandes et propriétés nécessaires au bon fonctionnement de CADP en parcourant l'ensemble des répertoires présents dans la variable `$PATH`. Dans le cadre du portage vers WINDOWS NT, il a été indispensable d'ajouter un certain nombre de tests tels que ceux concernant les commandes `write.exe` et `iexplore.exe`. Par ailleurs, il s'agit de tester la présence de la commande `less` si la commande `more` n'existe pas (cas de la solution CYGWIN)

De plus, certaines modifications d'ordre général ont dû être réalisées afin d'optimiser l'utilisation de CADP. Notamment, il s'est avéré utile de renommer tous les fichiers textes manipulés par CADP (instructions, aides, informations diverses) en les suffixant de l'extension `.txt` afin de permettre l'ouverture par double-clic de l'éditeur par défaut (`notepad`, généralement) sous WINDOWS NT.

4.1.2 Portage des fichier Makefiles

Les Makefiles sont des fichiers destinés à automatiser la construction d'un ou plusieurs programmes exécutables en gérant des dépendances et des actions à lancer lorsque certaines règles deviennent vraies (voir section 2.3.2).

Dans le cadre de la compilation croisée, stratégie choisie pour la construction des binaires WIN32 de CADP, les Makefiles continuent d'être lancés depuis la machine hôte. Cela signifie que la syntaxe des Makefiles reste identique pour le portage des applications de la boîte à outils CADP.

Les outils `Make-makefile` et `make.Make-makefile` réalisés par l'équipe VASY, permettant la construction et l'exécution automatique de Makefiles selon l'architecture désirée, ont néanmoins été modifiés, d'une part, pour prendre en compte la nouvelle architecture et, d'autre part, pour permettre la construction "native" de CADP sous WIN32 dans l'environnement

CYGWIN. De ce fait, il a été nécessaire de réaliser les modifications suivantes :

- utilisation de BISON au lieu de YACC pour la génération croisée (vers WIN32) d'analyseurs syntaxiques,
- utilisation de FLEX au lieu de LEX pour la génération croisée ou native d'analyseurs lexicaux,
- utilisation de BYACC au lieu de YACC pour la génération native d'analyseurs syntaxiques,
- prise en compte de deux architectures : celle sur laquelle on se trouve au moment de l'exécution du `make` et celle pour laquelle on souhaite construire les exécutables.

De plus, l'appel de `Make-makefile` prend un argument supplémentaire, comme suit :

```
Make-makefile [ $ARCH ]
```

où l'option facultative `$ARCH` définit l'architecture-cible.

Le traitement de l'argument `$ARCH` est le suivant :

- Si l'option `$ARCH` est absente, alors on considère que `$ARCH` est égale à l'architecture de la machine-hôte. On est alors dans le cas d'une compilation native.
- Si l'option `$ARCH` est présente et identique à l'architecture de la machine-hôte, alors on est encore dans le cas d'une compilation native.
- Si l'option `$ARCH` est présente et différente de l'architecture de la machine-hôte, alors on est dans le cas d'une compilation croisée. Dans ce cas, `$ARCH` définit l'architecture de la machine cible.

Enfin, dans le cadre de la compilation par `gcc` en compilation croisée ou en natif sous WIN32 (compilation "à la volée" des programmes C générés par CADP), l'option "`-ansi`" avait pour effet d'inhiber l'inclusion de nombreuses fonctions de `string.h` (`strdup` par exemple). Pour pallier ce problème, il aurait fallu ajouter la ligne :

```
extern char *strdup() ;
```

dans les sources de CADP.

Nous avons finalement choisi d'ajouter l'option `-U__STRICT_ANSI__` (qui inhibe l'option `-ansi`) sur la ligne de compilation `gcc` et d'enlever les lignes :

```
extern char *strdup() ;
```

Note : l'exécution de l'outil `Make-makefile` n'est pas envisageable de façon native dans l'environnement MICROSOFT car son compilateur `cl` ne dispose pas d'option pour générer les dépendances d'un fichier objet, contrairement à `gcc` (`gcc -M`) ou `cc` (`cpp -M`).

4.1.3 Compilation croisée

Comme il en a déjà été question à la section 3.5.1, la compilation croisée à destination de WINDOWS NT ne peut être réalisée qu'à l'aide des bibliothèques CYGWIN et MINGW32.

Cela signifie qu'un programme ne pourra être compilé de façon croisée que si chaque fonction qu'il met en œuvre est présente dans l'une de ces deux bibliothèques. Dans le cas où elles n'existeraient pas, il faudrait :

- soit modifier le code source pour trouver un palliatif,
- soit développer les primitives adéquates réalisant leur implémentation dans l'API WIN32.

Dans cette dernière option, nous avons utilisé des structures telles que :

```
#ifdef _win32
        ... code API win32
#else
        ... code Unix
#endif
```

A ce jour, nous n'avons choisi d'implémenter que les compilateurs croisés WIN32 opérant dans les systèmes d'exploitation suivants :

- SUN OS 5.X
- LINUX

L'annexe A présente la procédure de construction du compilateur croisé SUN OS 5.X vers WIN32 utilisant la bibliothèque MINGW32 et le compilateur EGCS.

4.1.4 Analyseur de code C : Scrutator

Afin de prévenir des différents problèmes potentiels liés au portage vers WIN32, il a été réalisé un outil permettant d'analyser le contenu des sources écrits en langage C.

Cet outil, écrit en langage de script `sh` (Bourne Shell), réalise des rapprochements sur les chaînes de caractères (*pattern matching*) à l'aide d'expressions régulières et à la commande `grep`. Exemple :

```
/bin/grep -ns popen $file | awk ' BEGIN {FS=":"}
        $0 ~ /\$/ {printf"\n\tLine %s - %s",$1,$2;i=$1} END {exit i}'
```

Cette séquence réalise un test de présence du caractère “$” associé à la commande `popen`, qui peut représenter un problème potentiel de portage. L'option `-ns` permet d'afficher le numéro de la ligne sur laquelle est éventuellement présent un tel problème et supprime tous les messages d'erreurs liés aux fichiers non lisibles ou inexistants.

De façon générale, cet outil, nommé `scrutator`, est capable de déceler et de signaler par un message d'avertissement (*warning*) les problèmes suivants :

- absence de l'option “`b`” dans l'appel `fopen()` lors de l'ouverture d'un fichier binaire,
- utilisation de la commande `fseek()` utilisée pour parcourir un fichier binaire,
- substitution de variable inopérante dans le cas d'un programme compilé de façon croisée avec MINGW32,
- test de l'architecture WIN32 non réalisé dans une séquence contenant des tests de l'architecture courante (`ARCHITECTURE_SUN_4, ARCHITECTURE_SUN_5`, etc),
- utilisation de sockets sans inclusion du fichier `winsock.h`,
- chemins de fichiers “codés en dur” dans le programme,
- initialisation de variables à `stdin`, `stdout` ou `stderr` avant le `main()`,
- utilisation de la commande `fork()` absente de la bibliothèque MINGW32,
- utilisation de la commande `fstat()` manipulant les attributs de fichier pouvant causer des problèmes sous WIN32,
- utilisation de la commande `symlink()` manipulant des liens symboliques seulement valable dans un système de fichiers NTFS,
- utilisation de la commande `kill()` utilisée sur un autre processus que le processus courant (`getpid()`),
- mise en œuvre de relations entre processus (inopérantes sous WIN32),
- utilisation d'un signal non traité dans la bibliothèque MINGW32,
- utilisation d'appels `strings.h` inexistants dans MINGW32,
- utilisation de commandes absentes dans MINGW32 telles que `rint()`, `setpgrp()`, `cuserid()`, `ftruncate()`, `clock()`, `times()`, `brk()`, `sbrk()`, `mount()`, `umount()`, `getgroups()`.

Le script `scrutator` donne également un début de solution à chaque problème répertorié. L'annexe B présente une synthèse des solutions proposées pour chaque problème potentiel.

4.2 Portage du code Tcl de l'interface graphique

Bien que le langage TCL soit interprété et considéré comme entièrement portable, la première exécution d'EUCALYPTUS sous WINDOWS NT a révélé des différences liées au gestionnaire de fenêtre (*Window Manager*) de chacune des deux plate-formes.

4.2.1 Polices de caractères

EUCALYPTUS utilisait trois polices de caractères différentes codées “en dur” dans le programme. Nous avons tout d'abord créé des variables d'environnement positionnées dans le script de lancement de l'interface (`xeuca`), afin d'en faciliter les paramétrages selon l'architecture utilisée :

`XEUCA_TITLE_FONT` : police du titre
`XEUCA_LEFT_WINDOW_FONT` : police de la fenêtre des icônes
`XEUCA_RIGHT_WINDOW_FONT` : police de la fenêtre des résultats.

Sous WIN32, TK reconnaît le format X-Font (XLFD) [Ous94] qui se présente ainsi :

`-FDY-FAM-WGT-SLT-SWT-ADS-PIX-PTS-RSX-RSY-SPC-WDT-CHS-ENC`

Ce format est représenté par un certain nombre de rubriques qui spécifient la police de caractères à utiliser, comme l'indique le tableau suivant :

FDY (foundry)	représente le nom de la société conceptrice de la police (Adobe,B&H, ...)
FAM (font family)	est le nom de la famille de police (times, courier ...)
WGT (weight)	représente l'épaisseur du caractère (bold, medium)
SLT (slant)	est le style (italic, roman, oblique ...)
SWT (setwidth)	est la largeur du caractère (normal, condensed)
ADS (additional style)	est une information supplémentaire sur le style
PIX (pixels)	est la hauteur du caractère en pixels
PTS (point size)	représente dix fois la taille d'un caractère donnée ci-dessus
RSX	représente le nombre de points par pouce dans la direction des X

RSY	représente le nombre de points par pouce dans la direction des Y
SPC (spacing)	représente l'espacement entre caractères (c,m,p) avec : 'c' pour Character-cell/monospaced 'm' pour monospaced 'p' pour proportional
WDT (average width)	représente dix fois la largeur moyenne en pixels
CHS et ENC (character set registry)	représentent l'encodage de la police : iso8859-1 Western Europe hp-roman8 Western Europe (Hewlett-Packard only) iso8859-2 Eastern Europe ksc5601.1987-0 Korean jisx0208.1983-0 Japanese

Exemple :

```
-adobe-courier-bold-r-normal-12-120-75-75-m-70-iso8859-1
```

Sous Sun Os, il est possible d'obtenir la liste des polices installées grâce à la commande "`xlsfonts | more`".

Le problème rencontré réside dans le fait que l'on puisse utiliser le caractère joker `"*"` dans le nom d'une police.

L'utilisation de `"-Adobe-Helvetica-Medium-R-Normal-*-240-*"`, par exemple, peut représenter une police unique sous Sun Os (`-adobe-helvetica-medium-r-normal-34-240-100-100-p-176-iso8859-1`, par exemple) si cette dernière est la seule installée. Mais, si sous Windows NT, la police "`-adobe-helvetica-medium-r-normal-240-2400-100-100-p-176-iso8859-1`" est installée, il en résulte que la taille de la police choisie sera 10 fois supérieure à la taille désirée.

En conclusion, il est important d'éviter au maximum le caractère joker dans la dénomination de la police désirée.

4.2.2 Configuration de l'écran

La fenêtre principale d'Eucalyptus est composée de deux cadres (*frames*) positionnés de façon précise et flanqués, chacun, d'un ascenseur sur sa droite. Cette précision est dépendante du *WindowManager* utilisé. Sous Windows NT, du fait de la taille des ascenseurs et des bordures des fenêtres, l'affichage est sensiblement différent. Il a donc fallu créer des variables d'environnement qui permettent de positionner et prendre en compte ces différences.

$XEUCA_SCREEN_WIDTH et $XEUCA_SCREEN_HEIGHT désignent, respectivement, le nombre de pixels définissant la largeur et la hauteur actuelle de la fenêtre Eucalyptus (quantité variable).

XEUCA_LEFT_WINDOW_WIDTH désigne le nombre de pixels entre le bord gauche de la fenêtre de gauche et le bord droit de l'ascenseur associé à la fenêtre de gauche (quantité constante).

XEUCA_CHARACTER_WIDTH désigne la largeur (en pixels) d'un caractère de la fenêtre de droite, dont la police est de taille fixe (quantité constante).

Sous UNIX, ces paramètres prennent la valeur :

XEUCA_LEFT_WINDOW_WIDTH=504

XEUCA_CHARACTER_WIDTH=6

Sous WIN32 :

XEUCA_LEFT_WINDOW_WIDTH=492

XEUCA_CHARACTER_WIDTH=7

Ces paramètres sont dépendants de la police de caractère choisie.

Pour calculer la taille (en pixels) de la fenêtre principale, il est possible d'utiliser la commande TCL suivante :

```
[wm geometry .]
```

Problème sous WIN32 : au lancement de l'interface, il se peut que "`wm geometry .`" renvoie une valeur erronée, de la forme `"1x1+X+Y"`. Ce comportement est attribué à un conflit entre des *threads* concurrents. On ne peut donc pas exploiter la taille réelle de l'écran à l'initialisation d'EUCALYPTUS.

4.2.3 Terminal X-Window

Il faut extraire du code TCL tout appel à des commandes X-Window inexistantes sous WINDOWS. En effet, la commande '`xterm`' n'existe pas ; il est donc inutile de positionner des paramètres d'ouverture de fenêtre XTERM_OPTIONS.

De plus, la configuration du terminal X-Window est enregistrée dans un fichier texte '`~/.Xdefaults`'. L'affichage est donc dépendant de cette configuration.

A titre d'exemple, EUCALYPTUS est mal cadré lorsque l'on positionne la variable `BorderWidth` (qui matérialise la largeur des bordures, dans ~/.Xdefaults) à une valeur supérieure à 1. Il a donc fallu prendre en charge cette fonction dans EUCALYPTUS en la repositionnant à 1, si elle est supérieure à 1.

Commandes TCL insérées :

```
frame .xoptions -class BorderWidth
```

```
set Border_Width [expr [option get .xoptions {} BorderWidth] + 0]

if [expr $Border_Width > 1] {
        option add *BorderWidth 1
}
```

Ces commandes permettent d'associer un cadre (*frame*) à la classe de données `BorderWidth`, puis d'associer la largeur de bordure obtenue à la variable locale `Border_Width`. La lecture des données de terminal X impose la création d'un cadre spécifique.

Il suffit alors simplement de tester la valeur de cette variable. Si cette valeur est supérieure à 1, alors on la repositionne à 1.

Note : sous WIN32, ces commandes sont présentes mais les résultats qu'elles renvoient ont une valeur nulle. Elles ne sont présentes que pour une compatibilité de syntaxe.

4.2.4 Menus déroulants

La gestion des menus laisse apparaître des différences de comportement entre WINDOWS NT et UNIX. Notamment, l'option "`-postcommand`" de la commande "`menu`" cumule les menus relatifs à un bouton à chaque fois que l'on clique dessus, sous WINDOWS NT.

Ce problème a été évoqué à la société Scriptics[21] qui l'a enregistré sous la référence 1480[22] afin de le solutionner dans l'une des prochaines versions de TK. En attendant, il a été choisi de pallier ce dysfonctionnement en reconstruisant le menu à chaque fois qu'il est invoqué, en le vidant et en le remplissant à nouveau.

4.2.5 Gestion de la souris

EUCALYPTUS met à disposition un outil, nommé `bcg_edit`, écrit en TCL, qui permet de consulter interactivement la représentation d'un graphe BCG au format Postscript.

Cet outil met à profit tous les avantages de la souris à 3 boutons fréquemment utilisée sous UNIX, notamment le bouton central pour lequel une procédure peut être associée à l'aide de la commande suivante :

```
.canvas bind <ButtonPress-2> {Proc}
```

Sous WINDOWS NT, parce que la plupart des souris ne comporte que 2 boutons, nous avons choisi de simuler le bouton central par la combinaison de la touche `shift` et du bouton gauche de la souris. De plus, il a été nécessaire de réaliser la même portion de code TCL

[21] Voir URL : `www.scriptics.com`

[22] voir http://www.scriptics.com/resource/software/patches/tk81b1

selon que l'utilisateur utilise le bouton central de la souris (sous Unix) ou qu'il fasse usage de la combinaison `shift` et bouton gauche de souris. Cette opération est réalisée par la création d'un nouvel évènement utilisateur comme le montre la séquence Tcl suivante :

```
event add «MON_NOUVEL_EVENEMENT» <ButtonPress-2>
event add «MON_NOUVEL_EVENEMENT» <Shift-ButtonPress-1>

.canvas bind «MON_NOUVEL_EVENEMENT» {Proc}
```

4.2.6 Communication par tube (*pipe*)

Les difficultés rencontrées lors du portage de l'interface Eucalyptus se sont articulées autour du mode de communication entre l'IHM elle-même et le Shell qui interprète les commandes construites par l'utilisateur.

Pour régler ce problème de communication, plusieurs alternatives se sont présentées :

- implémenter à partir des appels `fork()`, `dup()` et `pipe()` fournis dans l'environnement Cygwin,
- redévelopper le mécanisme mis en œuvre dans le programme C `duplex` directement dans l'Api Win32 (se reporter à la figure 2.5 de la section 2.4.2),
- modifier le source Tcl en lui ajoutant la gestion des tubes interne à l'interpréteur `tclsh`.

C'est la dernière proposition que nous avons choisi de retenir. Cette alternative permet donc de supprimer le programme `duplex`, ce qui permet d'une part, de se passer d'appels trop dépendants de l'architecture (`fork() ...`) et d'ôter, d'autre part, un programme à la longue liste des exécutables Cadp à maintenir.

Les dernières versions de Tcl/Tk (à partir de TK8.0) offrent la possibilité d'ouvrir des tubes afin d'exécuter des commandes, de leur associer un identificateur de sortie standard et de scruter chaque caractère reçu sur cet identificateur afin de le traiter comme résultat.

Pour réaliser l'équivalent du programme `duplex` dans Eucalyptus, plusieurs étapes ont été nécessaires :

Ouverture d'un tube à chaque commande envoyée vers le Shell : la syntaxe de ce mécanisme sous TCL a la forme suivante :

```
if [catch {open "|sh -c $Command" r} tube] {
        Erreur
} else {
        fileevent $tube readable Log
```

```
}

proc Log {} {
        if [eof $tube] {
                if {[catch {close $tube} status]} {
                        Display "Erreur $status"
                } else {
                        Display "Fin de Commande"
                }
        } else {
                gets $tube line
                Display $line
        }
}
```

La commande à exécuter est représentée par la variable `$Command`. L'exécution de cette commande est réalisée par l'ouverture, en mode lecture (`r`) d'un tube associé à un descripteur entier (`tube`).

La commande `fileevent` permet de démarrer un mécanisme d'interception de tous les messages de sortie envoyés par la commande associée à l'identificateur de fichier `$tube`. A chaque fois qu'un message (ligne de caractères terminée par CRLF) est intercepté, la procédure `Log` est exécutée. Cette procédure récupère les lignes reçues et les affiche (`Display $line`) jusqu'à ce que EOF soit intercepté. Dès lors, le descripteur `tube` est fermé et le diagnostic final de la commande est affiché.

La séquence d'ouverture du tube est précédée de `'sh -c'`. Cette mention est inutile sous Unix car chaque commande est nécessairement interprétée par un Shell. Or, sous Windows, il est indispensable de préciser si la commande sera interprétée par un Shell ou par l'Api Win32 (`command.com` ou `cmd.exe`) ; sans cette option, cela génèrerait inévitablement des erreurs.

Ouverture d'un tube unique pour toutes les commandes : la syntaxe Tcl présentée ci-dessus a ensuite été améliorée afin de rapprocher le mécanisme souhaité de l'ancien mode de fonctionnement implémenté par `duplex`. Cela a donné le code TCL suivant :

```
if [catch {open "|sh" r+} tube] {
        Erreur
} else {
        fconfigure $tube -buffering none
        fileevent $tube readable Log
}
```

```
proc Log {} {
        if {[gets $tube line]<0} {
                close $tube
        } else {
                Display $line
        }
}

puts $tube $Command
```

Le principe de cette solution consiste à n'ouvrir qu'un seul tube tout au long de l'exécution d'EUCALYPTUS, d'envoyer chaque commande construite par l'utilisateur sur l'entrée standard du tube (à l'aide de la commande `puts`) et de recevoir les résultats produits à l'aide de la commande `gets`. Le mode d'ouverture est cette fois "`r+`", ce qui signifie que le descripteur de fichier `$tube` est accessible en lecture et en écriture. Cette solution présente les avantages suivants :

– comparée à la solution précédente (ouverture d'un tube par commande), elle se rapproche davantage du mécanisme `duplex`,
– elle est moins coûteuse en temps CPU, puisqu'elle n'ouvre qu'un seul tube, alors que la solution précédente en ouvrait un par commande produite,
– minimise les changements à réaliser dans EUCALYPTUS.

Les différents tests réalisés autour de cette méthode dans les environnements CYGWIN et MKS TOOLKIT ont conduit au constat suivant : le Shell (`bash`) de CYGWIN fonctionne en mode bufferisé (bloc de 512 octets) pour les flux de caractères associés aux tubes, aux sockets et aux fichiers.

Dans le mode de fonctionnement d'EUCALYPTUS, il faut "flusher" (c'est à dire, vider le tampon mémoire) les sorties à chaque caractère intercepté . Pour cela, il suffit de modifier la commande TCL permettant la création du tube de la manière suivante :

```
[catch {open "|sh -i" r+} tube]
```

L'option `-i` du shell permet de le rendre interactif, donc bufferisé en mode caractère.

La dernière crainte liée à ce mode de fonctionnement provient de l'éventualité de collision entre les commandes envoyées et les résultats reçus sur le tube. En effet, ce dernier est matérialisé par un seul descripteur de fichier, ce qui laisse supposer que des collisions peuvent avoir lieu.

Un examen minutieux du code source du langage TCL a révélé le fait que le caractère bi-directionnel du tube “cache” en réalité l’implémentation de deux tubes bien distincts. L’interpréteur de TCL utilise alors l’un ou l’autre selon que la commande invoquée est `puts` ou `gets`.

Fusion de `stdout` et `stderr` : pour en terminer avec la similitude avec `duplex`, il restait une fonction essentielle à implémenter : la fusion de `stdout` et `stderr`.

En effet, dans `duplex`, le descripteur de fichier numéro 2 est fermé et le descripteur de fichier numéro 1 est dupliqué (commande `dup()`), ce qui a pour effet de fusionner la sortie standard et celle d’erreur afin d’afficher tous les messages de façon synchronisée dans la fenêtre des résultats d’Eucalyptus.

Pour cela, il suffit de modifier la commande d’ouverture du tube vue précédemment de la façon suivante :

```
[catch {open "|sh |& cat" r+} tube]
```

La fusion est assurée par l’opérateur `|&` et l’exécution d’un filtre neutre `cat`. Cependant, cette notation n’autorise pas l’interactivité du shell (option `-i`) vue précédemment. Ce problème est solutionné par l’ajout de l’option `-u` (*unbufferized*) à la commande `cat`.

La version finale du mécanisme d’ouverture de tube prend donc la forme suivante :

```
[catch {open "|sh |& cat -u" r+} tube]
```

Il est important de souligner que cette formulation ne peut fonctionner que sous Windows NT ou Windows 98, car le mécanisme d’ouverture de tube est inopérant sous Windows 95.

4.2.7 Conversion Texte/Binaire

Dans l’IHM Eucalyptus, les commandes constituées sont envoyées vers un interpréteur Shell qui s’exécute en parallèle. L’envoi de ces commandes est réalisé par la commande “`puts`” qui écrit sur l’identificateur de canal (descripteur de fichier) associé au Shell. La syntaxe en est la suivante :

```
puts <$Channel_id> <$String>
```

Cette commande envoie également une séquence “`\n`” après la chaîne `$String`, ce qui convient tout à fait pour réaliser la validation de la commande envoyée vers le Shell.

Dans les scripts TCL, la fin de ligne est toujours représentée par un simple caractère "`\n`". Cependant, pour les fichiers et les périphériques (*devices*), la fin de ligne peut être représentée de façon différente selon les plate-formes.

En lecture, le système de gestion des entrées/sorties TCL (TCL/IO) traduit automatiquement la représentation de fin de ligne "`\r\n`" en "`\n`". En écriture, TCL/IO réalise exactement l'opération inverse.

WINDOWS NT utilise la convention DOS de 2 caractères "`\r\n`" pour réprésenter la fin de chaque ligne dans un fichier texte. UNIX, lui, utilise uniquement le caractère "`\n`" pour la même opération.

Le mode binaire traite un fichier comme une séquence d'octets sans les interpréter. Le mode texte supprime un "`\r`" en face de chaque "`\n`" en lecture et insère un "`\r`" en face de chaque "`\n`" en écriture.

Certains interpréteurs Shell fonctionnant sous WINDOWS NT réalisent eux-mêmes cette conversion bi-latérale. C'est le cas du Bash Shell de Cygnus. Le problème est que TCL, par défaut, réalise lui aussi cette conversion. Donc, dans le cas d'un script TCL exécuté sous le Bash de Cygnus, il faut inhiber la traduction car le Shell interprète le caractere "`\r`" comme un octet de plus et non comme un simple retour chariot.

Pour solutionner ce problème, la première idée a été de supprimer le "`\n`" envoyé automatiquement par la commande "`puts`" grâce a son option `-nonewline` et de le gérer manuellement dans la chaîne transmise de la façon suivante :

```
puts -nonewline $Channel_ID "$String \012"
```

Mais, TCL réalisant la même conversion en sortie, le caractère "`\n`" ou "`\012`" devient "`\r\n`".

La seconde idée, fructueuse celle-là a été de trouver une option de TCL qui permet d'éviter cette conversion. Il suffit de rajouter la commande suivante après la création du descripteur de fichier associé au Shell :

```
fconfigure $Channel_ID -translation binary
```

Cette option permet d'éviter toute conversion de fin de ligne et satisfait donc le mode de fonctionnement associé à CYGWIN.

Pour les autres environnements UNIX fonctionnant sous WINDOWS, il faut conserver une conversion "`\n`" vers "`\r\n`" en utilisant la syntaxe :.

```
fconfigure $Channel_ID -translation crlf
```

Pour gérer ces deux types de fonctionnement, il a été nécessaire de modifier la commande "`arch`" (script shell renvoyant l'architecture sur laquelle il est exécuté) afin de connaître l'émulation UNIX utilisée sous WIN32.

Dès lors, l'option `-detailed` de la commande "`arch`" affiche :

win32-cygnus	pour CYGWIN
win32-mks	pour MKS TOOLKIT
win32-uwin	pour UWIN
win32-interix	pour INTERIX

De plus, nous avons créé un script shell réalisant le test de l'"émulation" choisie et renvoyant la bonne option. Ce script se nomme "`xeuca_crlf`" et se présente ainsi :

```
ARCH=`$CADP/com/arch -detailed`

case $ARCH in
        sun3 | sun4 | sun5 | iX86 )
                echo "auto" ;;
        win32-cygnus )
                echo "binary" ;;
        win32-mks )
                echo "crlf" ;;
        * )
                echo "`basename $0`: unknown architecture '$ARCH'"
                exit 1 ;;
esac
exit 0
```

L'appel dans le code TCL est réalisé comme suit :

```
fconfigure $Channel_ID -translation [exec sh xeuca_crlf]
```

4.3 Commandes et propriétés système réécrites

4.3.1 Mail

La gestion de la messagerie est radicalement différente sous WINDOWS NT par rapport à son homologue sous UNIX.

Sous UNIX, le fonctionnement de la messagerie est basé sur le lancement d'un processus `sendmail` associé à un fichier de configuration `sendmail.cf` qui répertorie l'ensemble des caractéristiques nécessaires à l'utilisation des protocoles SMTP [Pos82] et POP3 [MR96]. Ces deux protocoles permettent d'expédier et recevoir des messages électroniques. Ils sont mis en œuvre par l'utilisation de la commande `mail` disponible sur la ligne de commande.

Sous Windows NT, le paramétrage de l'envoi et la réception de messages est interne, donc spécifique, à chaque "client" Mail. Il n'existe pas de fonction pouvant être appelée sur la ligne de commande. Les possibilités de client Mail intégrées sont des applications telles que "Outlook Express" ou des fonctionnalités telles que la gestion Mapi (*Messaging Application Program Interface*).

Pour apporter une solution d'envoi de Mail, nous avons d'abord développé notre propre client Mail, puis nous sommes finalement orienté vers un produit existant sur le marché.

Développement d'un client SMTP : dans un premier temps, il a été choisi de développer un client Smtp, c'est-à-dire une application offrant un service de communication entre deux machines pour échanger des informations constituant un message électronique.

Smtp est le protocole standard le plus répandu pour la messagerie électronique dans le monde Internet, des réseaux et des communications mondiales. Il permet un échange structuré entre client et serveur.

Sous Unix comme sous Windows NT, le protocole Smtp est associé au port TCP 25. Les commandes reconnues sur ce port sont les suivantes :

HELO	EHLO	MAIL	RCPT	DATA
RSET	NOOP	QUIT	HELP	VRFY
EXPN	VERB	ETRN	DSN	

Pour réaliser l'envoi d'un mail, les commandes minimales à respecter sont :

- MAIL From:<exp> : pour définir l'adresse de l'expéditeur du message,
- RCPT To:<dest> : pour déterminer l'adresse du (ou des) destinataire(s),
- DATA : pour commencer le début du message à expédier,
- . : pour indiquer la fin du message,
- QUIT : pour clore l'échange.

Puisque la fonction Mail nécessaire à Cadp doit s'intégrer à Eucalyptus, il a été choisi de la développer en Tcl. Cela a donné le script suivant :

```
proc lire {canal} {
        # pour lire et ignorer les messages de retour sur le canal
        # lie' a' la socket ouverte sur le port 25
        gets $canal
}

proc envoyer {canal cmd} {
```

```
		# pour envoyer les commandes sur le canal de communication
		puts $canal $cmd
		flush $canal
	}

	proc envoyer_mail {serveur_SMTP dest exp sujet message} {
		set standard_SMTP_socket 25
		set socket [socket $serveur_SMTP $standard_SMTP_socket]
		envoyer $socket "MAIL From:<$exp>"
		lire $socket
		foreach destinataire $dest {
			envoyer $socket "RCPT To:<$destinataire>"
		}
		lire $socket
		envoyer $socket DATA
		# debut du message
		lire $socket
		puts $socket "From:  <$exp>"
		puts $socket "To:  <$dest>"
		puts $socket "Subject:  $sujet\n"
		foreach line [split $message \n] {
			puts $socket [join $line]
		}
		puts $socket .\nQUIT
		# fin du message
		lire $socket
		close $socket
	}
```

Dans ce script, l'envoi d'un mail est réalisé par l'appel à la procédure `envoyer_mail` qui prend comme paramètres l'adresse du serveur SMTP (`serveur_SMTP`), les adresses e-mail du destinataire (`dest`) et de l'expéditeur (`exp`) du mail, le sujet (`sujet`) et le message (`message`) envoyés. Cette procédure réalise l'ouverture d'une socket sur le port TCP/IP numéro 25 du serveur (port standard réservé au protocole SMTP, voir `/etc/services` sous UNIX). Ensuite, elle envoie des commandes (appel à la procédure `envoyer`) dans le langage du protocole pour déterminer quels sont l'expéditeur (`MAIL From:`), les destinataires (`RCPT To:`) et les données du message (`DATA`).

La lecture des résultats de ces commandes est réalisée par l'appel à la procédure `lire` qui se contente de vider la mémoire tampon tout en ignorant son contenu.

La fin des données est marquée par l'envoi d'un point suivi immédiatement d'un retour à la ligne. Dès lors, il suffit de quitter l'exécution du protocole SMTP en envoyant la commande appropriée. Ces deux dernières opérations sont réalisées par l'unique commande :

```
puts $socket .\nQUIT
```

Le type de communication présenté dans ce script fonctionne indifféremment et de façon identique sous UNIX et WINDOWS NT car l'implémentation du protocole SMTP est la même sur chacune des deux plate-formes.

Cependant, nous avons abandonné cette solution parce qu'elle réalisait une gestion trop sommaire des erreurs et pour éviter de maintenir des applications nécessitant des compétences pointues en protocoles et réseaux. Il a finalement été choisi de se procurer une application du marché pouvant être exécutée depuis la ligne de commande.

BLAT : la dernière option testée et finalement adoptée, pour fournir à CADP un client Mail, a été le logiciel libre (*freeware*) "`blat`".

Parmi les avantages principaux de "`blat`", on peut citer, entre autres :

– sa capacité à enregistrer dans la base de registres les paramètres nécessaires a l'envoi de messages (adresse du serveur SMTP, adresse de l'expéditeur),
– la possibilité d'être utilisé sur la ligne de commande à l'aide d'une syntaxe très simple.

Principalement, on associera les options suivantes à la commande `blat` :

– `blat -install` : permet de paramétrer (enregistrer dans la base de registres) le nom du serveur SMTP et l'adresse e-mail de l'expéditeur,
– `blat -profile` : renvoie, si "`blat`" a été correctement installé et configuré, le nom du serveur SMTP et l'adresse e-mail de l'expéditeur,
– `blat FILENAME -t DEST -s SUBJECT` : permet d'envoyer le fichier `FILENAME` à `DEST` en indiquant le sujet `SUBJECT`.

Il a été choisi de créer un script shell permettant de gérer l'envoi de mail quelque soit l'architecture (UNIX ou WINDOWS NT) sur laquelle on se trouve. Ce script (`xeuca_mail`) est utilisé dans quatre contextes différents :

– `xeuca_mail -server` qui renvoie :
  * soit la chaîne `"xeuca_mail_automatic"` si l'on est sous UNIX et que le *daemon* `sendmail` fonctionne,
  * soit la chaîne `"xeuca_mail_failed"` si l'on est sous UNIX et que le *daemon* `sendmail` n'est pas lancé,
  * soit la chaîne `"xeuca_mail_unknown"` si l'on est sous WINDOWS et que l'on n'a pas lancé "`blat -install`" auparavant,
  * soit le nom du serveur SMTP si l'on est sous WINDOWS et que `blat` a déjà été configuré.

– `xeuca_mail -from` qui renvoie :
  * soit la chaîne `"xeuca_mail_automatic"` si l'on est sous UNIX et que le *daemon* `sendmail` fonctionne,
  * soit la chaîne `"xeuca_mail_failed"` si l'on est sous UNIX et que le *daemon* `sendmail` n'est pas lancé,
  * soit la chaîne `"xeuca_mail_unknown"` si l'on est sous WINDOWS et que l'on n'a pas lancé "`blat -install`" auparavant,
  * soit l'adresse e-mail de l'expéditeur si l'on est sous WINDOWS et que `blat` a déjà été configuré.
– `xeuca_mail -send FICHIER DESTINATAIRE` qui permet d'envoyer `FICHIER` à `DESTINATAIRE` sous UNIX.
– `xeuca_mail -send SERVEUR_SMTP EXPEDITEUR FICHIER DESTINATAIRE` qui permet d'envoyer `FICHIER` à `DESTINATAIRE` sous WINDOWS en utilisant `SERVEUR_SMTP` avec comme adresse de retour `EXPEDITEUR`.

4.3.2 Man

L'interface EUCALYPTUS propose une aide en ligne matérialisée par des appels à la commande UNIX "`man`", pour :

- les outils de CADP (ALDEBARAN, CAESAR, LOTOS, APERO, etc),
- les formats de fichiers traités par CADP (Extended LOTOS, programmes C, expressions de composition, etc),
- les fonctions proposées dans les menus (Edit, Compile, Reduce, etc).

Dans les environnements UNIX fonctionnant sous WINDOWS étudiés, la solution CYGWIN est la seule à ne pas proposer de commande `man`.

La solution MKS TOOLKIT est incapable de gérer des pages de manuel au format NROFF pour les commandes autres que celles qu'elle met en œuvre dans son répertoire `/bin`. Néanmoins, MKS TOOLKIT propose une commande `man` associée à une variable `MANPATH` permettant de visualiser des pages de manuel préalablement converties au format `plaintext` (un format de texte complété de caractères interprétables par un terminal : gras, sauts de pages et de lignes, etc).

La variable `MANPATH` contient des chemins de répertoires dans lesquels se trouvent les pages réparties dans une arborescence fixée :

```
man --+
      |
      +--> cat.1--+
```

```
|               |
|               +- commande1.1
|               +- commande2.1
|               + etc ...
|
+--> cat.2--+
|               |
|               +- commande1.2
|               +- commande2.2
|               + etc ...
|
+ etc ...
```

INTERIX propose une solution identique à celle de MKS TOOLKIT.

La commande `man` proposée par UWIN est un script shell qui réalise une conversion d'un fichier au format NROFF vers le format HTML, puis lance un navigateur Internet pour afficher le résultat de la conversion. Pour faire fonctionner cette commande, il suffit de positionner la variable `MANPATH` et de l'exporter. Le répertoire indiqué doit contenir un répertoire `catl` qui contient lui-même les pages (suffixées `.l`) au format NROFF.

Pour consulter les pages de manuel de la boîte à outils CADP, il a été choisi de convertir toutes les pages au format `plaintext` afin de permettre leur consultation par une simple commande `cat`.

Pour réaliser cette conversion, il suffit d'utiliser la commande `nroff` en respectant la forme de syntaxe suivante :

```
nroff -man <MANDIR>/manX/commande.X > <MANDIR>/catX/commande.X
```

où `MANDIR` est un répertoire contenant des pages de manuel.

Pour afficher la page obtenue, on pourra opter également pour la commande "`less -s`" qui réalise une présentation élégante du résultat à l'écran.

4.3.3 Visualisation d'un fichier Postscript

EUCALYPTUS permet de consulter un fichier au format Postscript (`.ps`) à l'aide de l'option "`show`" du menu des commandes UNIX. Sous SUN OS, cette option est assurée par le programme "`ghostview`".

Sous WINDOWS NT, il a été choisi d'assurer cette fonction en utilisant un logiciel libre du marché : "`gsview32.exe`" [23]

[23] Disponible à l'adresse Internet : `http://www.cs.wisc.edu/~ghost/aladdin/get550.html`

La dernière version à ce jour est la version 5.5. Pour assurer le fonctionnement de ce programme, il suffit de télécharger le fichier "`gsv27550.exe`", puis d'ajouter le chemin d'accès à ce programme dans la variable d'environnement `PATH` en tapant les deux commandes suivantes :

```
PATH=$PATH:/gstools/gsview
export PATH
```

Un script shell (`xeuca_ps`) a été écrit pour automatiser l'emploi de "`ghostview`" ou "`gsview32.exe`" selon la plate-forme utilisée. Les lignes suivantes décrivent la syntaxe utilisée pour différencier le type d'architecture sur laquelle la commande est exécutée :

```
case $ARCH in
        sun3 | sun4 | iX86 | sun5 )
                CADP_PS_VIEWER="ghostview" ;;
        win32 )
                CADP_PS_VIEWER="gsview32.exe" ;;
        * )
                echo "$COMMAND: unknown architecture $ARCH"
                exit 1 ;;
esac

eval $CADP_PS_VIEWER 2>/dev/null
```

4.3.4 Propriétés d'un fichier

Eucalyptus permet de consulter les propriétés d'un fichier à l'aide de l'option "`properties`" du menu des commandes Unix. Cette commande renvoie les informations suivantes :

```
Filename:          arch
Type:              executable shell script
Size:              1169 bytes
Mode:              -rw-r-r-
Owner:             mazzilli
Group:             vasy_euc
Last modified:     Apr 1 12:28
Last accessed:     Apr 16 15:03
```

Cette fonction est basée sur le résultat de la commande "`ls -lg`" découpé grâce à la commande "`cut -f`" qui permet de donner des valeurs à chacune des rubriques affichées, à l'exception de la rubrique `Type`. En effet, cette rubrique est renseignée par le résultat de la commande shell "`file`".

Le problème réside dans le fait que cette commande n'existe pas dans l'environnement Cygwin. Il a donc été choisi de la redévelopper en langage C par l'accès à la base de registre. Les lignes suivantes sont extraites de ce programme afin de montrer l'Api d'accès aux registres Win32 :

```
extension = strstr(argv[1],".");
sprintf(Buf,"SOFTWARE\\Classes\\%s",extension);

lres= RegOpenKeyEx (HKEY_LOCAL_MACHINE, Buf, 0, KEY_READ,&hKey);

if (lres==ERROR_SUCCESS)
{
        DWORD dwType;
        DWORD dwBytes = 64;


        RegQueryValueEx( hKey,"",0,&dwType, szValue, &dwBytes);
        RegCloseKey(hKey) ;
}

sprintf(Buf,"SOFTWARE\\Classes\\%s",szValue);

lres= RegOpenKeyEx(HKEY_LOCAL_MACHINE,Buf,0, KEY_ALL_ACCESS,&hKey) ;

if (lres==ERROR_SUCCESS)
{
        DWORD dwType;
        DWORD dwBytes = 64;

        RegQueryValueEx( hKey,"",0,&dwType, szValue, &dwBytes);
        printf("Value : %s\n",szValue) ;
        RegCloseKey(hKey) ;
}
```

Ce programme prend en entrée un nom de fichier complet (chemin et extension renseignés) et en extrait le suffixe (partie située à droite du point) à l'aide de la commande :

```
str= strstr(argv[1],".");
```

Il constitue ensuite la clé d'accès à la base de registres en concaténant la chaîne :

```
SOFTWARE\\Classes
```

à l'extension précédemment enregistrée.

A l'aide de cette clé, il accède en lecture au chapitre `HKEY_LOCAL_MACHINE` des registres WIN32. Si cet accès aboutit, il lit la valeur de la clé (le nom interne du type de document associé à l'extension) et la stocke dans la variable `szValue` grâce à la commande :

```
RegQueryValueEx( hKey,"",0,&dwType, szValue, &dwBytes);
```

La deuxième étape consiste à concaténer la chaîne de caractères

```
SOFTWARE\\Classes\\
```

à la variable `szValue` afin de récupérer le type de document concerné et l'afficher.

Pour l'extension `.doc` par exemple, la clé :

```
HKEY_LOCAL_MACHINE\\SOFTWARE\\Classes\\doc
```

renvoie le code interne `Wordpad.Document.1`.

Puis, la clé

```
HKEY_LOCAL_MACHINE\\SOFTWARE\\Classes\\Wordpad.Document.1
```

renvoie la chaîne "`Document Wordpad`".

C'est cette chaîne de caractères qui est ensuite affichée à l'écran.

4.3.5 Editeur de texte

Cette fonction étant assurée "en dur" dans EUCALYPTUS, il a simplement été choisi de sortir la fonction afin de l'isoler dans un script shell : `xeuca_edit`. L'utilisateur doit simplement positionner une variable d'environnement `$EDITOR`. A défaut, elle sera positionnée à :

- `vi` sous UNIX
- `write.exe` sous WIN32

4.3.6 Impression

Cette fonction étant assurée "en dur" dans EUCALYPTUS, il a simplement été choisi de sortir la fonction afin de l'isoler dans un script shell : `xeuca_print`. Ce script réalise un appel à la commande :

- `lpr` sous UNIX
- `write /p` sous WIN32

4.3.7 Navigateur Internet

Cette fonction étant assurée "en dur" dans EUCALYPTUS, il a simplement été choisi de sortir la fonction afin de l'isoler dans un script shell : `xeuca_web`. L'utilisateur doit simplement positionner une variable d'environnement `$NAVIGATOR`. A défaut, elle sera positionnée à :

- `netscape` sous UNIX
- `iexplore.exe` sous WIN32

4.3.8 Ouverture d'un interpréteur shell en mode console

Cette fonction étant assurée "en dur" dans EUCALYPTUS, il a simplement été choisi de sortir la fonction afin de l'isoler dans un script shell : `xeuca_shell`. Ce script réalise un appel à la commande :

- `xterm` sous UNIX
- `cmd.exe /c start sh` sous WIN32

4.4 Installator : suppression du module Expect

Après avoir recensé les modifications apportées dans le cadre des commandes utilisateur, penchons nous sur le cas du protocole FTP à travers l'interface graphique INSTALLATOR.

Comme il en a été question précédemment, INSTALLATOR utilise EXPECT pour réaliser la communication avec le protocole FTP afin de transférer les fichiers nécessaires à l'installation de CADP.

Cette communication nous a d'abord conduit à étudier le protocole FTP puis le mode de fonctionnement d'EXPECT sous WIN32, pour en déduire qu'il fallait trouver une autre solution et, enfin, opter pour un client FTP (en logiciel libre) écrit en TCL.

Ce dernier a été intégré dans l'interface graphique INSTALLATOR comme le décrit la présente section.

4.4.1 Protocole FTP

FTP [PR85] est un outil standard utilisant TCP/IP et permettant un transfert de fichiers (textes et binaires) dans les deux sens, entre une machine locale et une machine distante. Un transfert de fichier par FTP passe par une phase préliminaire d'accès à la machine distante (nom d'utilisateur et mot de passe doivent être fournis pour établir la connexion et initialiser le transfert proprement dit). Comme pour les autres utilitaires de type UNIX, FTP fonctionne sur le modèle client-serveur qui se traduit, dans ce cas, par la communication entre un processus client `ftp` et un processus serveur `ftpd`.

Format de la commande :

```
ftp [options] [hostname]
```

Le détail des options de cette commande ne sera pas présenté. En revanche, il sera question des "sous-commandes" du client FTP qui permettent de transférer des fichiers depuis ou vers la machine distante (`hostname`). Les principales "sous-commandes" du client FTP utilisées dans CADP sont les suivantes [TT91] :

Commandes de connexion/déconnexion

`open hostname`	connexion avec la machine distante
`close`	déconnexion sans sortir de `ftp`
`quit`	déconnexion avec sortie de `ftp`

Opérations sur les fichiers

`delete rfile`	suppression du fichier distant `rfile`
`mdelete rfile1, ... rfilen`	suppression de plusieurs fichiers distants
`get rfile [lfile]`	transfert du fichier distant `rfile` sur la machine locale sous le nom `lfile` ou le même nom si `lfile` n'est pas spécifié
`mget rfile1,...rfilen`	réception de plusieurs fichiers du site distant
`put lfile [rfile]`	transfert du fichier local `lfile` vers la machine distante sous le nom `rfile` ou le même nom si `rfile` n'est pas spécifié
`mput lfile1,...lfilen`	transfert de plusieurs fichiers du répertoire local courant vers le site distant

Opérations sur les répertoires

`dir`	listing au format long du répertoire distant
`lcd [ldir]`	changement de répertoire en local
`cd [rdir]`	changement de répertoire distant
`ls`	listing au format court du répertoire distant
`mkdir [rdir]`	création du répertoire distant `rdir`
`rmdir [rdir]`	suppression du répertoire distant `rdir`
`pwd`	nom du répertoire courant distant `rdir`

Options de transfert

`ascii`	fichier de type ASCII (option par défaut)
`binaire`	fichier de type binaire
`status`	état courant du transfert

`ftp` lancé sans argument affiche le prompt `ftp>`. Au prompt, la commande `help` permet d'avoir une aide sur les commandes acceptées par le client FTP.

Enfin, `ftp` est la commande qui sert d'interface au protocole FTP dont les commandes, de plus bas niveau, sont les suivantes [PR85] :

- Commandes de contrôle d'accès :

  ```
  ACCT    CDUP    CWD     PASS    QUIT    REIN    SMNT    USER
  ```

- Commandes de paramétrage du transfert :

  ```
  MODE    PASV    PORT    STRU    TYPE
  ```

- Commandes de service FTP :

  ```
  ABOR    ALLO    APPE    DELE    HELP    LIST    MKD
  NLST    NOOP    PWD     REST    RETR    RMD     RNFR
  RNTO    SITE    STAT    STOR    STOU    SYST
  ```

Ces commandes garantissent un fonctionnement universel de chaque client FTP sur toutes les plate-formes.

4.4.2 Expect sous WINDOWS NT

INSTALLATOR assiste l'utilisateur à l'installation de CADP en réalisant des accès et des transferts depuis le site FTP du projet VASY. Actuellement ce site FTP est placé sur une machine

UNIX, et cela ne devrait pas changer dans l'avenir. Il n'est donc, de ce fait, envisagé aucune modification liée au présent projet de portage.

Le problème étant résolu côté serveur, il était indispensable de vérifier que les processus client `ftp` avaient un fonctionnement proche selon les différentes architectures. Rapidement, les différentes évaluations ont été satisfaisantes car FTP est un service standard et universel fonctionnant sur toutes les machines UNIX et non-UNIX utilisant TCP/IP.

Néanmoins, parce que les clients FTP présents sous UNIX et WIN32 présentent des légères différences ou parce que l'implémentation d'EXPECT varie d'une plate-forme à l'autre[24], des modifications ont été nécessaires dans le code EXPECT afin qu'il puisse fonctionner sous WINDOWS NT :

- A la connexion : sous UNIX, le client FTP affiche le prompt "`Name (host:user):`" alors que sous WIN32, la connexion peut se réaliser au prompt "`User (host:user):`". Le fait qu'EXPECT fonctionne directement sur la valeur des messages envoyés par le protocole dont il a le contrôle rend les programmes développés beaucoup trop dépendants de l'architecture. Cependant, il a fallu tester le type de plate-forme sur laquelle INSTALLATOR est exécuté pour positionner le prompt FTP adéquat.
- Au lancement du processus `ftp` : dans INSTALLATOR, l'ouverture du processus interactif `ftp` est réalisé à l'aide de la commande :

  ```
  spawn ftp ftp.inrialpes.fr
  ```

  Le processus `ftp` lancé sous UNIX vide son tampon après chaque écriture sur sa sortie standard. Cela n'étant pas le cas sous WINDOWS NT, il est indispensable de forcer ce mécanisme à l'aide de l'option `-pipes` de la façon suivante :

  ```
  spawn -pipes ftp ftp.inrialpes.fr
  ```

  Cette commande fonctionne aussi sous UNIX et a même pour effet d'accélérer les traitements interactifs.
- A l'envoi d'une chaîne vers le protocole FTP : chaque chaîne de caractères envoyée vers le client FTP doit être ponctuée d'un retour chariot. Dans INSTALLATOR, cela se traduit par :

  ```
  send string\n
  ```

  Sous WIN32, cette commande n'a aucun effet sur le client FTP. Il est nécessaire d'utiliser plutôt la syntaxe suivante :

  ```
  send string\r
  ```

[24] Expect sous NT, voir : `ftp://bmrc.berkeley.edu/pub/winnt/tcltk/expect/README.NT`

> Cette syntaxe est celle qu'il faut retenir, y compris sur Unix car seul le retour chariot garantit un envoi fiable des commandes vers le client Ftp. La raison pour laquelle le "`\n`" fonctionnait sous Unix est que la couche du pseudo-terminal traduit les "`\n`" en "`\r`", contrairement au terminal Win32.

Sous Windows NT, la solution se complique car Expect a été porté (ExpectNT) mais le compilateur Ice (voir section 4.4.5) ne sait pas compiler du code Tcl qui inclut du code ExpectNT.

De ce fait, il fallait se résoudre à fournir le code Tcl d'Installator aux utilisateurs, à moins que nous nous procurions ou redéveloppions un client Ftp basé sur les sockets et indépendant d'Expect.

4.4.3 Utilisation de Ftp_Lib à la place d'Expect

La solution finalement adoptée pour l'implémentation du protocole Ftp dans Installator a été `ftp_lib.tcl`, un client Ftp en logiciel libre disponible à l'adresse Internet suivante : `http://home.t-online.de/home/Steffen.Traeger/tindexe.htm`.

Etant écrit en Tcl, il suffit d'insérer son code dans celui d'Installator et d'utiliser les commandes décrites dans le tableau suivant :

Syntaxe	Description
Open <server> <user> <passwd>	Ouverture de la session Ftp vers un serveur Ftp pour un utilisateur et son mot de passe associé
Close	Termine la session Ftp en cours
Cd <directory>	Changement de répertoire sur le serveur
Pwd	Renvoie le chemin du répertoire courant sur le serveur
Type <?ascii|binary?>	Etablit le type de transfert
List <?directory?>	Contenu du répertoire courant sur le serveur
NList <?directory?>	Idem List mais liste abrégée
FileSize <file>	Renvoie la taille du fichier spécifié sur le serveur
ModTime <file>	Renvoie la dernière date de modification du fichier spécifié sur le serveur
Delete <file>	Supprime le fichier spécifié sur le serveur
Rename <from> <to>	Renomme le fichier spécifié sur le serveur
Put <local> <?remote?>	Transfert un fichier local vers le serveur

Syntaxe	Description
Append <local> <?remote?>	Concatène un fichier local à un fichier présent sur le serveur
Get <remote> <?local?>	Transfert un fichier du serveur vers le répertoire courant local
Reget <remote> <?local?>	Permet de ne transférer que la partie manquante d'un fichier depuis le serveur vers le répertoire courant local. Très utilisé lors du transfert interrompu d'un fichier volumineux
Newer <remote> <?local?>	Idem Get mais le transfert n'a lieu que si le fichier sur le serveur est plus récent que celui présent sur le répertoire courant local
MkDir <directory>	Permet de créer un répertoire sur le serveur
RmDir <directory>	Permet de supprimer un répertoire sur le serveur

On peut également choisir de capturer tous les messages échangés avec le protocole en modifiant la procédure DisplayMsg pour envoyer tous les messages de sortie vers un fichier de son choix plutôt que la sortie standard, de la façon suivante :

```
proc DisplayMsg {msg {state ""}} {
variable VERBOSE

        set FD [open Monfichier a+]

        switch $state {
          data          {if {$VERBOSE} {puts $FD $msg}}
          control       {if {$VERBOSE} {puts $FD $msg}}
          error         {puts $FD "ERROR: $msg"}
          default       {if {$VERBOSE} {puts $FD $msg}}
        }

        close $FD
}
```

Cette solution de client Ftp écrit en Tcl présente l'avantage de pouvoir être compilée comme cela était déjà le cas sur l'ancienne version de Installator. Par ailleurs, on réalise du même coup la suppression du module Expect, ce qui représente un outil de moins à maintenir dans Cadp.

4.4.4 Intégration de Ftp_Lib dans Installator

La suppression du module EXPECT et l'intégration de la bibliothèque `ftp_lib.tcl` a provoqué quelques bouleversements dans INSTALLATOR, notamment en ce qui concerne les points suivants :

- Exécution en tâche de fond : le module EXPECT mettait en œuvre une commande `expect_background` permettant de réaliser un traitement interactif en tâche de fond, utilisé dans INSTALLATOR pour dessiner de façon parallèle une barre de progression du transfert en cours.

  Dans FTP_LIB, il n'existe pas un telle fonction. En revanche, la commande "`Open`" implémentée dans FTP_LIB offre une option `-progress` permettant d'associer une procédure qui s'exécutera à chaque fois qu'un bloc vient d'être transféré. Cette caractéristique convient tout à fait au mécanisme de barre de progression évoqué ci-dessus, à la différence près que cette barre ne sera recalculée qu'à la fin de chaque bloc transféré et non de façon permanente pendant le transfert, comme cela était le cas avec EXPECT.

- Lenteur du transfert : la réception d'un fichier depuis le serveur est plus lente avec FTP_LIB qu'avec la solution EXPECT. Cela provient du fait qu'il y a dans `ftp_lib.tcl` une gestion plus rigoureuse de la synchronisation par la manipulation de délais de garde (*timeout*). Cette rigueur prévient les dysfonctionnements lors des sessions FTP mais a aussi pour effet de dégrader les temps de réponse.

  Un moyen d'atténuer cette dégradation est de faire varier la taille de bloc afin de réduire le nombre de synchronisations réalisées. En effet, de nombreuses synchronisations sont réalisées à la fin de chaque bloc transféré. Par défaut, la taille de bloc utilisée est de 4096 octets. La commande `Open` offre la possibilité de modifier cette valeur grâce à son option `-blocksize`. Ainsi, la nouvelle taille de bloc choisie pour le transfert de fichiers implémenté dans INSTALLATOR est de 40960 octets grâce à la commande :

  ```
  Open ftp.inrialpes.fr anonymous user@ -progress Barre \
          -blocksize 40960
  ```

  Note : l'augmentation de la taille de bloc provoque une fréquence plus faible de redessinement de la barre de progression de transfert. Il s'agit donc de trouver le meilleur compromis entre vitesse de transfert et qualité d'affichage de la barre de progression.

- Transfert de fichiers : lors du transfert du dernier bloc, le port CONTROLE du client FTP envoie un message "`get_close`" pour indiquer qu'il va fermer le port de communication après l'acquittement de transfert du dernier bloc.

  Or, la commande "`update`" présente dans le script `installator.tcl` provoque l'arrivée prioritaire de l'événement de clôture du transfert. Cela a pour effet de fermer prématurément le port de communication entre le client et le serveur FTP. INSTALLATOR se retrouve dans une impasse où il pense être en cours de transfert du dernier bloc de façon infinie plaçant ainsi le client et le serveur en attente mutuelle (*deadlock*).

Ce problème est illustré sur la figure 4.1.

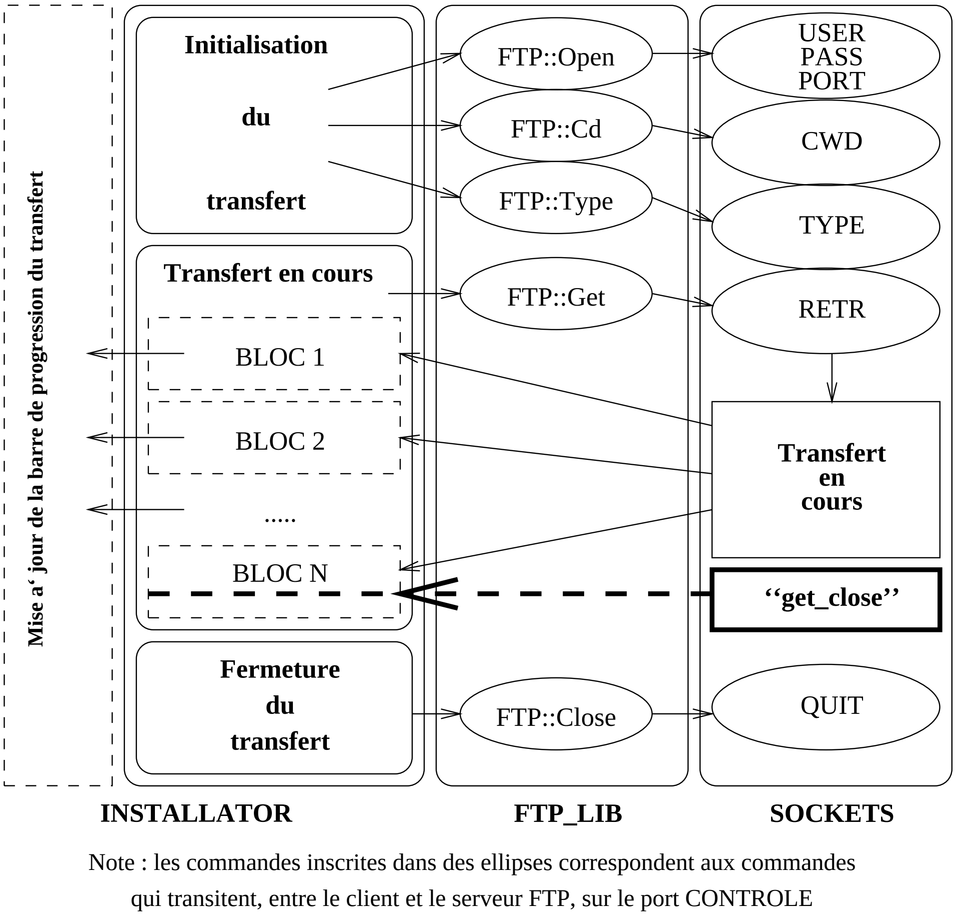

Figure 4.1: Problème du "update" dans INSTALLATOR

Pour y remédier, nous avons simplement ajouté l'option `idletasks` à la commande `update` de la façon suivante :

```
update idletasks
```

Ainsi, ne sont pris en compte que les événements liés au redessinement de l'écran tout en ignorant la priorité des événements liés aux descripteurs de fichier.

4.4.5 Génération de Byte-Code Tcl

Afin de préserver la confidentialité du code TCL écrit, INSTALLATOR était jusqu'à présent fourni dans sa version binaire générée grâce à la version 1.3 du compilateur ICE distribué par la société Icem Cfd Engineering[25]. Ce compilateur permet de prendre du code TCL/TK/TIX/EXPECT en entrée afin de produire du langage C, qui peut lui-meme être compilé afin de générer un binaire exécutable. Le problème réside dans le fait que le code source généré n'est pas portable.

Pour règler ce type de problème, nous avons réalisé des tests autour des solutions suivantes :

- La version 2.0 de ICE : cette version permet de produire du Byte-Code TCL en sortie. Ce Byte-Code est un fichier binaire pouvant être interprété par n'importe quelle architecture disposant d'une machine virtuelle TCL. Cette caractéristique est fort intéressante dans le cadre du présent projet.

  La version 2.0 de ICE n'a, à ce jour, pas intégré la version WINDOWS NT d'EXPECT. Cependant, cela ne représente pas un problème puisqu'il a été choisi de se passer définitivement du module EXPECT.

  Pour utiliser la version 2.0 de ICE, il suffit d'avoir téléchargé ICE depuis le site FTP `ftp.dnai.com` et positionné la variable `TCL_ROOT` au chemin du compilateur.

  – Pour continuer à produire du code C, la commande est la suivante :

    ```
    $TCL_ROOT/bin/tcl_compiler -ltk -c -buildshell myprog.tcl
    ```

    Cette commande produit des fichiers `myprog.c` et `myprog.out` distribuables librement.

  – Pour générer du Byte-Code, la commande est la suivante :

    ```
    $TCL_ROOT/bin/tcl_compiler -ltk -buildshell myprog.tcl
    ```

    Cette commande produit un fichier `myprog.ptcl` nécessitant le programme `wish` livré avec le compilateur.
    `myprog.ptcl` et `wish` peuvent être distribués librement.

  Le problème de cette solution est qu'il est nécessaire de payer une licence ICE par architecture pour laquelle on souhaite distribuer du code compilé.

- Le compilateur TCLPRO[26] de Scriptics : cette solution est un environnement de programmation TCL contenant, entre autres modules, un programme permettant de produire du Byte-Code TCL (bien évidemment différent de celui produit par le compilateur ICE). Pour ce faire, la commande est la suivante :

  ```
  procomp myprog.tcl
  ```

  Cette commande produit un fichier `myprog.tbc` interprétable par n'importe quelle version de WISH supérieure à la version 8.0.3, à condition de disposer du module `Tbcload` ou du programme `prowish` livrés avec TCLPRO.

[25] voir l'adresse Internet http://www.icemcfd.com/
[26] `http://www.scriptics.com/support/faq.html#S7`

Les fichiers et programmes `myprog.tbc`, `Tbcload` et `prowish` peuvent être distribués librement après avoir acquitté les droits d'une licence TclPro (1200 dollars) par architecture cible.

- Brouilleur de code Tcl : les solutions de production de Byte-Code vues ci-dessus ont finalement été abandonnées, du fait de la dépendance qu'elles génèrent, au profit d'un utilitaire développé au sein de l'équipe Vasy (appelé `conspirator`). Le but de ce programme, réalisé à l'aide des langages C et Lex, est de remplacer tous les identificateurs présents dans le code Tcl par des chaînes de caractères générées automatiquement, de façon à brouiller sensiblement la lecture du code source.

4.5 Autres améliorations dans l'IHM

Exceptées les modifications entraînées par les points abordés précédemment et qui contribuent à l'amélioration globale de l'interface homme-machine Eucalyptus, diverses corrections ont apporté un soutien supplémentaire à l'ergonomie des produits ou à l'efficacité du code.

4.5.1 Gestion des fenêtres

- Dans Eucalyptus, lorsqu'on essayait d'ouvrir une fenêtre déjà ouverte, un message apparaissait pour indiquer cet état de fait : *"This window is already opened"*. Cela obligeait à jongler avec les fenêtres ouvertes pour retrouver celle dont on avait besoin. Dorénavant, si la fenêtre est déjà ouverte, elle est placée au premier plan grâce à la commande Tk :

```
raise $window
```

- La fenêtre principale d'Eucalyptus est retaillable, c'est-à-dire que l'on peut augmenter ou réduire sa taille et stocker cette dernière pour la prochaine utilisation de l'interface.
- De la même manière, on peut enregistrer la position de la fenêtre de façon à travailler dans une configuration identique lors de chaque utilisation d'Eucalyptus. Ces éléments sont stockés dans le fichier `~/.xeucarc`.
- Chaque fenêtre ouverte dans Eucalyptus dispose de deux boutons `Cancel` et `OK` permettant tous les deux de fermer la fenêtre active après avoir réalisé un certain nombre de traitements. Selon le gestionnaire de fenêtres utilisé sous Unix, l'utilisateur peut disposer d'une autre possibilité de fermeture de la fenêtre active que le script `eucalyptus.tcl` ne "voit" pas. Si l'utilisateur ferme la fenêtre en cliquant sur le bouton implémenté par le gestionnaire de fenêtres, cela engendre le fait que l'interface Eucalyptus, n'ayant pas été avertie de cet événement, affichera le message *"This window is already opened"* à la prochaine tentative d'ouverture de la fenêtre concernée. Pour remédier, il a été nécessaire de capter l'événement de fermeture à l'aide de la commande :

```
bind $Current_Frame <Destroy> {
        ... code Cancel ...
}
```

Cette séquence de commandes TCL permet de capturer l'événement `WM_DELETE` émis par le gestionnaire de fenêtres afin de lui associer le même code que celui du bouton `Cancel`.

4.5.2 Optimisation du code

Dans l'environnement TCLPRO, proposé par la société Scriptics, vu précédemment, il existe un analyseur de scripts TCL permettant de trouver des erreurs de syntaxe, d'une part, et de réaliser quelques vérifications statiques à propos de l'usage d'identificateurs par l'utilisateur, d'autre part.

Bien que cet outil (basé sur la commande `procheck`) permette de mettre en évidence des fragments de "code mort", il ne signale pas les noms des procédures et variables non utilisées qui représentent souvent un lourd handicap dans la maintenance de programmes interprétés.

Le brouilleur de code implémenté au sein de l'équipe VASY dont il a été question un peu plus haut, doit permettre de combler la lacune de TCLPRO.

4.6 Portage des autres outils de la bibliothèque CADP

4.6.1 Code C

Cette section a pour objet d'inventorier les points à prendre en compte dans le cadre du portage de programmes écrits en langage C. Pour cela, il sera pris comme support les programmes de la boîte à outils CADP pour lesquels des modifications concrètes ont été nécessaires. Ces aménagements pourront, dans l'avenir, servir de base à la rédaction d'un document spécifiant les normes de programmation au sein de l'équipe VASY.

Par ailleurs, afin de s'assurer de la stricte conformité du code C avec la norme ANSI, il est conseillé de compiler les programmes sources en indiquant l'option ANSI. Pour le compilateur C de SUN OS, la commande à utiliser est la suivante :

```
cc -Xc ...
```

Les lignes qui suivent ont pour but de présenter les modifications apportées dans les principaux programmes de CADP.

- `othello`

  CADP dispose d'une version du fameux jeu de réflexion OTHELLO. Ce programme a été écrit en langage Pascal et traduit en C par l'utilitaire "`p2c`". La conversion Pascal vers C de ce programme a initialement produit un code source ne répondant pas à la norme ANSI. Le simple ajout de l'option `-ansi` à la ligne de commande `p2c` a produit un code source C "propre" et a permis au compilateur croisé de construire un binaire fonctionnel.

- `hostinfo`

  Cet utilitaire destiné à recueillir les informations systèmes nécessaires à la constitution d'une nouvelle licence ou la modification de la licence actuelle d'utilisation de CADP a été fortement remanié dans la mesure où il faisait appel à des fonctions système ou réseau inconnues de la bibliothèque MINGW32 telles que les commandes `gethostname()` (dont le rôle est de donner le nom de hôte courant) et `sysinfo()` (dont le rôle est de récupérer des informations relatives au système d'exploitation).

  Pour rendre la commande `gethostname()` disponible, il a été nécessaire de lancer la gestion "WSA" (*Windows Socket* API). Cela est réalisé par la suite de commandes C suivantes :

```
#include <winsock.h>

WORD wVersionRequested;
WSADATA wsaData;
int err;

wVersionRequested = MAKEWORD( 1, 1 );
err = WSAStartup( wVersionRequested, &wsaData );
if ( err != 0 ) {
        /* Winsock DLL introuvable */
        WSACleanup();
        HOSTNAME[0]='\0' ;
        return 1 ;
}
...

if ( LOBYTE( wsaData.wVersion ) != 2 ||
        HIBYTE( wsaData.wVersion ) != 0 ) {
                /* Winsock DLL introuvable */
                WSACleanup( );
                HOSTNAME[0]='\0' ;
                return 2 ;
}

gethostname (HOSTNAME, 4096);
```

Dans les lignes qui précèdent, la commande `WSAStartup`, permettant de lancer la version 1.1 de l'API de gestion de socket WINDOWS NT, renvoie un code d'erreur `err`. Si ce code est différent de 0, on ferme proprement la gestion de sockets à l'aide de la commande `WSACleanup`. Puis, on vérifie que la version de DLL présente correspond bien à la version 1.1 (ou supérieure). En effet, la version renvoyée par `WSAStartup` est la version demandée si la version présente sur la machine est égale ou supérieure à la version demandée. Si tout s'est bien déroulé, on peut faire appel à la commande `gethostname` pour charger `HOSTNAME` à la valeur du nom du hôte courant.

La commande `sysinfo()` est utilisée pour obtenir le nom de domaine d'appartenance du hôte courant. Sous WINDOWS NT, cette opération sera réalisée à l'aide d'un accès à la base de registres, de la façon suivante :

```
#include <winreg.h>

LONG lres ;
HKEY hKey ;

lres = RegOpenKeyEx(HKEY_LOCAL_MACHINE,
        "SYSTEM\\CurrentControlSet\\Services\\Tcpip\\Parameters",
        0, KEY_READ,&hKey) ;
if (lres==ERROR_SUCCESS)
{
        DWORD dwType ;
        DWORD dwBytes = 64;

        lres = RegQueryValueEx(hKey,"Domain",0,&dwType, DOMAINNAME,
               &dwBytes);
        RegCloseKey(hKey) ;
}
else
{
        DOMAINNAME[0] = '\0' ;
}
```

Le nom de domaine (`DOMAINNAME`) est récupéré grâce à la base de registres en lisant la valeur `Domain` de la clé :

```
HKEY_LOCAL_MACHINE\\SYSTEM\\CurrentControlSet\\Services\\Tcpip\\Parameters
```

Note : à l'édition de liens, ne pas oublier la bibliothèque `libwsock32.a`.

- `sxppp` est un programme, appartenant à l'outil `caesar.indent`, qui compte le nombre de caractères envoyés vers `stdout`. Comme il l'a été évoqué à différentes reprises dans ce document, WINDOWS représente la fin de ligne par deux caractères ("`\r\n`"). Cela a conduit à remplacer les lignes :

```
putc (NEW_LINE, stdout);
sxppvariables.char_count++
```

  par les lignes suivantes :

```
#if defined _WIN32
#define NEW_LINE_BYTES        2
#else
#define NEW_LINE_BYTES        1
#endif
...
putc (NEW_LINE, stdout);
sxppvariables.char_count+=NEW_LINE_BYTES;
```

- `xtl_expand` et `xtl_lib`

  Dans ces programmes, seule la prise en compte de la nouvelle architecture WIN32 a été réalisée afin de "satisfaire" le pré-processeur du C :

```
#if   defined (ARCHITECTURE_SUN_3)
           code Sun OS 3.x
#elif defined (ARCHITECTURE_SUN_4)
           code Sun OS 4.x
#elif defined (ARCHITECTURE_SUN_5)
           code Sun OS 5.x
#elif defined (ARCHITECTURE_PC_LINUX)
           code Linux
#elif defined (ARCHITECTURE_PC_WIN32)
           section vide
#else
           indiquer ici une fonction inexistante afin de
           "planter" le pre'-processeur
#endif
```

  Quelques programmes, comme celui-ci, sont construits sous la forme indiquée ci-dessus et nécessitent de ce fait l'option de définition de macro adéquate au moment de la compilation :

```
cc -DARCHITECTURE_XXXXXX ...
```

- `mcl_expand` est un outil permettant de produire un système d'équations modales à partir d'une formule de logique temporelle fournie en entrée. Dans ce programme, les appels aux commandes suivantes (prototypées dans le fichier `strings.h`) :

  ```
  char *index(const char *s, int c);
  char *rindex(const char *s, int c);
  ```

  ont respectivement été remplacés par les commandes strictement équivalentes suivantes (prototypées dans le fichier `string.h`) :

  ```
  char *strchr(const char *s, int c);
  char *strrchr(const char *s, int c);
  ```

  Le remplacement de ces fonctions réalisant des opérations sur des chaînes de caractères a été réalisé car Mingw32 n'implémente pas les fonctions prototypées dans le fichier `strings.h`.

- `indent` est un indenteur de code C destiné à faciliter la lecture des programmes sources. Pour porter `indent`, il a tout d'abord fallu modifier le fichier `Makefile` le construisant pour prendre en compte l'utilisation du compilateur croisé. Puis, la même opération que pour le programme `mcl_expand` a été nécessaire. Enfin, l'appel à la commande suivante (prototypée dans le fichier `strings.h`) :

  ```
  void bcopy(const void *s1, void *s2, size_t n);
  ```

  a été remplacé par la commande suivante (prototypée dans le fichier `string.h`) :

  ```
  void *memmove(void *s1, const void *s2, size_t n);
  ```

  Cette modification d'appel à des fonctions ayant le même rôle (copie d'octets d'une chaîne de caractères vers une autre) doit néanmoins se faire minutieusement car on constate qu'il faut inverser les arguments `s1` et `s2` lors du passage de `bcopy()` à `memmove()` pour garantir le même fonctionnement.

- `libcaesar.a`

  Le portage de cette bibliothèque a nécessité d'inclure le fichier `winsock.h` dans le cas d'une architecture Win32 afin de fournir un accès à l'appel réseau `htonl()`.

  Cet appel convertit des entiers non signés de 32 bits du format d'ordre d'octets de la machine hôte vers le format d'ordre d'octets requis par le réseau. Le format requis par le réseau est celui selon lequel les octets sont censés transiter dans le réseau. Tous les hôtes d'un réseau doivent faire transiter les octets selon le même ordre, mais tous les hôtes n'ont pas forcément une représentation interne des octets identique à celle du réseau. Les outils de Cadp utilisent cette fonction dans tous les programmes manipulant des champs de bits afin de garantir que le format d'ordre de traitement des octets sera le même quelque soit l'architecture.

Sur les architectures ou le format est identique à celui du réseau, la fonction `htonl()` est définie comme une macro `NULL`.

Le stockage des octets sur ordinateurs PC est de type "*little endian*" ce qui signifie que les octets sont rangés du poids le plus faible vers le poids le plus fort. A l'inverse les processeurs SPARC utilisent la convention "*big endian*" (octets forts vers octets faibles).

En guise d'illustration, le programme suivant :

```
#include <stdio.h>

main ()
{
        unsigned long y = 0xF1F2F3F4 ;

        printf ("%x", *((unsigned char *) &y)) ;
        printf ("%x", *((unsigned char *) &y) + 1 )) ;
        printf ("%x", *((unsigned char *) &y) + 2 )) ;
        printf ("%x", *((unsigned char *) &y) + 3 )) ;
}
```

donne le résultat `F1F2F3F4` sous SPARC et le résultat `F4F3F2F1` sous PC. C'est pourquoi il est préférable d'initialiser la variable `y` de la façon suivante :

```
        unsigned long y =  htonl (0xF1F2F3F4) ;
```

Pour utiliser la fonction `htonl()` sous WINDOWS NT, il faut donc avoir inclus le fichier `winsock.h` et utiliser la bibliothèque `libwsock32.a` au moment de l'édition de liens.

- `caesar.adt`

  Ce programme utilisait l'envoi de signaux au processus courant à l'aide de la commande `kill()` appelée de la façon suivante :

  ```
          kill ( getpid(), 15 ) ;
  ```

  Cet appel permettait de sortir du processus courant tout en renvoyant un code de sortie permettant de réaliser un traitement particulier à l'aide de la commande `signal()`.

  MINGW32 n'implémente pas la commande `kill` mais la commande `raise()` existe et convient tout à fait à réaliser la fonctionnalité décrite précédemment :

  ```
          raise(15) ;
  ```

Mais la commande `raise()` n'existant pas dans un environnement Sun Os 4.x, il a été nécessaire d'ajouter les lignes suivantes au début de chaque programme concerné :

```
#ifdef ARCHITECTURE_SUN_4
#define raise(x)  kill(getpid(),x)
#endif
```

- `caesar` utilise un certain nombre de signaux non implémentés dans Mingw32 : `SIGHUP, SIGQUIT, SIGBUS, SIGPIPE`. En guise de traitement, `caesar` ignore ces signaux par l'appel à la commande suivante :

  ```
  signal(SIGxxx,SIG_IGN) ;
  ```

  Il a été choisi de remplacer chaque appel de la forme précédente par les commandes :

  ```
  #include <signal.h>
  #ifdef SIGxxx
          signal(SIGxxx,SIG_IGN) ;
  #endif
  ```

  afin d'éviter un problème à la compilation vers l'architecture Win32, système pour lequel les signaux évoqués ci-dessus n'existent pas.

En complément, l'annexe B présente, sous forme d'un tableau, la correspondance entre les appels C Unix et leurs homologues Win32 en détaillant la nature des opérations à réaliser pour garantir leur fonctionnement optimal sur les deux environnements.

4.6.2 Lex et Yacc

Les programmes écrits dans les langages Lex et Yacc présents dans Eucalyptus n'ont nécessité aucune modification dans le cadre du portage vers Win32. En revanche, le code C généré à partir des outils `lex` et `yacc` peut poser des problèmes notamment dans le cas de l'initialisation de variables globales tel qu'il en a été question dans la section 3.6. En effet, la notation non-Ansi suivante :

```
FILE    *mystdin = {stdin} ;
```

nécessite d'être modifiée de la façon suivante :

```
FILE    *mystdin ;
...
main () {
```

```
mystdin = {stdin} ;
}
```

Pour éviter cette modification fastidieuse, et même impossible lorsque le code est généré à la volée (pendant l'exécution d'un programme) il a été choisi de préférer `flex` à `lex` pour assurer le respect de la norme ANSI.

Par ailleurs, certains programmes tels que `bcg_expand` ont été remaniés pour éviter d'utiliser la bibliothèque `libfl.a` au moment de l'édition de liens. Pour cela, il a simplement été rajouté deux déclarations de fonctions :

```
int yywrap() {
        return 1 ;
}
int main() {
        yylex() ;
}
```

Chapitre 5

Conclusion

Ce chapitre propose un bilan du travail réalisé au sein du projet VASY et introduit des perspectives possibles au présent projet. Pour la partie bilan, nous présenterons successivement :

- les remarques générales liées au présent dossier,
- les réalisations diverses,
- les apprentissages et expériences liés au projet.

5.1 Bilan du travail effectué

5.1.1 Remarques générales

De l'intérêt d'Internet

Les atouts du réseau Internet ne sont plus à démontrer, à tel point qu'il devient banal de dire que l'accès à l'information par le biais d'Internet procure une source intarissable d'éléments, même si ces derniers nécessitent parfois un peu de discernement.

En effet, il serait inconcevable de penser qu'un tel projet de migration eût été possible sans la mine d'or que représente Internet.

Notamment :

- le "réseau" a permis d'amasser une foule d'informations sur le sujet et recueillir des opinions à propos de telle ou telle solution,
- la capacité à obtenir instantanément (par téléchargement) des versions d'évaluation de chacun des produits cités dans ce dossier a été déterminante,
- l'adhésion à des listes de diffusion (*mailing lists*) a permis d'être tenu informé au jour le jour des nouveautés relatives à un produit (nouvelles versions, corrections de bogues,

etc). Les listes de diffusions utilisées (les plus actives) dans le cadre de ce projet ont été celle concernant la solution CYGWIN[27], celle à propos de la compilation croisée[28] et celle relative au produit UWIN[29],

- enfin, les forums de discussion (*newsgroups*) ont représenté un lieu d'échange d'expériences et de conseils pratiques précis à l'image de "`comp.lang.tcl`" (*newsgroup* très actif du langage TCL et de ses extensions).

Information de dernière minute

En septembre 1999, MICROSOFT acquiert la société Softway Systems Inc. qui distribue la solution INTERIX. Comme nous l'avons vu dans la section 3.5.2, INTERIX est une sous-système qui vient en remplacement du sous-système POSIX livré avec WINDOWS NT. Ce dernier présente de nombreuses limitations, et c'est pourquoi l'acquisition de Softway Systems par MICROSOFT devrait apporter une solution plus complète grâce au nouveau système POSIX (INTERIX).

Mise au point complète de l'existant

L'objectif du présent projet est clairement la migration d'applications UNIX vers l'environnement WINDOWS NT. Même si la boîte à outils CADP traite d'une problèmatique particulière qui est la validation de systèmes distribués, elle constitua, pour nous, une base de travail assez complète (voir section 2.2).

En outre, au fur et à mesure que la démarche de portage avançait, un autre objectif sous-jacent, mais non moins important, a été atteint : la mise au point et l'uniformisation de l'existant.

En effet, chaque outil porté a nécessité un effort de compréhension de son fonctionnement afin que son portage soit plus efficace. Cela a naturellement entraîné le fait qu'il soit modifié dans une optique globale et soit muni des armes qui le rendent le plus portable possible. Une optimisation ou un "nettoyage" du code a été réalisé chaque fois que cela a été possible.

De plus, étudier les programmes de façon précise nous a conduit à intégrer des modifications ou améliorations demandées par les utilisateurs, notamment dans le cas des interfaces graphiques EUCALYPTUS et INSTALLATOR.

En guise de synthèse, nous pouvons citer les points suivants :

- Tous les appels systèmes écrits "en dur" dans le code de l'interface graphique ont conduit à la création de scripts shell, indépendants de l'architecture, réalisant la même opération ou une opération optimisée.

[27] `cygwin@sourceware.cygnus.com`
[28] `crossgcc@cygnus.com`
[29] `uwin-users@research.att.com`

- Des améliorations ont été apportées dans les IHM afin de prendre en compte des aspects aussi variés que la configuration de la fenêtre principale, la gestion des polices de caractère ou la gestion des entrées/sorties produites par les programmes appelés.
- Des erreurs ont été corrigées dans les IHM. A titre d'exemple, la barre de progression du transfert FTP ne déborde plus.
- Des produits trop dépendants de l'architecture, tels que EXPECT ont été supprimés au profit d'une gestion basée sur des protocoles standard.
- Le script shell `tst` réalisant les tests de présence des commandes utilisées dans CADP a été complété des contrôles supplémentaires requis par l'environnement WIN32. De plus, pour UNIX, un diagnostic est réalisé sur le bon fonctionnement du *daemon* `sendmail`.
- Concernant le code C, la compilation croisée a permis de "tamiser" le code et s'assurer qu'il était conforme à la norme ANSI ou qu'il répondait aux exigences du portage vers WINDOWS NT.
- Les scripts Shell (`sh`) ont également subi des modifications afin de prendre en compte les caractéristiques des environnements CYGWIN et MKS TOOLKIT.

La section suivante apporte des éléments de volume (taille des programmes portés et quantité de modifications apportées) aux aspects évoqués ci-dessus.

5.1.2 Réalisations

Cette section est un retour en terme de synthèse sur les réalisations (présentées dans le chapitre 4) produites durant le projet.

Compilation croisée

Au début du présent projet de migration des applications de la boîte à outils CADP vers WINDOWS NT, les doutes les plus profonds étaient associés au portage du code C. Au fur et à mesure que les tests et évaluations ont avancé, ce doute s'est d'abord estompé et a finalement fini par devenir quasi invisible.

Cet état de fait est dû à la construction du compilateur croisé composé du compilateur C EGCS et de la bibliothèque MINGW32.

Bien que sa construction ait été minutieuse, les résultats qu'il a ensuite apporté à l'équipe VASY dans sa démarche de construction des binaires CADP ont été une large récompense aux efforts préalables. En effet, intégré à l'outil `Make-makefile`, il a permis de produire les exécutables au format WIN32 par la simple commande :

```
make win32
```

Ainsi, la plupart des outils a été portée vers WIN32 grâce à ce compilateur croisé.

Le tableau suivant récapitule les noms des outils écrits en C portés (ainsi que leurs tailles) dans le cadre du présent projet, et donne un aperçu des taux de modifications qui ont été effectuées pour parvenir au résultat escompté :

Nom du programme	Nombre de lignes de code	Nombre de lignes modifiées
`othello`	1581	0
`hostinfo`	107	80
`libsx.a`	13426	20
`bcg_expand`	100	6
`libXTL.a`	3500	0
`xtl_expand`	2000	0
`mcl_expand`	700	2
`libf2.a`	3320	2
`ppatruntime.a`	14000	0
`indent`	3142	5
`libBCG.a`	24000	15
`libBCG_IO.a`	10000	2
`xtl`	26000	20
`libcaesar.a`	7200	4
`caesar.adt`	8900	4
`evaluator.a`	5164	40
`exhibitor.a`	2300	5
`caesar.indent`	550	14
`caesar`	44000	4
TOTAL	169990	223 (soit 0,13 %)

Scrutator

Le script `scrutator` a été développé en Bourne Shell (`sh`) afin d'aider l'équipe VASY à déceler les problèmes potentiels dans le code source C.

Réalisant des rapprochements de chaînes de caractères avec le code source à l'aide de la commande `grep`, il ne peut en aucun cas représenter une solution explorant de façon exhaustive les programmes de CADP mais il permet néanmoins de "dégrossir" considérablement le problème épineux du portage des programmes C.

Portage des interfaces graphiques

Le portage des interfaces graphiques écrites en TCL et ses extensions a indéniablement constitué le problème majeur du portage de par la quantité importante de fonctionnalités dépendantes du système qu'elles mettaient en œuvre.

Ces fonctionnalités, d'abord intégrées dans le script TCL, ont été extraites du code TCL pour donner naissance à des scripts shell donnant le même résultat dans les environnements UNIX et WINDOWS NT. Puis, l'appel de ces scripts depuis l'interface graphique a donné lieu à de profonds réaménagements tels que synthétisés dans le tableau suivant :

Nom du script TCL	**NB lignes avant**	**NB lignes après**	**NB lignes modifiées**
`eucalyptus.tcl`	4519	4625	555
`bcg_edit.tcl`	3568	3545	58
`installator.tcl`	2418	2632	562
`ftp_lib.tcl`	0	1642	1642
TOTAL	10505	12444	2817 (soit 0,27 %)

Portage des scripts shell

Les scripts shell ont également subi de nombreuses modifications afin de répondre aux nouvelles exigences des interfaces graphiques EUCALYPTUS et INSTALLATOR ou pour simplement assurer un fonctionnement optimum sous WINDOWS NT. Le tableau suivant donne un aperçu des volumes de scripts shell impliqués dans cette refonte :

Nom du script TCL	**NB lignes avant**	**NB lignes après**	**NB lignes modifiées**
`arch`	26	64	38
`rfl`	417	417	2
`tst`	594	871	461
`xeuca`	137	132	64
`xeuca_convert`	231	241	27
`xeuca_echo`	23	23	0
`xeuca_fc2info`	25	25	0
`xeuca_info`	64	94	47
`xeuca_man`	109	114	27
`xeuca_ps`	37	43	13
`xeuca_version`	58	58	0
`xeuca_postscript`	0	49	49
`xeuca_print`	0	31	31
`xeuca_shell`	0	31	31
`xeuca_web`	0	36	36
`cadp_crlf`	0	47	47
`cadp_edit`	0	48	48

Nom du script TCL	NB lignes avant	NB lignes après	NB lignes modifiées
cadp_mail	0	180	180
install_df (ex run_df)	30	35	7
install_logname	0	64	64
install_rfl (ex run_rfl)	25	25	0
install_setup (ex run_setup)	67	84	17
install_uncompress (ex run_uncompress)	22	39	17
install_version (ex cadp_version)	27	27	0
TOTAL	1892	2778	1206 (soit 0,64 %)

5.1.3 Apprentissage et expérience

Les réalisations logicielles citées précédemment ont été l'occasion d'acquérir un certain nombre de connaissances telles que :

- les mécanismes systèmes tels que la communication inter-processus, la gestion des tubes, l'implémentation du `fork` et l'API WIN32,
- l'environnement de compilation du C comprenant les commandes de compilation, d'édition de liens, les Makefiles, le gestionnaire de versions `sccs`, le vérificateur de programmes C `lint`, l'analyseur lexical (LEX) et l'analyseur syntaxique (YACC),
- la programmation en C sous SUN OS et dans l'environnement MICROSOFT Visual C++,
- le langage TCL et ses extensions TK, TIX et EXPECT,
- les protocoles SMTP et FTP,
- l'administration d'un serveur WINDOWS NT.

De plus, ce projet m'a permis de comprendre de nombreux principes et m'a apporté une bonne culture générale à propos des différents concepts qui se situent à l'intersection des deux mondes UNIX et WINDOWS NT.

Enfin, le fait d'évoluer dans un contexte tel que celui de l'INRIA a conduit à un enrichissement personnel du fait des personnes côtoyées tant au niveau de l'équipe VASY que dans différents autres projets.

5.2 Etablissement d'une méthodologie générale

Le présent document représente l'étude du portage concret des applications de la boîte à outils CADP vers WINDOWS NT. De par la diversité des problèmes rencontrés et des outils qui composent CADP (langages compilés, langages de script, protocoles, utilitaires, etc), nous pouvons estimer que cette étude représente la généralité du problème du portage d'applications UNIX vers WINDOWS NT.

Par ailleurs, une telle étude montre que chaque projet de portage est unique, de par ses spécificités techniques, ses enjeux économiques et sa cible d'utilisateurs.

En effet, dans le cas qui nous concerne (portage de CADP), il a été choisi d'utiliser MINGW32 pour construire le compilateur croisé, bien que CYGWIN semblait apporter des bibliothèques C plus complètes. Ce choix fût purement économique puisque MINGW32 produit des binaires indépendants de toute DLL contrairement à CYGWIN qui nécessite la présence (onéreuse) de `cygwin.dll`.

C'est pourquoi, même si une des perspectives de ce document serait naturellement de produire une "recette" générale, il est périlleux de définir une méthodologie globale de portage qui puisse s'appliquer directement dans tous les cas.

Néanmoins, nous conseillerons, à titre d'heuristique globale, les points suivants :

- Uniformiser l'écriture des scripts shell en n'utilisant qu'un seul interpréteur : le Bourne Shell (**sh**), de par son degré de compatibilité avec les autres interpréteurs.
- Eviter le codage "en dur" de spécificités dépendantes du système telles que le chemin des fichiers, la gestion des fenêtres ou la primitive d'impression d'un document.
- Eviter l'utilisation d'applications "exotiques" qui n'ont aucune chance de fonctionner dans un autre environnement.
- Se conformer strictement à la norme ANSI pour l'écriture de programmes C.
- Considérer une architecture logicielle qui prend en compte l'importance de la maintenance ultérieure des sources.
- Choisir une solution modulaire qui permette une intégration aisée des nouveaux outils.

Enfin, l'annexe B établit une table de correspondance entre les appels systèmes UNIX et WINDOWS NT.

5.3 Perspectives

Le travail réalisé a permis d'établir une démarche à suivre et une liste d'outils à utiliser afin de migrer une grande partie de la boîte à outils CADP. Le portage complet de CADP nécessite encore une recompilation progressive de chacun des outils qui le composent tout en prenant en compte les modifications nécessaires à l'optimisation du code pour l'environnement WIN32.

Les programmes restant à générer, par la compilation croisée, pour l'environnement WIN32 sont les suivants : aldebaran, bcg_io, bcg_labels, bcg_lib, des2aut, exp2c, exp2fc2, libbcg_draw.a, libbcg_info.a, libbcg_io.a, libbcg_open.a, libexpopen.a, projector.a, xsimulator.a et xtl.

Dès lors, une version WIN32 de CADP pourra être mise à disposition des utilisateurs.

Outre ces aspects liés à la compilation croisée des programmes existants, il demeure un certain nombre de points qui amélioreraient sensiblement les outils sous différentes formes :

- Amélioration de `scrutator` par l'analyse syntaxique

  `scrutator` réalise aujourd'hui des rapprochements entre le code C et des chaînes de caractères prédéfinies à l'aide de la commande `grep`. Cela n'est pas optimal car la syntaxe du C peut s'avérer complexe et ce type de rapprochement pourrait laisser passer quelques problèmes potentiels. C'est pourquoi, il serait intéressant, et bien plus exhaustif, d'utiliser un analyseur syntaxique (YACC, par exemple) avec une grammaire simplifiée du C, pour mieux analyser les expressions régulières présentes dans le code C.

- Amélioration de INSTALLATOR

  Actuellement, INSTALLATOR peut être téléchargé sous forme d'un fichier `.shar` auto-extractible sur la ligne de commande d'un interpréteur shell.

  Afin de "coller" davantage à l'apparence WINDOWS, il conviendrait d'utiliser plutôt un format de fichier ZIP (à l'aide de l'outil WINZIP, par exemple).

- Réécriture l'interface graphique en JAVA

  L'écriture d'interface graphique en TCL (et ses extensions) est attractive au premier abord, mais elle présente un inconvénient majeur de par le fait qu'un script soit interprété et non compilé.

  La maintenance de ce type de scripts est difficile car toute erreur de syntaxe n'est décelée qu'au moment de l'exécution. De plus, le code écrit n'est en aucun cas protégé sauf si celui-ci est "brouillé" comme cela l'a été choisi dans cette étude (voir section 4.4.5) .

  C'est pourquoi, il serait intéressant d'étudier, à l'avenir, la possibilité d'utiliser JAVA pour l'écriture des interfaces homme-machine de la boîte à outils CADP.

- Intégration des solutions INTERIX et UWIN

  A ce jour, les seules solutions préconisées pour le bon fonctionnement de CADP sont CYGWIN et MKS TOOLKIT car les environnements UWIN et INTERIX présentent des problèmes dans la gestion de la communication par tube implémentée dans EUCALYPTUS et INSTALLATOR (voir section 4.2.6).

  Si ces problèmes venaient à être resolus, l'intégration de UWIN et INTERIX serait possible.

Afin d'évaluer le bon fonctionnement de la communication par tube sous WINDOWS NT, le script suivant peut être testé :

```
set input [open "|sh " r+]
fconfigure $input -translation binary
fconfigure $input -buffering none
fileevent $input readable Log

puts $input "ls"
console show

proc Log {} {
        global input

        gets $input line
        puts $line
}
```

Ce script ouvre une console destinée à afficher le résultat de la commande `ls`.

Il fonctionne tout à fait normalement avec les solutions CYGWIN et MKS TOOLKIT. Pour cette dernière, il faut simplement remplacer l'option `binary` par `crlf` (opération réalisée par le script shell `cadp_crlf`).

Annexe A

Construction du compilateur croisé

Cette annexe a pour but de présenter la procédure de construction du compilateur croisé SUN OS 5.X vers WIN32 utilisant la bibliothèque MINGW32 et le compilateur EGCS.

- Etape 1 : téléchargement des sources du compilateur

  Considérant que la construction du compilateur croisé requiert les sources du compilateur EGCS, il est nécessaire de télécharger `binutils` (pour l'assemblage et l'édition de liens) depuis le site internet `http://www.gnu.org/order/ftp.html` et `egcs` depuis `ftp://egcs.cygnus.com/pub/egcs/releases/egcs-1.1.2/`.

  Note : on notera `$DOWNLOAD` le répertoire dans lequel les sources ont été téléchargées.

- Etape 2 : téléchargement des sources des bibliothèques

  Afin de se munir des bibliothèques et fichiers `.h` indispensables à la constitution de l'environnement WIN32, il est primordial de télécharger les solutions MINGW32 depuis le site internet `http://www.geocities.com/Tokyo/Towers/6162/mingw32_980701_tar.gz` et WIN32API (fonctionnant avec MINGW32) depuis `ftp://agnes.dida.physik.uni-essen.de/home/janjaap/mingw32/newnew/win32api-980701-4.zip`.

- Etape 3 : positionnement de variables d'environnement

  Note : on réalisera les opérations qui suivent dans un Bourne Shell.

```
COMPILER=egcs
LIBRARY=mingw32
VERSION=-1.1.2

SRCDIR= Chemin qui contiendra les sources suite au uncompress
PREFIX= Chemin qui contiendra les binaires construits
BUILDGCC= Chemin tampon de fabrication du cross-compilateur GCC
BUILDBINUTILS= Chemin tampon de fabrication des utilitaires GNU
```

```
TMPDIR=/tmp
TARGET=i386-pc-mingw32
PROGRAM_PREFIX=i386-pc-$LIBRARY-
```

- Etape 4 : Installation de la bibliothèque Mingw32

```
install -d $SRCDIR
install -d $PREFIX
install -d $PREFIX/include
install -d $PREFIX/lib
install -d $PREFIX/$TARGET/include
install -d $PREFIX/$TARGET/lib

mkdir $SRCDIR/Mingw32
cd $SRCDIR/Mingw32
gtar xvfz $DOWNLOAD/mingw32_980701.tar.gz

cp -r include/* $PREFIX/include
cp -r lib/* $PREFIX/lib
cp -r include/* $PREFIX/$TARGET/include
cp -r lib/* $PREFIX/$TARGET/lib

mkdir $SRCDIR/Win32API
cd $SRCDIR/Win32API
unzip $DOWNLOAD/win32api-980701-4.zip
cp -r include/* $PREFIX/include
cp -r lib/* $PREFIX/lib
cp -r include/* $PREFIX/$TARGET/include
cp -r lib/* $PREFIX/$TARGET/lib
```

- Etape 5 : Décompression des sources

```
gtar xfvz $DOWNLOAD/binutils-2.9.1.tar.gz
gtar xfvz $DOWNLOAD/egcs-core$VERSION.tar.gz
```

- Etape 6 : Configuration de `binutils`

```
install -d $BUILDBINUTILS
cd $BUILDBINUTILS
$SRCDIR/binutils-2.9.1/configure -v -prefix=$PREFIX -target=$TARGET \
-program-prefix=$PROGRAM_PREFIX
gmake CC="gcc -static" all
```

```
gmake CC="gcc -static" install
```

- Etape 7 : Configuration du compilateur

  ```
  install -d $BUILDGCC
  cd $BUILDGCC
  ${SRCDIR}/${COMPILER}${VERSION}/configure -v -prefix=$PREFIX  -target=$TARGET
  cd gcc
  gmake installdirs
  cd ..
  vi Makefile
  ```

  Ici, il est indispensable de modifier le fichier `Makefile` en remplaçant, dans la règle `.PHONY:` la ligne

  ```
  LANGUAGES="c c++"
  ```

  par la ligne

  ```
  LANGUAGES="c"
  ```

  Puis, continuer la procédure par les commandes suivantes :

  ```
  gmake LANGUAGES="c" cross
  gmake LANGUAGES="c" install
  ```

Note : parce que Mingw32 ne comporte pas de bibliothèque `libm.a` (les fonctions mathématiques sont présentes dans la bibliothèque générique `libmingw32.a`), il est recommandé de la construire afin d'éviter tout problème lors de la compilation de programmes l'utilisant.

Pour cela, il suffit de compiler un programme `libm.c` ne comportant que la ligne suivante :

```
static int pseudo_mingw32_libm = 0;
```

et générer la bibliothèque par la commande :

```
ar -rcv libm.a libm.o
```

Annexe B

Table de correspondance des appels systèmes UNIX vers WINDOWS NT

Dans cette annexe, nous avons synthétisé les commandes ou propriétés du langage C qui représentent un problème potentiel de portage vers WIN32.

Pour chacune d'elle, nous proposons une solution impliquant une modification dans le code source.

L'ensemble des commandes UNIX et de leurs équivalences sous WINDOWS NT est présenté sous forme d'un tableau présenté sur la page suivante.

Commande C sous Unix	**Equivalent sous Windows NT**	**Observations**
`fopen(filename, "w")`	`fopen(filename, "wb")`	Ouverture d’un fichier binaire
`bzero(s, n)`	`memset(s, ’\0’, n)`	Manipulation de chaînes de caractères
`bcopy (s1, s2, n)`	`memcpy (s2, s1, n)`	Manipulation de chaînes de caractères
`bcmp (s1, s2, n)`	`memcmp (s1, s2, n)`	Manipulation de chaînes de caractères
`index (s, c)`	`strchr (s, c)`	Manipulation de chaînes de caractères
`rindex (s, c)`	`strrchr (s, c)`	Manipulation de chaînes de caractères

`system("$VAR/mycmd")`	`var = getenv(VAR) ;`
	`sprintf(command,"%s/mycmd",var) ;`
	`system(command) ;`
		Substitution de variables

`popen("$VAR/mycmd","r")`	`var = getenv(VAR) ;`
	`sprintf(command,"%s/mycmd",var) ;`
	`popen(command,"r")`
		Substitution de variables

`execl("$VAR/mycmd", ...) ;`	`var = getenv(VAR) ;`
	`sprintf(command,"%s/mycmd",var) ;`
	`execl(command, ...) ;`
		Substitution de variables

`FILE *mystdin = stdin`	`FILE *mystdin ; mystdin = stdin`	Initialisation de variables globales

`signal (SIGxxx, PROC)`	`#ifdef SIGxxx`
	`signal (SIGxxx, PROC)`
	`#endif`
		Prise en compte des signaux absents

`rint`	`ceil` ou `floor`	Arrondi correspondant à la norme IEEE754 [MD97]
`unsigned long y=0xF1F2F3F4`	`unsigned long y=htonl(0xF1F2F3F4)`	Problème du format d’ordre des octets
`cuserid()`	`getlogin()`	Nom de l’utilisateur connecté
`ftruncate()`	mécanisme `read()/write()`	Donne une nouvelle taille à un fichier

Bibliographie

[ASU86] Alfred V. Aho, Ravi Sethi, and Jeffrey D. Ullman. *Compilers: Principles, Techniques and Tools.* Addison-Wesley, 1986.

[Cor96] Digital Equipment Corporation. *Unix to Microsoft Windows NT - Application Migration Guide.* 1996.

[Cus93] Helen Custer. *Inside Windows NT.* foreword by David N. Cutler. - Microsoft Press, 1993.

[EP87] Sandra L. Emerson and Karen Paulsell. *Troff : Typesetting for Unix Systems.* McGraw-Hill, 1987.

[Gro93] James Groves. *Windows NT : Les reponses.* Microsoft Press, 1993.

[IEE90] IEEE. *ISO/IEC 9945-1:1990, Information technology – Portable Operating System Interface (POSIX) – Part 1: System Application Program Interface (API) [C Language].* 1990.

[Int98a] Interix. Porting Applications in C. Technical Report 002, Softway Systems, Incorporated, September 1998.

[Int98b] Interix. Porting Shell Scripts. Technical Report 001, Softway Systems, Incorporated, June 1998.

[ISO88] ISO/IEC. LOTOS — A Formal Description Technique Based on the Temporal Ordering of Observational Behaviour. International Standard 8807, International Organization for Standardization — Information Processing Systems — Open Systems Interconnection, Genève, septembre 1988.

[Kor97] David Korn. Porting Unix to Windows NT. Technical Report, AT&T Laboratories, January 1997.

[MD97] Jean-Michel Muller Marc Daumas. *Qualité des Calculs sur Ordinateur : Vers des arithmétiques plus fiables.* Masson, 1997.

[Mic88] Sun Microsystems. *SunOs Programming Utilities and Libraries.* Sun Microsystems, 1988.

[MR96] J. Myers and M. Rose. *Post Office Protocol - Version 3.* Request For Comments (RFC 1939) - Information Sciences Institute University of Southern California, 1996.

[Ous94] John K. Ousterhout. *Tcl and the Tk Toolkit.* Addison-Wesley, 1994.

[Pos82] Jonathan B. Postel. *Simple Mail Transfer Protocol.* Request For Comments (RFC 821) - Information Sciences Institute University of Southern California, 1982.

[PR85] J. Postel and J.K. Reynolds. *File Transfer Protocol.* Request For Comments (RFC 959) - Information Sciences Institute University of Southern California, 1985.

[Qui97] Ellie Quigley. *Unix Shells by example.* Prentice Hall, 1997.

[RFBL99] J. Mogul H. Frystyk L. Masinter P. Leach R. Fielding, J. Gettys and T. Berners-Lee. *Hypertext Transfer Protocol – HTTP/1.1.* Request For Comments (RFC 959) - Information Sciences Institute University of Southern California, 1999.

[Rif91] Jean-Marie Rifflet. *La programmation sous Unix, Deuxième édition.* McGraw-Hill, 1991.

[Rit84] B.W Kernighan D.M. Ritchie. *Le langage C.* Masson, 1984.

[Sch93] Herbert Schildt. *The Annotated ANSI C standard : American national standard for programming languages C . ANSI-ISO 9899-1990.* McGraw-Hill, 1993.

[Sol98] David A. Solomon. *Inside Windows NT Second Edition.* Microsoft Press, 1998.

[Sup98] Datafocus Technical Support. The Nutcracker Porting Guide. Technical Report Version 4.0, DataFocus, Incorporated, April 1998.

[TT91] J.R. Tong-Tong. *Les communications sous Unix.* Eyrolles, 1991.

[Wel95] Brent Welch. *Practical Programming in Tcl and Tk.* Prentice Hall, 1995.

Index

Aldo MAZZILLI - Mémoire d'ingénieur C.N.A.M. - Grenoble 2000

Etude de la migration et de la portabilité des applications informatiques d'Unix vers Windows NT

Bien qu'il soit toujours périlleux de prédire le futur de l'industrie informatique, on peut raisonnablement prévoir que, pendant les prochaines années, deux grandes classes de systèmes d'exploitation vont coexister et se partager le marché : d'une part, les systèmes basés sur Unix (par exemple Solaris, Linux, Aix, etc.) et, d'autre part, les systèmes d'exploitation développés par Microsoft, notamment Windows NT.
Pour la communauté informatique académique (qui, par tradition, possède une forte culture dans l'utilisation d'Unix), se pose le problème de la prise en compte du système Windows NT.
Parmi les principales raisons qui motivent ce choix, on peut citer : l'importante base installée sous Windows NT, le développement de logiciels applicatifs à destination du grand public, la nécessité de collaborer avec certains industriels travaillant sous Windows NT, la réduction des coûts de matériel, l'harmonisation du parc informatique entre chercheurs et administratifs, etc.
Notre travail concerne la migration des applications développées par l'équipe Vasy de l'Inria Rhône–Alpes sur une plate-forme Unix vers l'architecture Windows NT. Concrètement, il s'agit d'apporter, à l'équipe Vasy, les méthodes et outils qui lui permettront de développer une version de la boîte à outils Cadp pour Windows NT.
Le portage des outils qui composent Cadp a nécessité la prise en compte des problèmes relatifs à la migration d'applications écrites en langage C, de scripts shell écrits en Bourne Shell, d'interfaces graphiques écrites en Tcl/Tk, d'outils basés sur les protocoles Smtp ou Ftp et de commandes utilisateur présentant une forte dépendance avec l'architecture (`man`, `lpr`, `xterm`, etc).
Comme sous-produit de notre travail, un compilateur croisé Unix vers Win32 et un analyseur de code ont été réalisés pour faciliter le portage des applications écrites en langage C.

Mots-clés :
migration, portage, compilation croisée, Unix vers Windows, langage C, scripts shell, interface graphique.

Keywords:
migration, cross-compilation, Unix to Windows, C language, shell scripts, graphical interface.