4M une messagerie multimedia opérant par des messages actifs

Elisabeth Roudier

ECOLE NATIONALE SUPERIEURE
DES MINES DE SAINT-ETIENNE

N° ordre: 89 ID

THESE

présentée à Saint-Etienne, le 15 Juin 1993 par

Elisabeth Roudier

4M, une messagerie multimédia opérant par des messages actifs

pour obtenir le titre de

DOCTEUR en informatique

de l'ECOLE NATIONALE SUPERIEURE DES MINES DE SAINT-ETIENNE
et de l'UNIVERSITE DE SAINT-ETIENNE

COMPOSITION du JURY

Monsieur M. Habib	*président et rapporteur*
Monsieur Y. Raynaud	*rapporteur*
Monsieur E. Ahronovitz	*examinateurs*
Madame Y. Ahronovitz	
Monsieur C. Dony	
Monsieur J.J. Girardot	
Monsieur C. Huitema	

ECOLE NATIONALE SUPERIEURE
DES MINES DE SAINT-ETIENNE

N° ordre: 89 ID

THESE

présentée à Saint-Etienne, le 15 Juin 1993 par

Elisabeth Roudier

4M, une messagerie multimédia opérant par des messages actifs

pour obtenir le titre de

DOCTEUR en informatique
de l'ECOLE NATIONALE SUPERIEURE DES MINES DE SAINT-ETIENNE
et de l'UNIVERSITE DE SAINT-ETIENNE

COMPOSITION du JURY

Monsieur M. Habib	*président et rapporteur*
Monsieur Y. Raynaud	*rapporteur*
Monsieur E. Ahronovitz	*examinateurs*
Madame Y. Ahronovitz	
Monsieur C. Dony	
Monsieur J.J. Girardot	
Monsieur C. Huitema	

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

Monsieur le Professeur Michel Habib du L.I.R.M.M. de Montpellier qui me fait l'honneur de présider ce jury de thèse, et qui a contribué par ses remarques à l'amélioration de ce rapport,

Monsieur le Professeur Yves Raynaud de l'université Paul Sabatier de Toulouse, pour avoir aimablement accepté d'être rapporteur de ce travail,

Monsieur Chistian Huitema, Directeur de recherche de l'INRIA Sophia-Antipolis pour sa participation au jury,

Monsieur Christophe Dony, Maître de conférence au L.I.R.M.M. de Montpellier pour sa présence dans le jury et ses remarques pertinentes sur ce rapport,

Madame Yolande Ahronovitz, Maître de conférence à l'Université Jean Monnet de Saint-Etienne pour être membre du jury,

Monsieur Jean-Jacques Girardot, Maître de recherche à l'Ecole Nationale Supérieure des Mines de Saint-Etienne pour voir encadré ce travail et l'avoir enrichi par ses intuitions et ses conseils judicieux,

Monsieur Ehoud Ahronovitz, ingénieur de recherche à l'Ecole Nationale Supérieure des Mines de Saint-Etienne pour ses explications, sa disponibilité et son soutien,

Monsieur le Professeur Bernard Péroche, Directeur du Département Informatique Appliquée de l'Ecole Nationale Supérieure des Mines de Saint-Etienne pour m'avoir accueillie dans son laboratoire,

Monsieur Paul-André Pays, ingénieur à l'INRIA Rocquencourt pour avoir initié ce travail,

Les membres du Département informatique Appliquée pour leur aide amicale et leur gentillesse, Marie-Line, Florence, Annie et Hélène, mes anciens compagnons de bureau, Suzan et Bernard, ainsi que Daniel, Dominique, François, Jean-Michel, Jean-Pierre, Gabriel, Marc, Michel, Mohand-Ourabah, Philippe et tous les autres,

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

Mes parents, Geneviève, Denis et Pascale pour leur affection et leur soutien constant au cours de toutes ces années.

Sommaire

INTRODUCTION 5

I LA MESSAGERIE MULTIMÉDIA 11

CHAPITRE 1 Présentation 13

1. LE MONDE DU MULTIMEDIA 13
   1.1. Concept du multimédia 13
   1.2. Applications multimédias 14
   1.3. Messagerie multimédia 15
2. ENVIRONNEMENT MULTIMÉDIA 18
   2.1. Contraintes techniques 18
   2.2. Contraintes logiques 22

CHAPITRE 2 État de l'art 29

1. SITUATION 29
2. SYSTÈMES EXISTANTS 30
   2.1. Andrew 30
   2.2. Diamond/Slate 31
   2.3. MMS (Multimedia Mail System) 32
   2.4. Systèmes commerciaux 33
   2.5. MetaMail 33
3. PERSPECTIVES 34
   3.1. Approches en cours 34
   3.2. Besoins 35
   3.3. Objectifs 36

II LE MODÈLE PROPOSÉ 39

CHAPITRE 3 Modèle de communication 41

1. PRÉSENTATION 41
   1.1. Contexte 41
   1.2. Exemples 42
   1.3. Besoins 43
2. MODELE DE MESSAGERS 44
   2.1. Composantes 44

2.2. Systèmes à acteurs 46
2.3. Caractéristiques de notre modèle 48
2.4. Communications entre acteurs 49
3. SYSTEME DE MESSAGERS 52
3.1. Architecture 52
3.2. Propriétés 54
3.3. Implantation 55
4. AVANTAGES DU MODELE 55

CHAPITRE 4 Messages multimédias 57

1. ARCHITECTURE SPATIO-TEMPORELLE 57
1.1. Présentation 57
1.2. Architecture spatiale 59
1.3. Synchronisation 61
2. COMPORTEMENT 63
2.1. Messages actifs 63
2.2. Devenir des messages 64
3. SCÉNARIO DE MESSAGES 65
3.1. Référence cinématographique 65
3.2. Exemple de scénario 65

CHAPITRE 5 Représentation 67

1. CHOIX D'UNE REPRÉSENTATION 67
1.1. Description structurée 67
1.2. Approche objets 69
1.3. Approche messagers 71
2. IMPLANTATION DU SCRIPT 73
2.1. Langage intégré 73
2.2. Choix du langage 74
3. CONCLUSION 76

III UNE RÉALISATION : 4M 79

CHAPITRE 6 Présentation générale 81

1. PRÉSENTATION 81
1.1. Le projet 81
1.2. Caractéristiques 82
1.3. Objectifs 82
2. DESCRIPTION 83
2.1. Principe de base 83
2.2. Fonctionnement 84
2.3. Choix effectués 85

2.4. Architecture du système 86

CHAPITRE 7 Modules fonctionnels 89

1. MODULE DE COMPOSITION (PCM) 89
   1.1. Principe 89
   1.2. Sous module PCT 90
   1.3. Sous module SCT 92
2. MODULE DE RESTITUTION (MRM) 93
   2.1. Principe 93
   2.2. Restitution 93
   2.3. Outils complémentaires 95
3. MODULE DE TRANSMISSION (MPM) 97
   3.1. Principe 97
   3.2. Format des messages 4M 97
   3.3. 4M et X400 98
4. AUTRES FONCTIONNALITÉS 99
   4.1. Gestion des messages 99
   4.2. Interface avec GLisp 100

CHAPITRE 8 Interface utilisateur 103

1. PRÉSENTATION 103
2. CONCEPTION DE L'INTERFACE 104
   2.1. Outils d'aide 104
   2.2. Caractéristiques 105
   2.3. Vue d'ensemble 107
3. IMPLANTATION 107
   3.1. Choix effectués 107
   3.2. Aspects de portabilité 108

CHAPITRE 9 Scripts 4M 113

1. FONCTIONS DE SCRIPT 113
   1.1. Langage d'extension GLisp 114
   1.2. Fonctions de présentation 115
   1.3. Fonctions de synchronisation 119
   1.4. Fonctions externes 119
   1.5. Fonctions de comportement 120
   1.6. Configuration 121
2. ÉXECUTION DES SCRIPTS 121
   2.1. Notion de tâches 121
   2.2. Synchronisation des tâches 122
   2.3. Implantation des tâches 122
3. EXEMPLES DE MESSAGES 124
   3.1. Structure générale 124

3.2. Considérations spatiales 124
3.3. Coordination temporelle 126
3.4. Exemple X11 127
3.5. Exemple récapitulatif 128
4. EXTENSIONS POSSIBLES 133

CONCLUSION 135

ANNEXE A 139

ANNEXE B 149

Bibliographie 155

Liste des figures 163

Glossaire 165

INTRODUCTION

Le multimédia se situe au croisement de l'audio-visuel, de l'informatique et des télécommunications. Il qualifie toute entité qui réunit divers médias en offrant une certaine interactivité avec l'utilisateur. Apparu depuis quelques années dans le monde informatique, son essor est timide ; si des applications existent, elles sont surtout axées pour l'instant dans les domaines de la formation et de l'information. Les applications les plus courantes sont les présentations multimédias (publicité, promotion de produits, bornes interactives) et les jeux. En fait, ce n'est pas l'absence de technologies qui freine le développement du multimédia ; bien au contraire, l'utilisateur se trouve devant un trop grand nombre de possibilités tant dans le choix des matériels et logiciels à utiliser que dans les standards à suivre. L'essor du multimédia et de ses applications est ralenti par l'hétérogénéité des environnements liée à l'absence de conversion ou de normalisation, comme d'ailleurs de nombreux outils existants (logiciels de PAO, par exemple).

La messagerie multimédia est l'une des nombreuses applications touchées par le concept du multimédia. Elle associe à la messagerie électronique des outils de conception et de transmission de messages composés de différents types de médias tels que des textes, des images, des graphiques, des séquences audio ou vidéo. Ainsi, la messagerie électronique, support privilégié de communication, devient par l'apport du multimédia, un moyen sophistiqué d'échanger des messages complexes et personnalisés. L'arrivée du multimédia contribue à enrichir les communications mais aussi entraîne de nouveaux besoins dans un domaine où la diversité des environnements est un aspect primordial. Si les systèmes de messagerie électronique ont connu un certain essor, peu de systèmes fournissent la possibilité de créer, d'envoyer et de recevoir des messages multimédias. En effet, bien que les premiers prototypes datent de quelques années, différents problèmes matériels et surtout logiciels ont ralenti leur développement. La difficulté est double : d'un côté, il faut assurer la transmission de messages multimédias dans des environnements non homogènes, de l'autre prendre en compte les caractéristiques des messages. Ainsi, bien que les messageries multimédias existantes montrent le potentiel de ce type de communication, elles sont encore peu utilisées, et seulement par des communautés spécifiques d'utilisateurs.

Les premières études dans le contexte de la messagerie multimédia, nous ont amené à réfléchir sur le problème complexe et plus vaste de l'échange de données quelconques (multimédias on non) dans des environnements hétérogènes. Ces problèmes de communication d'informations entre applications ou usagers ont constitué le contexte de l'étude présentée. Nous proposons dans cette étude une méthode originale d'échanges de données *actives* s'appuyant sur un modèle théorique de messagers acteurs chargés de réaliser des communications spécifiques.

Cette thèse est organisée en trois parties.

La première partie expose le cadre de l'étude : la messagerie multimédia. Dans un premier chapitre nous présenterons rapidement le monde du multimédia, ses applications, et les nouveaux besoins qu'il engendre. Le domaine du multimédia est très vaste et évolue rapidement, aussi ne nous intéresserons-nous qu'aux aspects de l'environnement multimédia se rapportant à la messagerie électronique : matériels disponibles, logiciels de

conception et de transmission de messages, formattage des messages. Le chapitre suivant a pour but de fournir un aperçu de l'état de l'art actuel de la messagerie multimédia. Sont décrit ainsi, pour référence, quelques systèmes de messagerie multimédia expérimentaux ou commerciaux dont nous détaillons les concepts et les diverses approches. A partir de l'étude de ces systèmes et des contraintes engendrées par la messagerie multimédia, nous exposons les objectifs de l'étude.

Les divers problèmes rencontrés dans le contexte de la messagerie multimédia, nous ont amené à nous intéresser au domaine plus vaste de la communication entre applications quelconques. Cette deuxième partie propose une solution générale répondant aux besoins de communication d'informations complexes et multimédias entre usagers et applications dans des environnements hétérogènes. Ce modèle est un modèle de communication dont le principe est de représenter les communications par des *agents actifs* circulant sur des réseaux. Il repose sur un modèle acteur dans lequel chaque objet contient en plus de ses propres données un certain *comportement* lui permettant de dialoguer avec l'environnement extérieur et d'échanger ou transmettre des informations par l'envoi de messages. Cette approche a pour but de faciliter les échanges entre applications non prévues pour communiquer entre elles. Elle permet de résoudre le problème de la communication d'informations spécifiques et difficilement structurables, comme par exemple un message multimédia. En effet un message multimédia ne doit pas être assimilé à un document multimédia statique car il est assujetti à des contraintes spatiales et temporelles. D'autre part, certaines actions vont être déclenchées par des événements tels que des interventions de l'utilisateur. Le modèle proposé permet de représenter les caractéristiques de ce type de message en offrant la souplesse et la puissance nécessaires. Le premier chapitre de cette partie détaille les aspects généraux et les concepts du modèle de communication proposé tandis que les deux chapitres suivants démontrent l'adéquation de ce modèle au domaine plus restreint de la messagerie multimédia : l'un s'attache à définir les caractéristiques propres des messages multimédias et en particulier leur architecture spatio-temporelle : l'autre présente les raisons du choix du modèle comme forme de représentation de ces messages, parmi diverses autres formes possibles.

Pour illustrer cette approche, nous avons développé un outil de messagerie multimédia, présenté dans une troisième partie. Le projet intitulé 4M (MultiMedia Mail Manager) est un agent utilisateur de messagerie pour la composition et la restitution de messages multimédias dans des environnements hétérogènes. Cet outil est un prototype simple dont l'objectif principal est de prendre en compte l'architecture spatio-temporelle spécifique des messages multimédias telle qu'elle a été présentée dans la partie précédente. Ceci est réalisé au moyen d'un scénario qui coordonne la présentation et le comportement du message et qui est décrit, conformément au modèle proposé, par un script. Celui-ci est transmis avec les différentes parties du message dans un message unique. Le message est ainsi représenté par un agent contenant les différents comportements nécessaires à la présentation interactive des informations transmises. Ces comportements sont prédéfinis (composition de texte, affichage d'images, restitution de sons, assemblage, synchronisation, interaction avec le client), ou construits spécifiquement pour ce message. Pour faciliter la création des scripts, on fournit un *langage de commande* construit sur le *langage d'extension*. Le message multimédia est restitué par l'exécution du script sur le site récepteur. Les différents chapitres de cette partie exposent les concepts mis en oeuvre et le fonctionnement de l'application. Après la présentation des caractéristiques du projet,

sont expliqués la composition des différents modules fonctionnels (composition, transmission et restitution des messages) et les aspects de l'interface utilisateur. Le dernier chapitre est axé sur les scripts des messages. Il présente les fonctions du langage d'extension fournies à l'utilisateur pour composer ses messages puis détaille les possibilités offertes par l'inclusion de scripts en s'appuyant sur divers exemples. La partie conclut sur les développements en cours apportés à cet outil.

Nous terminons en montrant l'intérêt et la validité du modèle proposé dans le cadre de la messagerie multimédia et les perspectives que l'on peut envisager par l'extension d'un tel modèle au domaine de la communication entre personnes et applications.

LA MESSAGERIE MULTIMÉDIA I

1. Présentation
2. État de l'art

CHAPITRE 1

Présentation

La messagerie électronique est un moyen de transmettre et d'échanger des idées ; le multimédia lui offre un support privilégié. Ce chapitre a pour but de présenter rapidement le monde du multimédia, ses applications et les nouveaux besoins qu'il engendre, en particulier dans le cadre d'une messagerie électronique. Le domaine du multimédia est très vaste et évolue rapidement, aussi ne nous intéresserons-nous qu'aux aspects de l'environnement multimédia d'aujourd'hui qui sont propres à la messagerie.

1. LE MONDE DU MULTIMEDIA

Combiner plusieurs supports d'informations comme le son, les images et la vidéo, est le rêve des informaticiens [KAY 79] et le but du multimédia. Considéré comme un gadget par les uns et comme une révolution par les autres, le multimédia est sans aucun doute le concept à la mode. En fait, le multimédia est surtout une évolution technologique logique due à l'invasion du "numérique" dans tous les secteurs de l'information [FOX 91]. Bien qu'apparu dans les années 80 et annoncé comme devant bouleverser le domaine du traitement d'informations, les applications sont encore aujourd'hui peu nombreuses [STEF 90]. En effet, le multimédia se situe dans une période charnière ; les technologies s'affrontent et aucun véritable standard fédérateur n'apparait. Le multimédia est donc le challenge des prochaines années.

1.1. Concept du multimédia

Le terme de multimédia est souvent utilisé de manière floue et imprécise. Il n'est pas rare de voir des produits aussi différents que des outils de vidéoconférence, des jeux informatiques voire même des ordinateurs qualifiés de multimédias. Il convient donc de donner une définition de ce concept.

Un *média* est un moyen par lequel l'information est perçue, exprimée, stockée et transmise. Dans un système informatique, l'information est traitée de façon interne et échangée avec l'utilisateur. La représentation interne de l'information n'est pas liée avec la présentation qui en est faite à l'utilisateur et qui peut être réalisée par différents types de médias. Par exemple, un texte qui est généralement stocké sous forme d'une séquence de caractères ASCII peut être édité sous forme formattée ou être synthétisé et restitué sous forme de voix.

Le *multimédia* est un qualificatif[1] qui caractérise un document, un produit ou une application qui incorpore plusieurs types de médias : textes, graphiques, images, sons et vidéos ; le terme multimédia faisant référence aux divers moyens et supports utilisés pour rassembler ces différents types d'informations. Le MHEG (Multimédia and Hypermedia Information coding Expert Group) définit le mot multimédia comme la propriété de manipuler différents types de médias de représentation. Cet adjectif doit normalement être rattaché à un nom qui précise le contexte : service ou application multimédia, terminal multimédia, présentation multimédia. Cependant, le terme multimédia est souvent employé tel quel ; dans ce cas, il désigne le domaine d'activité au même titre que la CAO ou la PAO.

Les différents médias sont caractérisés par la quantité mais aussi la qualité des informations que chaque élément peut fournir à l'homme. Un texte est composé d'une suite de mots et de phrases qui formalisent des idées. Il transmet donc des informations précises. Un texte dactylographié véhicule moins d'informations que le même texte écrit à la main car la calligraphie de l'écrivain apporte une touche plus personnelle. Ce même texte parlé informe par le ton de la voix. Une image peut parfois remplacer un long discours (Lao-Tseu « une image vaut dix mille mots »). Le multimédia permet d'absorber des quantités d'informations plus importantes ou sous une forme plus significative car il fournit à l'usager la possibilité d'utiliser plusieurs de ses sens pour percevoir cette information. Il lui donne le pouvoir de choisir le type de média le plus adéquat pour représenter l'information. L'existence de nouveaux médias plus expressifs et plus conviviaux augmente le potentiel des applications et crée de nouvelles opportunités.

1.2. Applications multimédias

Avec l'introduction de produits spécialisés, surtout au niveau du stockage des informations, et de machines toujours plus puissantes, la technologie multimédia a fait son entrée dans le monde informatique. Conscients de l'étendue du marché, des constructeurs et des développeurs de plus en plus nombreux offrent des matériels et des logiciels "orientés multimédias". Cependant, les applications sont encore rares et portent sur des domaines spécifiques : la formation ou l'éducation, la maintenance, la promotion de produits, les catalogues, les agences de voyage ou immobilières, les bornes interactives d'information. A ceci s'ajoute tout ce qui est du ressort de la vidéo-transmission :

1. Le Robert définit le terme multimédia ainsi "multimédia: adj, qui concerne plusieurs médias, qui est destiné à la diffusion par plusieurs médias."

visiophone, téléconférence et vidéoconférence, messageries vocales [LIPP 90], [ROBI 90], [VIN 91].

Ces nouveaux types d'applications bénéficient de la combinaison des médias informatiques "traditionnels" avec la technologie du son et de la vidéo. Par exemple, un didactitiel est capable de montrer des séquences documentaires et d'inclure des enregistrements sonores sur le sujet étudié ; un système de publication peut inclure et combiner des textes et des graphiques ou même des images animées. L'apport déterminant du multimédia est surtout l'interactivité. Sans elle, le mixage texte-son-images ne serait guère différent de la bande vidéo. L'utilisateur d'un produit multimédia n'est pas un spectateur passif ; c'est lui qui commande, qui choisit ce qu'il veut voir. Un véritable dialogue peut s'engager. C'est pourquoi la plupart des applications aujourd'hui appartiennent au domaine de la formation et de l'information mais aussi des jeux [LESU 90].

Il est évident que le multimédia ouvre des horizons nouveaux à la fois aux constructeurs et aux développeurs, mais change aussi la vision et l'utilisation de l'ordinateur par l'homme. Le propre du multimédia sera de générer de nouveaux types d'applications auxquelles il devra s'adapter et avec lesquelles il va évoluer. La messagerie électronique est l'une des nombreuses applications qui entendent bénéficier de ses apports.

1.3. Messagerie multimédia

1.3.1. Présentation

La messagerie multimédia est d'abord une messagerie électronique. La messagerie électronique est un moyen de communication qui permet d'émettre et de recevoir des messages entre des personnes ou des applications. Généralement, un système de messagerie est construit sur un réseau d'ordinateurs qui assure le transfert des informations brutes entre machines. Conceptuellement, il comprend des éléments qui assurent le transport de messages et d'autres qui fournissent des services de messagerie spécifiques à l'utilisateur comme le courrier inter-personnel. Pour la bonne coopération entre ces composants, chaque système est régi par des protocoles de communication. Une dizaine d'années de recherche et de développement dans la communauté informatique ont conduit à de nombreux systèmes opérationnels comme celui de la messagerie Internet.

La messagerie électronique connaît depuis quelques années un développement croissant ; elle est devenue un support de communication privilégié des entreprises et des organismes de recherche. Fortement utilisée aux Etats-Unis, l'Europe et les autres pays ont compris l'intérêt d'un tel outil de communication au point que les réseaux publics et privés se multiplient. Cet engouement est dû au fait que la messagerie électronique possède de nombreux avantages par rapport au téléphone ou au fax. Elle combine les avantages du courrier par lettre (étant un moyen de communication asynchrone, les deux correspondants ne sont pas tenus d'être présents en même temps ; ceci est essentiel quand on sait qu'un appel téléphonique sur deux n'aboutit pas car le correspondant est absent ou occupé) avec la rapidité et les outils associés au traitement par ordinateur (les messages peuvent être triés, stockés, modifiés ou envoyés à d'autres destinataires.) Enfin, le même

message peut être envoyé simultanément à un grand nombre de personnes dispersées géographiquement.

Mais la messagerie n'est pas qu'une messagerie entre personnes, elle est de plus en plus utilisée entre applications comme moyen de communication. L'échange de données informatisées (EDI) qui vise à transmettre des documents essentiellement commerciaux en est un exemple [HAYE 90]. Une autre forme de communication est celle réalisée entre l'ordinateur et l'utilisateur dans le but d'obtenir des services. Ainsi, il est devenu habituel pour un utilisateur de demander des informations à des serveurs par le biais de la messagerie. Là encore, les avantages sont nombreux : réponse et traitement automatique, gain de temps, etc.

Au départ, la messagerie était utilisée comme un moyen de communiquer des messages composés de textes. Le plus souvent, ces messages remplaçaient les notes et les mémos et avaient un but uniquement informatif. De plus en plus, ce type de communication électronique est employé pour échanger des informations complexes ou spécifiques. L'arrivée du multimédia a naturellement conduit à vouloir intégrer d'autres types de données dans les messages.

1.3.2. Caractéristiques

La messagerie multimédia, se distingue de la messagerie électronique classique par le fait qu'elle permet la conception et la transmission de messages composés de différents types de médias comme :

- du texte (sous forme ASCII, formatté),
- des graphiques 2D ou 3D,
- des sons, musiques et voix (digitalisés ou destinés à des synthétiseurs),
- des images fixes N/B ou couleur (réelles ou synthétiques),
- des images animées couplées ou non avec du son,
- des fac-similés,
- des données structurées (ex : feuilles de tableurs),
- des documents structurés, etc.

Cette liste n'est évidemment pas exhaustive, d'autres formes de représentation d'informations peuvent être envisagées. Tout message multimédia véhiculant des informations représentées sous une forme autre que du texte non formatté est donc considéré comme multimédia. Cependant, un message multimédia est souvent beaucoup plus complexe, il utilise des types de médias divers qui peuvent être combinés ou imbriqués les uns dans les autres.

1.3.3. Exemple de message multimédia

Prenons l'exemple de ce qu'un message multimédia peut contenir (cf FIGURE 1). C'est une extrapolation d'un exemple concret, un courrier d'informations envoyé aux futurs participants d'un congrès au Japon.

Chaque intéressé reçoit environ dix feuilles d'explications contenant :

- une lettre de présentation en anglais,
- le plan de l'immeuble où a lieu le congrès,
- des informations sur le voyage avec des exemples de phrases en anglais et en japonais pour s'orienter,
- une sorte de traduction phonétique pour la prononciation,
- une fiche des horaires des trains,
- une présentation de la ville avec textes explicatifs, plans et photos.

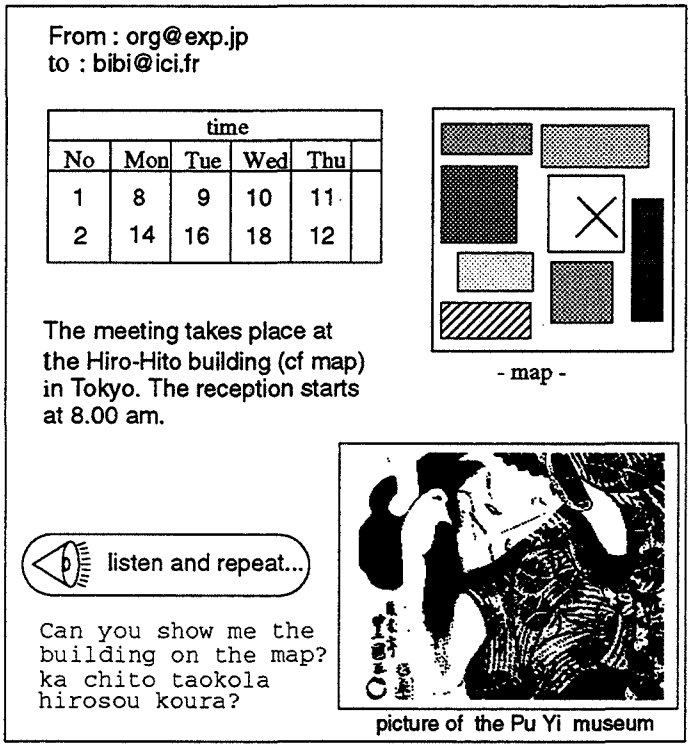


FIGURE 1 *: exemple de message multimédia*

Il est facile d'imaginer un message multimédia rassemblant ces différentes informations. Un tel message contient divers types de médias de présentation. Les renseignements d'ordre général sont formalisés par des textes en anglais et en japonais utilisant ainsi des caractères alphabétiques et idéographiques. Des graphiques permettent de représenter les plans et les tableaux, et des images correspondent aux photos. Les phrases de prononciation sont remplacées par une voix, plus expressive et plus facile à

apprendre. On peut aussi concevoir une forme plus évoluée de ce message, par exemple, inclure ou non le son pour la prononciation phonétique, prévoir l'affichage du message en une autre langue selon le choix du destinataire, fournir les horaires d'avions selon le pays d'origine, etc. Un tel message peut être composé et envoyé, sous forme personnalisée, à plus de cinq cents personnes en quelques heures, ce qui ne peut être fait par fax. Chaque participant peut ensuite ajouter ses propres notes, imprimer tout ou partie du message ou travailler sa prononciation. Si un tel message est conceptuellement réalisable, il ne peut, à l'heure actuelle, être reçu dans sa forme originelle par l'ensemble des destinataires que si ceux-ci disposent des spécificités matérielles et logicielles requises par la messagerie multimédia.

2. ENVIRONNEMENT MULTIMÉDIA

Le concept du multimédia met en effet en oeuvre une multitude de technologies : des matériels spécifiques de traitement et de stockage des informations, des logiciels de manipulation de données multimédias, le transport de ces données, leur compression et leur restitution [ADIE 93]. L'introduction des techniques de traitement du son et des images pose des problèmes d'architecture et entraîne de nouveaux besoins au niveau matériel et logiciel. De nombreuses recherches ont conduit au développement de matériels spécifiques de plus en plus performants (machines dédiées au multimédia, supports de grande capacité) qui sont nécessaires à toute application multimédia. Parallèlement, des logiciels multimédias se sont développés dans des domaines comme la composition de document (par exemple l'environnement multimédia QuickTime [WAYN 91]) ou de vidéo. Cependant, les logiciels pour gérer les données multimédias sont encore peu nombreux et rarement compatibles entre eux. Il en est de même pour les formats de représentation et transmission des données. Si tout ceci reste vrai pour toute application multimédia, le problème est particulièrement crucial dans le cas de la messagerie qui ne peut fonctionner véritablement sans des standards bien définis à cause des environnements hétérogènes des utilisateurs [HUIT 89] [ROUD 90]. Examinons les aspects techniques et logiques de l'environnement multimédia tel qu'il se présente aujourd'hui dans le contexte de la messagerie électronique.

2.1. Contraintes techniques

2.1.1. Besoins matériels

La messagerie multimédia, comme toute application multimédia, nécessite un matériel adéquat. De façon concrète, un système multimédia complet peut idéalement être schématisé par la réunion d'un ordinateur, d'une télévision, d'un magnétoscope, d'une caméra, d'une chaîne HI-FI, d'un fax, d'un modem et de tous les outils permettant de manipuler et mixer des images, des sons et des données sous forme numérique ou analogique. Les fonctions de la machine sont ainsi étendues pour gérer tous les type de données multimédias (cf FIGURE 2).

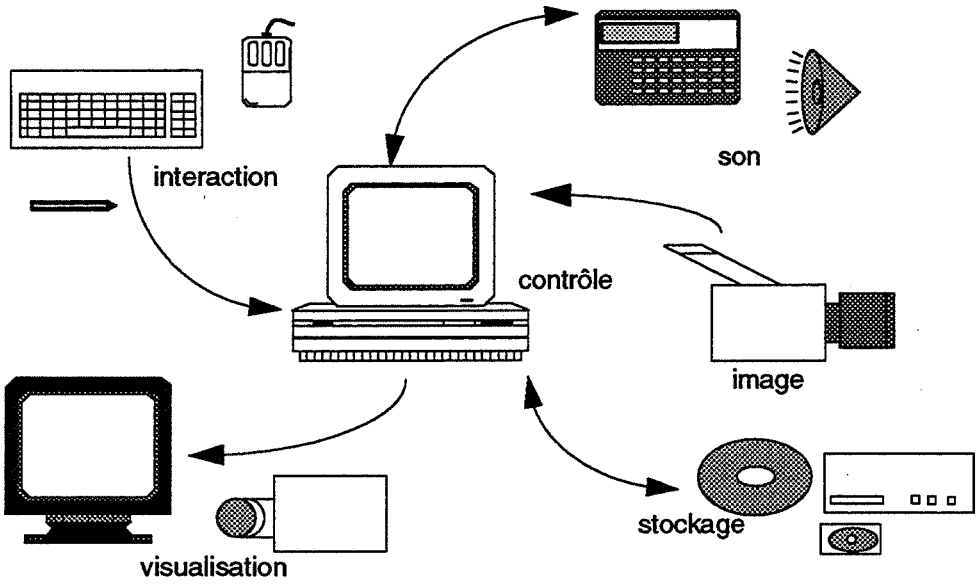


FIGURE 2 : *station multimédia type*

Un bon environnement multimédia nécessite donc un certain nombre de capacités techniques. Ainsi, les architectures des machines doivent offrir de nouvelles possibilités comme :

- des composants spécialisés de traitement du signal capables de réaliser des opérations en temps réel et des sorties à haut débit, en particulier pour la voix,
- des techniques logicielles ou matérielles de compression/décompression pour réduire l'espace occupé par les fichiers de données (sons et images),
- des systèmes permettant de stocker de gros volumes d'informations avec des temps d'accès très rapides,
- une vitesse de CPU adaptée.

De plus, si l'on veut manipuler des documents contenant des voix, des images ou des fac-similés, il est nécessaire d'avoir à sa disposition les périphériques d'entrée/sortie pour gérer ce type de média. Parmi ceux-ci, on peut citer :

- un affichage de haute définition avec un environnement multi-fenêtres,
- un dispositif sonore intégré ou relié au système (par exemple microphone, haut-parleur, téléphone),
- un scanner ou une caméra vidéo pour saisir des images,
- un fac-similé, une imprimante,
- des dispositifs d'interaction (clavier, souris...).

2.1.2. Matériels disponibles

Il existe déjà des machines orientées vers le multimédia. Dans le monde de la micro-informatique, on peut citer le standard *MPC* (Multimedia Personal Computer) de Microsoft qui précise les éléments de base que doivent intégrer les plate-formes classiques des micro-ordinateurs pour supporter des applications multimédias. De son côté, IBM propose le *PS/2* qui permet l'acquisition et la manipulation d'images vidéo. Chez Apple, les *Macintosh* disposent en standard des outils autorisant un affichage graphique de qualité et la manipulation du son depuis 1984 [NOEL 90]. Dans le monde des stations de travail, la *Sparcstation*, comme le *Macintosh*, est équipée de convertisseurs A/D et D/A 8 bits pour manipuler des données audio. Plus élaborée, la station *NeXT* de Steve Jobs [GARR 90] inclut en standard un processeur de traitement du signal permettant d'associer des fonctions de traitement d'images et de sons de haut-niveau. La station multimédia de Commodore, l'*amiga 4000*, est dotée elle aussi de possibilités sonores et graphiques avancées grâce à la présence de processeurs spécialisés [BOUN 92]. Étant l'une des machines les moins chères du marché, on la retrouve dans de nombreuses réalisations professionnelles.

Certains matériels bon marché peuvent être intégrés sur des PC pour restituer des sons, manipuler des données MIDI (Musical Instrument Digital Interface) ou gérer les images vidéo [KIM 91]. Intel a annoncé le processeur vidéo i750 pour les applications multimédias [HARN 91]. Enfin, il faut noter que des recherches sont en cours pour concevoir une station multimédia européenne répondant à des soucis de standardisation [MORE 90].

2.1.3. Volume des informations

Les messages multimédias comme toutes les informations multimédias sont très volumineux dès que l'on intègre des données comme le son ou la vidéo. Par exemple, un méga-octet d'espace correspond à six secondes de son de qualité CD, à une image couleur 640x480 sur 24 bits/pixel non compressée ou à une simple mémoire de trame vidéo d'1/ 30ème de seconde. La figure suivante illustre l'impact du multimédia sur le volume des informations (cf FIGURE 3).

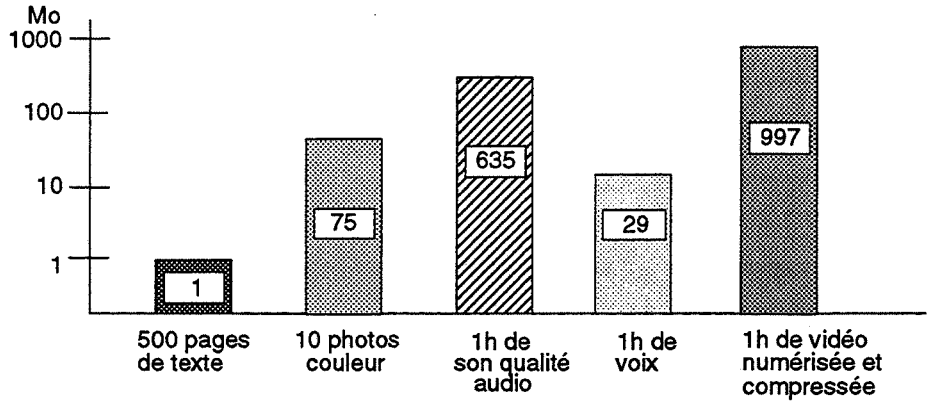


FIGURE 3 *: impact du multimédia sur le volume des informations*

La manipulation de tels volumes entraîne naturellement des problèmes de stockage et de transmission de données.

- Stockage

Les applications multimédias exigent des conditions de stockage et d'accès aux données particulièrement efficaces en raison de l'utilisation conjointe d'images, de textes, de sons et d'animations vidéo. Pour enregistrer les messages multimédias ou les objets les composant, il est nécessaire de disposer de supports adaptés. Bien que les CD-ROMs ou les disques optiques permettent de stocker un plus grand nombre d'informations, ceci est parfois insuffisant (un CD-ROM enregistre environ 70 minutes de son stéréo mais seulement 30 secondes de vidéo). Pour faire face à ces besoins, de nouveaux supports apparaissent comme le *CDI* (CD Interactif) ou le *CD-ROM XA* (Extended Architecture) promulgué par Philips et Sony. Ce dernier en unifiant les standards du CD-ROM et du CD-I peut enregistrer jusqu'à 16 heures de musique. De plus, les temps d'accès aux CD-ROMs deviennent de plus en plus rapides (220 ms pour les CD-ROMs de Toshiba), ce qui contribue à améliorer l'interaction de certaines applications. [Multi 93].

- Transmission

La taille des messages requiert nécessairement des réseaux de plus en plus rapides avec des vitesses de transmission allant de 100 kb/s à plusieurs mégabits/s. On s'oriente donc vers des réseaux à haut débit. L'objectif est d'obtenir une véritable intégration des technologies, soit un réseau capable de transporter à la fois des informations, de la voix, des données et des images. Les réseaux RNIS ont été développés dans cette optique, mais bien qu'ils permettent la transmission de toutes sortes de données, ils n'offrent pas actuellement des vitesses suffisantes pour les volumes dont on envisage la transmission. Une solution est le réseau numérique à large bande nommé B-ISDN (Broadband ISDN) qui devrait permettre de supporter un grand nombre de médias à des taux constants ou variables [POPE 90]. B-ISDN est un réseau WAN (Wide Area Network) qui peut transmettre non seulement de la voix et des données mais aussi des images animées sur des canaux de 150 à 600 Mbps. Il est en général couplé avec ATM (Asynchronous Transfer Mode), qui offre une interface universelle [BOUD 92]. Cette technique semble la plus apte à transporter des données informatiques ou téléphoniques. B-ISDN, de même que la technologie ATM, font actuellement l'objet de nombreuses recherches et devraient être opérationnels, vers le milieu des années 90.

- Compressions /décompressions

Pour pallier les problèmes de volume engendrés par les nouveaux types de médias, plusieurs solutions existent. La première est de traiter ces types de médias par des matériels externes et spécialisés. La seconde est d'utiliser des techniques de compression.

Ces algorithmes de compression/décompression sont propres à chaque type de média et doivent être pris en compte lors de la transmission et de la réception des messages. De nombreuses techniques sont disponibles et donnent de très bons résultats. D'un rapport de 1 à 2 pour le texte, la compression peut aller de 1 à 50 pour une image. Malheureusement, ces techniques ont pour inconvénient une perte de temps quelquefois importante et le risque d'altération des informations. Un compromis est d'utiliser des algorithmes

asymétriques qui réalisent des compressions lentes mais des décompressions en temps réel. Il y a actuellement divers systèmes de compression/décompression disponibles sur le marché : *DVI* (Digital Video Interactive) et Indeo (standard de compression/ décompression vidéo) d'Intel [VEZI 91] et le *CDI* de Philips. DVI utilise un algorithme de compression de mémoires de trames vidéo qui permet le stockage de plus d'une heure de vidéo plein écran sur un CD. Avec Indeo couplé avec un processeur i750, les films sont visualisés en mode plein écran à la cadence de 30 images par seconde.

Mais même en appliquant des algorithmes de codage de sons ou de voix ou des techniques de compression d'images, il n'en reste pas moins que les messages multimédias sont, en général, de 10 à 200 fois plus volumineux que des messages textes.

2.2. Contraintes logiques

Une messagerie électronique se doit d'être employée par toute une communauté d'utilisateurs. Outre les contraintes matérielles, s'ajoutent des contraintes logiques : standards, logiciels appropriés, compatibilités des systèmes de messagerie, transmission des messages. Comme nous allons le voir, nous sommes loin à l'heure actuelle, d'une situation normalisée [BORE 91a]. Tous ces problèmes devront cependant être résolus si l'on veut assurer une parfaite cohérence des messages et la fiabilité des communications.

2.2.1. Représentation des messages

- Besoins de normalisation

A l'heure actuelle, il n'existe pas de norme pour représenter les messages multimédias et ce manque de standard retarde le développement de la messagerie multimédia. Sans un tel format, il n'est pas possible d'assurer aux utilisateurs une quelconque compatibilité entre leurs outils de messagerie. Le problème est difficile car il existe une grand choix de formats existants sans qu'aucun ne soit suffisamment complet et reconnu. D'un côté, les organismes de normalisation tentent de définir des formats standards, de l'autre, de nombreux vendeurs offrent leur propre format de représentation [WILS 92] dans l'attente d'une quelconque uniformisation. Aussi, on assiste actuellement à deux courants différents. Le premier promulguant un standard de haut-niveau style *ODA*, *HyTime* ou *Postscript,* comme standard de message multimédia. Le second préconise l'approche inverse ; c'est-à-dire choisir des standards de bas-niveau existants comme standards de fait et de les utiliser comme base de départ. De plus, il est possible que la création d'un format adéquat s'avère prématurée tant que l'on ne connaît pas tous les champs d'application possibles.

Quelque soit la solution adoptée, un standard multimédia doit permettre d'identifier le type de message (multimédia ou non) et les différentes parties de messages ; par exemple, savoir où la voix digitalisée commence et finit, extraire une image dans une suite animée. Il doit aussi permettre de représenter le contenu de chaque partie de message et prendre en compte les formats existants formels ou de facto et les contraintes pragmatiques des systèmes de transfert de messages existants. Il doit permettre de définir tous les types

d'objets, de fournir des mécanismes pour représenter leurs relations et d'être suffisamment ouvert pour pouvoir ajouter d'autres types d'objets [KUO 83].

- Possibilités actuelles

On distingue les formats d'architectures de documents qui déterminent la structure et l'aspect général du document (ODA, RTF, RIFF), les formats de notation associés aux contenus qui peuvent être mélangés et utilisés dans différentes architectures et les formats plus organisationnels qui fonctionnent de la même manière quels que soient les formats d'architecture ou de contenus employés (HyTime) (cf FIGURE 4). Bien que des études soient en cours, aucun ne satisfait actuellement toutes les exigences d'un standard de messages multimédias.

Parmi les standards de haut-niveau envisagés, on trouve les formats suivants :

- *ODA* (Office [Open] Document Architecture) définit une architecture de documents qui permet la spécification d'une structure logique (l'organisation logique du document) et/ou sa structure physique (comment le document est représenté). L'architecture ODA permet d'intégrer de nouveaux types de médias. Actuellement, elle supporte du texte structuré, des graphiques géométriques et des rasters[1]. Des extensions sont envisagées pour intégrer des nouveaux médias (audio, vidéo), des informations de couleur et de sécurité ainsi que des aspects temporels [ODA 88].

- *Postscript* est un langage de définition de page pouvant décrire du texte, des images et des graphiques, mais le son n'est pas inclus.

- *HyTime* (Hypermedia/Time-based document structuring language) est un langage construit sur SGML (Standard Graphical Markup Language, métalangage normalisé pour la création de document s'appuyant sur des modèles de document-type). Il est proposé comme standard pour représenter la structure de documents multimédias, hypertextes et hypermédias basés sur le temps et l'espace [NEWC 91]. HyTime standardise les liens et la synchronisation des portions des documents hypermédias et leurs composants d'informations multimédias. Il ne concerne pas l'architecture du document, les notations de contenus ou le codage des objets [GOLD 91].

- *MHEG* (Multimedia and Hypermedia Information coding Experts Group) est lui aussi un format pour la représentation des documents hypermédias destiné aux applications interactives multimédias. Il spécifie les objets multimédias manipulés et fournit des règles pour leur structuration et leurs hyperliens.

Ces standards de haut-niveau sont, pour la plupart, des standards essentiellement organisationnels. Ils décrivent comment assembler les différentes pièces mais les éléments sont toujours codés selon une collection de standards sur lesquels il n'y a pas de consensus. En effet, pour la représentation des types de médias (supports et contenus), on est confronté à un problème de choix. Si pour certains types, des standards semblent émerger, pour d'autres, il reste encore trop de possibilités.

1. Le format raster est un codage par balayage de l'ensemble des points d'une image.

• Pour les parties textes, on trouve de nombreux langages de composition tels que *Troff, Tex* et *LaTeX, SGML, ODA/ODIF.* Aucun d'entre eux ne peut être considéré comme un standard de facto. En effet, *Troff* n'existe que dans le monde Unix, *TeX* est un produit de domaine public mais est peu utilisé hors du cadre universitaire, et il existe peu d'implantations de la norme *ODA.* De plus, un certain nombre de constructeurs définissent leur propre format comme *RTF* de Microsoft.

• Pour les fac-similés, une standardisation *(G3* ou *G4)* existe pour les images noir et blanc, incluant un algorithme de compression mais il n'y a pas d'équivalence pour les images couleur ou à niveaux de gris. Un nouvel algorithme de compression pour images binaires JBIG est à l'étude pour remplacer G3 et G4.

• Pour les parties audio, il existe des algorithmes standardisés utilisant le standard *ADCPM* (ADaptative Pulse Code Modulation, G721) qui définit un codage de données avec compression. Une autre solution est l'utilisation de la norme *MIDI* qui permet la communication par messages entre instruments de musique et le transfert de fichiers MIDI [ARTO 87]. D'autres formats de représentation et de codage existent comme SMML (Standard Music Markup Langage), ou G722 (codage audio 7Khz) et de nouveaux standards sont à l'étude (H200 pour la compression des sons).

• Pour les images fixes, de nombreuses représentations publiques ou propriétaires sont utilisées *(TIFF, GIF, ODA, X Bitmap, SUN rasterfile).* De même, pour les images vidéo, un grand nombre de codage analogiques *(PAL-SECAM, NTSC),* ou digitaux existent *(DVI, CD-I),* [LIEB 91]. Un début d'uniformisation a conduit à trois normes complémentaires. La norme *JPEG* (Joint Photographic Experts Group) définit un standard de compression des images fixes plus un format d'échange. La norme *MPEG* (Moving Pictures Experts Group), récemment approuvée par l'ISO (International Standard Organisation), s'applique à l'image animée en proposant un standard de compression audio et vidéo [WALL 91] [GALL 91]; c'est la norme la plus importante par sa polyvalence et son champ d'application. Un nouveau standard MPEG II pour la vidéo de haute qualité est en cours de spécification. Enfin, la norme *H261,* fonctionnellement comparable à la précédente, doit répondre aux besoins des applications de type visiophone et vidéoconférence. Enfin, IIF (Image Interchange Facility) permet de définir un format pour échanger des images quelconques.

• Pour les graphiques, *GKS* (Graphical Kernel System), ainsi qu'*ODA* ou *Postscript* permettent de définir des figures et des graphiques. Mais, là encore, il existe de nombreux formats propres aux logiciels employés *(CGM, XPM).*

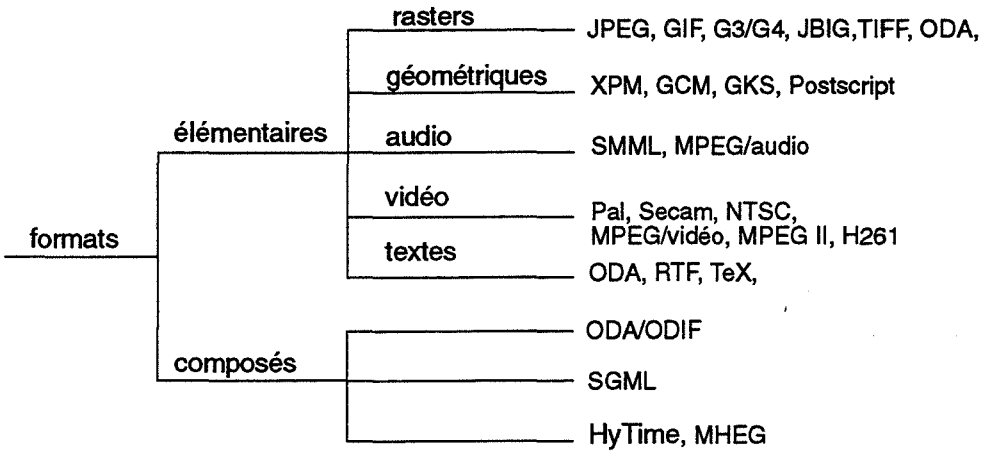


FIGURE 4 : *récapitulatif des principaux formats en cours*

2.2.2. Composition et restitution de messages

La messagerie multimédia implique la création de différentes parties de message, leur assemblage et leur restitution dans un message. Une telle composition peut être réalisée, soit par un éditeur multimédia combinant les fonctions d'éditeurs spécifiques, soit en créant les différentes parties avec des outils correspondant au type de média puis en les assemblant dans un document unique. La première solution permet d'avoir un outil intégré mais la plupart du temps incompatible avec d'autres systèmes. La deuxième solution se heurte à un trop grand choix de logiciels ou, au contraire, à un manque d'outils de composition. En effet, si dans le domaine du traitement de texte et du traitement d'image, on trouve de nombreux outils performants, il n'existe pas de consensus sur leur utilisation et ceux-ci ne sont pas toujours compatibles entre eux. Inversement, de nombreuses recherches ont été réalisées pour le traitement du son mais les outils correspondants sont encore peu disponibles et nécessitent un matériel très spécifique. Si les stations de travail offrent certaines possibilités audio au niveau du matériel, il existe peu de logiciels de qualité pour les exploiter. Dans les deux cas, l'interface utilisateur doit permettre de créer et assembler les différentes parties du message en fournissant des possibilités de traitement de texte, sons et images.

2.2.3. Transmission des messages

La transmission des messages relève de la responsabilité de logiciels particuliers. Ceux-ci doivent être capables d'encapsuler dans un message unique le contenu multimédia du message et les informations de l'en-tête et assurer que ce message sera transmis de telle façon que le destinataire comprenne ce que l'expéditeur voulait exprimer. Deux types de problèmes se posent. Premièrement, certains éléments de messagerie responsables du transfert des messages sont limités au niveau de la taille maximale des messages. Cette limite peut s'avérer trop restrictive d'où le risque d'avoir des messages tronqués ou même impossibles à transmettre. Deuxièmement, pour pouvoir reconnaître et traiter des

messages non-textes, il est nécessaire de définir un format de transmission qui prennent en compte le type des données (format texte ou binaire). Sans de tel format standard, il est impossible d'envoyer des messages multimédias. A l'heure actuelle, la seule façon d'assurer l'envoi d'un message multimédia est de coder le message en 8 bits, tout en sachant que cela n'est pas sûr à 100% et de fournir au destinataire le mode d'emploi pour lire le message. Ceci n'est évidemment pas acceptable dans le contexte d'une messagerie largement utilisée. Le problème est donc de savoir comment procéder pour transmettre sans perte d'informations un message multimédia. Parmi les familles de messageries, les plus utilisées (SMTP et X400) travaillent à résoudre ce problème en proposant des standards d'adressage et de formats [HAYE 92].

- Protocole SMTP (RFC 822, 1049 et 1154)

SMTP (Simple Mail Transport Protocol) est le protocole de messagerie utilisé par la plupart des utilisateurs dans le monde Unix. Bien qu'il soit le plus répandu, son ancienneté pose certains problèmes. Des recherches récentes ont conduit à la définition de protocoles chargés de prendre en compte les nouveaux besoins engendrés, entre autres, par le multimédia.

RFC 822 définit un protocole de représentation de messages qui spécifie des détails importants sur les en-têtes de messages et en particulier les informations de type "content-type" qui permettent de connaître le type de contenu des messages. Ce protocole ne définit pas les contenus de messages ou corps de messages qui sont considérés comme du texte US-ASCII. C'est le but de MIME (Multipurpose Internet Mail Extensions) qui a été adopté par l'Internet pour représenter les messages multimédias [RFC 1341]. L'idée de ce format est relativement simple : standardiser la façon dont les utilisateurs marquent les parties d'un message multi-parties et les laisser décrire le format des données de chaque partie dans les en-têtes. Ceci est réalisé par deux spécifications d'entêtes pour chaque partie de message, l'une indiquant le type de contenu, l'autre le type de codage. La première se présente comme un type de contenu général puis un nom spécifique au type de contenu et d'éventuels paramètres auxilaires (ex : text/richtext). La spécification de codage est théoriquement indépendante du contenu et indique quelles transformations doivent être faites pour représenter le corps d'une façon acceptable pour le transport. Si de nombreux codages ne sont pas désirables, on peut définir de nouveaux types de contenus selon les besoins. Dans le but d'assurer que l'ensemble des valeurs (couples content-type/sous-type) soient développées d'une façon cohérente et bien spécifiée, MIME définit un processus d'enregistrement qui utilise IANA (Internet Assigned Number Authority) comme un registre central pour de telles valeurs.

MIME redéfinit ainsi le format des corps de messages pour permettre de représenter des messages non textuels sans perte d'informations. Il est conçu pour fournir des facilités pour inclure des objets multiples dans un message unique, pour représenter les corps dans des caractères autres que l'US-ASCII, pour représenter les messages textes multi-polices, les images et les portions audio et plus généralement pour faciliter des extensions futures en définissant de nouveaux types de messages. Un document associé, RFC 1342, étend les champs d'en-têtes Internet pour leur permettre de contenir des données autres que du texte US-ASCII [RFC 1342].

Le développement de MIME offre de nouvelles opportunités aux systèmes de messagerie Internet. En effet, il a l'avantage d'être extensible, de permettre d'avoir des formats alternés, des parties de messages externes, des messages multi-parties et multimédias sans un gros investissement ; aussi, de plus en plus d'agents utilisateurs de messagerie offrent des extensions MIME (Mail, elm, Andrew). Néanmoins, si MIME permet de construire des messages multi-parties multimédias en fournissant un mécanisme d'en-têtes appropriées, il ne prend pas en compte la structure du message et les relations complexes entre les types de médias.

- Norme X400

X400 est un standard conçu pour résoudre certains des problèmes posés par SMTP. Son but est de fournir un service international pour l'échange de messages électroniques sans restriction sur le type des informations codées véhiculées. Cette norme a été proposé en 1984 par le CCITT sous forme d'une série de recommandations qui définissent un système de messagerie en mode "enregistrement et retransmission" (Store and Forward). Ce système de messagerie est souvent appelé X400 MHS (Message Handling System) ou messagerie X400. Cette norme définit un service de transfert de messages et un service de courrier inter-personnel. Elle explicite aussi le format des messages véhiculés. Le standard originel a été étendu en 1988 et 1992 pour inclure des formats pour des messages non textes.

Le système de messagerie X400 comprend, entre autres, des ensembles d'utilisateurs, d'agents utilisateurs (UA) et d'agents de transfert de messages (MTA) (cf FIGURE 5).

• Un agent utilisateur est un processus d'application qui permet de préparer, émettre et recevoir des messages. L'UA dialogue avec un système de transfert de messages MTS (Message Transfer System) ou un système de stockage et leur soumet des messages provenant de l'expéditeur.

• Le MTS remet les messages qui lui ont été soumis aux destinataires, aux unités d'accès et/ou aux unités de stockage. Le MTS est composé d'un ensemble de MTA qui coopèrent pour transférer et remettre un message aux destinataires spécifiés. Le message est transféré de proche en proche depuis le MTA de l'expéditeur jusqu'à celui du destinataire.

Le service de courrier inter-personnel est défini au dessus du MTS pour fournir le service de courrier classique. Les messages inter-personnels sont véhiculés par les messages du MTS et sont traités comme leurs contenus. Un tel message est composé d'une en-tête et d'un corps. L'en-tête correspond à un ensemble de champs/attributs tels que "destinataire", "sujet". Le corps contient l'information que l'utilisateur veut communiquer. Il se compose d'une séquence de "parties de corps" (body-part), chacune pouvant être de type différent : texte IA5, voix, teletex. Un identificateur indique la nature du body-part. Certains identificateurs sont déjà standardisés, d'autres peuvent être spécifiés pour des applications particulières ou pour des usages privés. La norme transmet des messages codés en binaire ; il est ainsi plus facile d'envoyer des messages non textuels même s'ils doivent être ré-encodées pour la messagerie SMTP.

■ Perspectives d'intégration

A l'heure actuelle, on assiste à un effort d'uniformité des deux côtés. Ainsi, pour encourager l'utilisation de la norme X400 dans le monde Internet, un projet est en cours pour construire les bases nécessaires pour utiliser la norme dans l'environnement Internet. Il devrait spécifier les détails d'implantation pour un grand nombre de type de messages binaires X400 de façon à ce que les utilisateurs X400 ou Internet puissent recevoir et envoyer des messages généralisés sans se préoccuper de leur transfert à travers les réseaux. Il est envisageable que dans un futur proche, des passerelles appropriées permettront la transmission de messages quelconques entre SMTP et X400, en utilisant les spécifications de format binaires développées par le projet Internet et les extensions MIME.

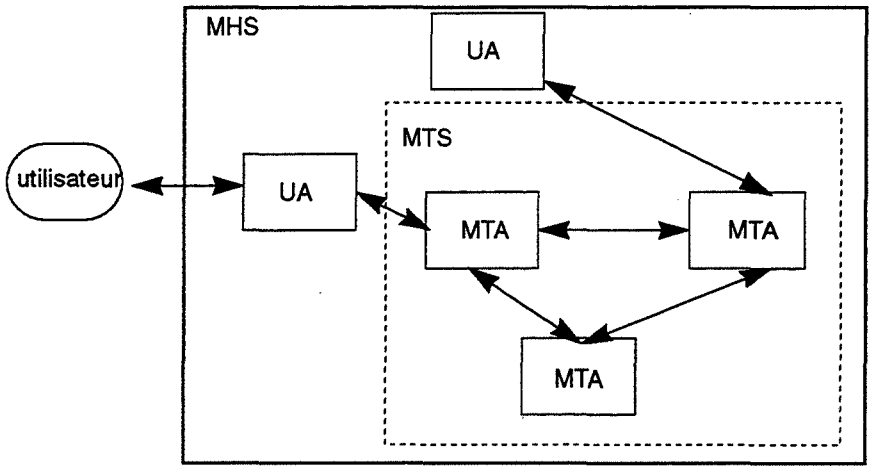


FIGURE 5 : *système de gestion de messages X400*

CHAPITRE 2

État de l'art

Parallèlement à l'essor du multimédia et de ses applications, la messagerie multimédia s'est progressivement développée ces dernières années. Les problèmes inhérents au multimédia et à la messagerie électronique ont cependant ralenti ce développement. Nous allons voir dans ce chapitre, quelques systèmes existants et les concepts mis en oeuvre pour leur utilisation. A partir de l'étude de ces divers systèmes, nous présenterons les besoins et les objectifs que nous nous sommes fixés.

1. SITUATION

La messagerie multimédia connaît à l'heure actuelle un essor important. Cet intérêt croissant est dû à plusieurs facteurs. La raison principale est que l'on peut transmettre des messages plus compréhensibles par l'utilisation de nouveaux types de médias. La combinaison du texte, des graphiques et du son augmente le potentiel des informations transmises et permet d'exprimer plus clairement des concepts. En effet, des études montrent que si une personne se souvient de moins de 25% des informations qu'elle a simplement écoutées, elle assimile plus de 45% des mêmes informations lorsqu'elle les a vues et écoutées. Elle est plus réceptive à un message en couleur qu'à un autre en noir et blanc, et le sera encore plus si le message est accompagné d'explications sonores. Outre une meilleure assimilation des messages, la messagerie multimédia offre un aspect plus attractif et permet ainsi d'atteindre un plus grand nombre d'utilisateurs. En effet, chacun peut choisir le ou les types de médias qui lui sont le mieux adaptés. Ainsi, il peut s'avérer plus facile et plus rapide d'envoyer un message vocal que de l'écrire ; le développement des messageries vocales en est la preuve. Enfin, l'apport du son et des images ajoute une note plus expressive aux messages. Il est agréable de recevoir des graphiques ou des images ou de connaître la voix de son interlocuteur.

La transmission d'informations supplémentaires dans les messages accroit donc considérablement les possibilités de la messagerie. L'utilisation de nouveaux médias permet d'enrichir les communications inter-applications et entre personnes et ordinateurs. Les applications peuvent échanger des documents complexes incluant d'autres types de médias : textes, graphiques, images et sons. L'obtention de services est simplifié et enrichi par ces nouveaux supports d'informations. A de simples questions, l'utilisateur peut obtenir des renseignements de types variés, tout en étant guidé par des interactions. L'aspect dialogue entre utilisateur et prestataire de service facilite des manipulations qui, à l'heure actuelle, sont souvent fastidieuses. La messagerie multimédia répond donc à un besoin au niveau utilisateur. L'apport du multimédia va entraîner sans aucun doute une utilisation encore plus importante de la messagerie électronique dans les années qui viennent.

En attendant, les systèmes de messagerie multimédias sont à l'heure actuelle peu répandus et faiblement utilisés. En effet, leur développement a été ralenti par les problèmes que nous avons déjà évoqués, problèmes inhérents au domaine du multimédia (nécessité d'éléments techniques, volumes importants à gérer) ou spécifiques à la messagerie (manque de formats standards, problèmes de transmission de données). Si matériellement, les aspects techniques sont pratiquement résolus, des défauts d'uniformisation subsistent au niveau logiciel. Ces problèmes devront être résolus avant d'offrir la possibilité aux personnes et aux applications d'échanger des messages complexes dans des environnements hétérogènes. Le premier impératif sera de trouver un standard adéquat et c'est ce à quoi s'emploient les différents organismes de standardisation. Mais ce processus, long et incertain, ralentit le développement de nouveaux logiciels de messagerie multimédia. Ceci explique en partie le petit nombre de systèmes existants.

Néanmoins, il existe déjà depuis quelques années des produits de messagerie multimédia. Parmi les premiers systèmes, on peut citer les deux prototypes que sont *Andrew* et *Diamond/Slate*. Ces systèmes ne sont pas les seuls; des systèmes expérimentaux *MMS (Multimedia Mail System)* ou commerciaux (*cc:Mail, QuickMail*), ont vu le jour. Ces divers produits sont encore peu utilisés et uniquement par des communautés distinctes d'usagers car ils sont incompatibles entre eux. Pour pallier ce problème, on assiste au développement d'outil comme *Metamail*. Nous allons présenter rapidement les caractéristiques de quelques-uns de ces systèmes.

2. SYSTÈMES EXISTANTS

2.1. Andrew

Le projet Andrew est issu de la collaboration d'IBM et de l'Université de Carnegie-Mellon (CMU) dans le but de fournir un environnement de communication dans le domaine universitaire. C'est un ensemble complet d'outils informatiques pour éditer des documents, créer des applications ou des programmes, envoyer des messages, etc [HORR 86], [ROSE 88].

Andrew est constitué de deux parties principales :

- le système de fichier *AFS (Andrew File System)* est un système de fichiers distribué, conçu spécialement pour fonctionner sur une grande échelle. Il fournit le même système de fichiers de base pour toute machine ayant AFS et peut inclure n'importe quelle machine sur l'Internet. Il se situe au dessus de l'arborescence du système de fichier hôte et permet des manipulations plus faciles de fichiers ou de messages.

- la partie *Toolkit (ATK : Andrew Toolkit)* est un environnement extensible pour la création et l'utilisation d'applications ou de documents multimédias. Elle permet le développement d'interfaces en fournissant une bibliothèque de routines de gestion de documents multimédias indépendantes du système de fenêtrage. L'architecture d'ATK supporte l'imbrication d'objets les uns dans les autres sur l'écran, ou sur la page imprimée. Parmi ceux-ci, on trouve des objets texte multi-polices éditables, des objets de tableurs, des objets raster, de nombreux objets de type widget. Plusieurs applications sont fournies à l'utilisateur tels qu'un éditeur multimédia, un tableur, etc.

L'application la plus utilisée est le système de messages *AMS (Andrew Message System)*. C'est un système de gestion de messages et de carnet de rendez-vous qui permet l'échange de documents Andrew à travers la messagerie et offre différentes possibilités d'organisation des messages. AMS supporte aussi les communications de textes seuls avec le reste du monde de la messagerie électronique [BORE 91b].

Les messages Andrew contiennent tous les types de données manipulés par le système : textes multi-polices, rasters, sons, widgets. La création des messages est réalisée par l'éditeur multimédia *ez*. Selon le type de média, celui-ci appelle automatiquement l'outil correspondant et permet d'inclure les parties créées dans le corps du message. Le format des messages est spécifique à Andrew et correspond au format généré par l'éditeur multimédia. L'utilisateur a la possibilité d'inclure des programmes dans son message en utilisant le langage *Ness*. Ce langage de programmation est basé sur ATK et peut être utilisé pour créer des programmes qui sont inclus dans des applications ou dans des documents. Un autre langage, *Flames*, permet de trier et classer automatiquement les messages lors de leur réception.

Au niveau architecture, deux composants VICE et VIRTUE forment l'environnement du système. VIRTUE est la partie utilisateur qui inclut le gestionnaire de fenêtres et la bibliothèque de base, ainsi que les programmes d'application (ATK). VICE est le système de fichiers émulant le système de fichiers Unix (AFS). Les communications entre VICE et VIRTUE sont fondées sur un modèle client-serveur [BORE 88].

2.2. Diamond/Slate

Diamond est un produit de BBN Laboratories dont l'objectif est d'enrichir les communications électroniques avec d'autres médias. C'est un système distribué pour la création, l'édition, la transmission et l'archivage de documents multimédias [CROW 85]. Il est implanté sur une architecture distribuée : des stations de points d'accès utilisateur et un ensemble d'ordinateurs partagés qui fournissent des services comme le stockage et l'impression des documents.

Un document Diamond peut contenir du texte, des graphiques, des images et de la voix ainsi que d'autres objets dont la présentation est exécutée par des programmes d'applications. Ces documents ont différentes utilisations : messages, mémos, notes. Un document multimédia Diamond est un objet structuré ; il contient une collection d'éléments de différents types. On associe à chacun des informations sur sa présentation et ses relations avec les autres éléments. La structure logique du document est modélisée sous forme de hiérarchie.

L'utilisateur compose ses documents au moyen d'un éditeur multimédia qui est constitué en fait de plusieurs éditeurs complexes réalisant des fonctions spécifiques à un média particulier. Cet éditeur fournit la possibilité de combiner les entrées à partir de plusieurs périphériques d'entrée/sortie : scanner, clavier, souris. Il est fortement intégré ; tous les objets sont affichés et édités sur une même surface d'affichage et dans un seul processus. L'accès de l'utilisateur aux fonctionnalités de Diamond est réalisé à travers une interface graphique.

Slate est un produit de BBN dans la continuation de Diamond [CROW 90]. De même principe que Diamond, Slate permet à travers une interface graphique, la création et la manipulation de documents multimédias pour des besoins de messagerie ou de téléconférence. Il dispose de nouvelles possibilités par rapport à Diamond, comme l'inclusion d'images en couleur, l'édition en plusieurs langues. Un langage de programmation (SEL : Slate Extension Langage) intégré dans l'éditeur, permet à un utilisateur de configurer à un haut-niveau l'interface.

Slate incorpore 5 éditeurs dans la même interface : éditeur de texte, tableur, outils de création de graphiques géométriques, de visualisation d'images et de production de voix. Chacun possède des fonctions génériques comme la création de nouveaux objets ou leur impression ainsi que des fonctions spécifiques à un type de média. Ils sont reliés à une structure principale permettant de les appeler selon le type d'objet sélectionné.

2.3. MMS (Multimedia Mail System)

Ce système a été développé par l'Université de Californie du Sud, l'ISI et DARPA [REYN 85], [POST 86]. Il comprend deux parties :

- le module de traitement de message MPM (Message Processing Module) qui réalise les fonctions de transfert du message dans l'environnement de communication.

- le programme d'interface utilisateur UIP (User Interface Program) qui permet la composition et la lecture des messages. Ceux-ci peuvent inclure du texte, des dessins et des images, de la voix digitalisée ou des feuilles de tableurs. Ils sont créés par des outils externes et représentés dans une structure hiérarchique. Les deux modules communiquent au travers de fichiers partagés.

L'utilisateur crée ses messages en utilisant les commandes du module UIP. Il prépare les différents champs et peut les modifier grâce à un éditeur. Le MPM accepte le message, détermine la destination et le transmet.

Nous reviendrons par la suite à ce système au niveau du contrôle des séquences des messages.

2.4. Systèmes commerciaux

Les produits commerciaux de messagerie multimédia sont de plus en plus nombreux sur le marché. Certains ne fournissent que la possibilité d'ajouter des sons à un message, d'autres sont plus élaborés et permettent d'obtenir des messages comportant plusieurs types de médias. On peut citer parmi les plus connus, *Mailtool* sous Openwindows 3.0 ou *cc:Mail* de Microsoft.

2.4.1. Mailtool

Le produit Multimedia Mail Tool est une interface graphique du programme mail d'Unix, disponible sous l'environnement OpenWindows de Sun. Il offre le moyen de composer, envoyer et recevoir des messages multimédias par l'inclusion de liens aux messages. Ce principe permet d'envoyer des fichiers de type quelconque tout en préservant le contenu du message original. Ces liens peuvent être des fichiers audio, des images ou des documents ou n'importe quel fichier existant. Ils sont codés sous forme texte avec une en-tête indiquant leur type et leur forme de codage et sont transmis avec le corps du message. Chaque fichier rattaché est représenté sous forme iconique dans une fenêtre adjacente. Le fait d'ouvrir l'icone appelle le programme de restitution correspondant au type de média (éditeur de texte, d'images). Cependant, l'outil ne reconnaît que les types de formats d'images ou de sons pour lesquels il a été conçu à savoir respectivement le format Sun raster et les codages 8bits/u-law ou G721.

2.4.2. cc:Mail

cc:Mail est un outil de messagerie développé pour PC et Macintosh. Cet outil permet de composer des messages comprenant du texte et des images N/B ou couleurs selon un format propre à l'outil. Il est possible de référencer des fichiers ou des documents existants de type texte, graphiques ou binaires. Un outil supplémentaire (VoxVoice ou VoxMail) est nécessaire pour incorporer dans les messages des fichiers sons (par référence). L'interface Mac est simple et conviviable avec un éditeur de texte et un éditeur graphique. Les fichiers référencés sont représentés sous forme d'icônes dans une fenêtre fille à partir de laquelle on peut visualiser leur contenu. L'interface disponible sur DOS est un peu plus complexe d'utilisation. Un éditeur graphique intégré permet de dessiner des images et un autre utilitaire permet de capturer des copies d'écrans.

2.5. MetaMail

MetaMail (MultiMedia for the Masses) est un produit développé par N. Borenstein à Bellcore. Il est disponible sur Internet depuis début 1992. Ce n'est pas vraiment un système de messagerie mais plutôt un outil qui vise à résoudre le problème de compatibilité entre des systèmes hétérogènes. Il permet de convertir n'importe quel programme d'agent utilisateur de messagerie sur Unix en un programme multimédia.

C'est une implantation générique de *MIME* [RFC 1341], le standard pour les formats de messages multimédias sur Internet.

L'implantation est souple et extensible car elle utilise un mécanisme de fichier de configuration permettant de spécifier des supports pour de nouveaux formats de données envoyés par messagerie. Sur un site hétérogène où plusieurs agents utilisateurs de messagerie sont disponibles, ce mécanisme permet d'étendre leur possibilités en supportant des messages multimédias par simple adjonction d'un fichier de configuration. Ce fichier spécifie pour un agent utilisateur de messagerie les outils externes qui doivent se charger des différents types de médias. A partir de ce mécanisme de base, le système fournit des outils pour un certain nombre de types de messages définis par *MIME* : textes ASCII ou formattés, formats d'images, séquences audio et vidéo, messages multi-parties, messages encapsulés, données binaires. D'autres types de médias peuvent facilement être ajoutés. L'outil MetaMail fournit aussi un support rudimentaire pour l'emploi de caractères non ASCII dans certaines en-têtes de messages.

Ce produit a donné lieu à une spécification qui suggère un format de fichier pour informer de multiples programmes d'agents utilisateurs sur les outils installés localement pour manipuler des messages dans différents formats [RFC 1343]. Ceci permet à de nombreux agents de messagerie d'obtenir la capacité de restituer une grande variété de messages. Le mécanisme est explicitement conçu pour fonctionner avec des systèmes basé sur Internet, toutefois, il est probable qu'en ajoutant certaines extensions, il peut fonctionner avec X400.

3. PERSPECTIVES

3.1. Approches en cours

Au cours des dernières années, divers produits de recherche et commerciaux ont élargi le domaine de la messagerie électronique par inclusion d'autres types de médias et ont démontré le potentiel énorme de la messagerie multimédia. Des systèmes comme *Andrew* et *Slate* mettent en évidence la valeur de formats multimédias très riches tandis que des sociétés commerciales tels que *NeXT* et *Microsoft* insistent sur l'intérêt d'envoyer des images et de la voix via la messagerie électronique. Mais, faute de standards multimédias bien acceptés, ces systèmes sont incompatibles entre eux. Par exemple un utilisateur de Slate ne peut lire un message multimédia créé par un utilisateur de Mailtool. Il est possible d'échanger des messages entre Andrew et Diamond en utilisant ODA comme langage intermédiaire, mais l'échange n'est ni automatique ni transparent à l'utilisateur et entraîne une perte d'informations. Or le but d'une messagerie est d'échanger des informations entre de nombreux utilisateurs dans un environnement matériel et logiciel hétérogène. La compatibilité des systèmes est donc un problème important et difficile à résoudre. Ces systèmes impliquent l'abandon par les usagers de leur système de messagerie favori en faveur de nouveaux outils. Les utilisateurs sont d'autant moins enclins à s'investir dans un produit que celui-ci ne peut leur assurer une transmission et une réception correctes de leur message. Enfin, ces systèmes sont en général des logiciels complexes et lourds à implanter.

Une approche différente de la messagerie multimédia consiste, à partir des systèmes de messagerie existants, à incorporer des caractéristiques multimédias. En l'absence de standards, une telle approche dite "bottom-up" permet de fournir des possibilités de messagerie multimédia aux usagers sans modifier leur environnement habituel. C'est l'approche choisie par MetaMail. L'objectif est une évolution progressive plutôt qu'un changement radical sans garantie d'uniformité. Elle permet ainsi de tester des nouveaux concepts sans attendre une uniformisation des standards. De plus, c'est en fournissant aux utilisateurs le moyen de tester les apports du multimédia dans la messagerie que l'on hâtera le processus de développement.

Néanmoins, si tous ces systèmes ont le mérite de fournir des possibilités multimédias aux utilisateurs de messagerie, la plupart ne prennent pas en compte l'aspect dynamique du message multimédia qui, par définition, est un objet évoluant dans l'espace et le temps et dialoguant avec son environnement Ces différents points sont explicités dans le paragraphe suivant.

3.2. Besoins

Les besoins de la messagerie multimédia sont réels et importants. Ce sont non seulement ceux liés à l'environnement multimédia que nous avons précédemment évoqués mais aussi ceux liés aux composantes propres des messages manipulés.

Dans les systèmes existants tels qu'Andrew ou Slate, les fonctions de création sont réalisées à partir d'un ou plusieurs éditeurs intégrés au travers d'une interface graphique. Des possibilités de présentation du message sont fournies à l'utilisateur pour lui permettre de concevoir le message désiré. Ces possibilités prennent en compte uniquement la notion d'espace, c'est-à-dire la disposition des différentes parties à l'intérieur du message. L'utilisateur peut ainsi disposer un graphique à côté d'un paragraphe de texte. L'aspect temporel du message, c'est à dire son déroulement dans le temps n'est pas pris en compte ; il est cependant essentiel dès que l'on intègre des données multimédias qui sont dépendantes du temps comme la vidéo.

Un autre aspect important à souligner est que dans la plupart de ces systèmes, le message est considéré comme un objet passif. Le destinataire ne peut que visualiser le message et prendre connaissance des informations contenues. Or, il peut s'avérer utile d'envoyer des messages actifs qui exécutent des programmes ou auto-délivrent des réponses. Ces message actifs sont des types spécialisés de messages qui transportent avec eux, en plus de leur contenu (normal ou multimédia), des informations qui dirigent une interaction particulière avec l'utilisateur. Un exemple de tel message est le message "vote" dans le système de messagerie Andrew [BORE 91b]. Dans ce type de message, des en-têtes spécifiques dirigent l'interface pour poser à l'utilisateur des questions à choix multiples. Le message "vote" se présente comme un message qui ouvre des boîtes de dialogue et posent des questions. L'utilisateur sélectionne des réponses parmi les choix possibles qui sont automatiquement dirigées vers les adresses désignées pour la collecte et le dépouillement des votes. On peut citer d'autres exemples de messages actifs comme les inscriptions automatiques à des listes de distribution ou les redistributions de renseignements sur un sujet. A l'heure actuelle, ce genre de demandes nécessite souvent

plusieurs messages successifs entrecoupés de recherches et de tris parmi les informations fournies. Les interactions possibles avec l'utilisateur sont donc un aspect primordial d'un système de messagerie. Elles permettent de guider l'usager pour obtenir rapidement les informations souhaitées. Elles lui fournissent aussi le moyen d'utiliser pleinement les caractéristiques multimédias des messages, comme par exemple, sélectionner une image parmi une vidéo ou rejouer une séquence sonore.

Enfin, un dernier aspect du message multimédia qui est généralement oublié dans l'ensemble de ces systèmes, est la prise en compte de l'évolution du message et des services qui accompagnent son transfert. En effet, un grand nombre de messages sont destinés à circuler entre des utilisateurs et des applications. Par exemple, un message d'informations sur un projet de construction est échangé entre les différents partenaires : promoteurs, architectes, entrepreneurs. Il est modifié, complété selon les aptitudes de chacun au fur et à mesure de l'avancement du projet. Ce type de message véhicule ainsi des informations dépendantes de l'état en cours du message : mises à jour, vérifications, etc, qui reflètent l'historique et le devenir du message. Cette notion d'historique est importante car elle permet de suivre la vie du message et l'ordre des ajouts et modifications apportés.

3.3. Objectifs

L'analyse de l'état actuel de l'environnement multimédia, des différents systèmes existants, et l'ensemble des besoins que nous venons de citer nous a amené à définir les contraintes que doit remplir un système de messagerie multimédia :

- Il doit permettre de spécifier à la fois les contraintes spatiales et les contraintes temporelles des messages. Ceci revient à expliciter comment les différentes parties du message s'organisent les unes par rapport aux autres dans l'espace du message et dans quel ordre elles doivent s'agencer à la réception du message.

- Un message multimédia est un message actif qui dialogue avec l'utilisateur. Il peut fournir des réponses, agir sur l'environnement de l'utilisateur ou exécuter des programmes. Il faut donc prendre en compte, non seulement la présentation du message mais aussi les actions possibles engendrées.

- Le message a une certaine vie qui lui est propre. Il faut donc pouvoir représenter, si nécessaire, ce qui doit être fait de chaque message, où doit-il être transmis, quelles sont les informations à apporter, etc.

- Les messages doivent être restitués, autant que possible, dans leur forme originelle quelle que soit la machine réceptrice. Pour cela, le système doit être capable de reconnaître et d'interpréter les différents types de médias, offrir des moyens de substitution pour la représentation de certaines données, et fournir des moyens de conversion entre formats dans le cas où l'expéditeur et le destinataire n'ont pas de systèmes compatibles.

- Si l'on veut que ce système de messagerie soit largement employé, il doit être simple à utiliser, souple et extensible.

Notre objectif est de considérer l'ensemble de ces contraintes propres à la messagerie multimédia ; diverses solutions sont possibles pour les réaliser. Nous avons vu dans le chapitre précédent qu'il n'existe pas de standard multimédia et la définition d'une structure adéquate est une tâche de longue haleine, qui semble prématurée à l'heure actuelle. Tenter de concevoir une représentation standard d'un ensemble d'éléments (textes, sons, images) qui eux-mêmes ne sont pas normalisés est un travail hasardeux. Notre approche ne sera donc pas de concevoir un "standard" multimédia mais plutôt de proposer un modèle intelligent, adapté aux objectifs fixés. Un tel modèle doit être suffisamment souple pour s'adapter aux contraintes liées à la communication d'informations complexes et multimédias dans un environnement hétérogène et être suffisamment puissant pour prendre en compte l'ensemble des impératifs liés aux messages multimédias.

Le modèle proposé se situe dans un contexte plus général que la messagerie multimédia. Il est basé sur un modèle de communication par agents actifs circulant sur les réseaux. Nous allons, dans la partie suivante, en exposer les caractéristiques, puis nous montrerons son adéquation à la messagerie multimédia.

LE MODÈLE PROPOSÉ II

3. Modèle de communication
4. Messages multimédias
5. Représentation

CHAPITRE 3

Modèle de communication

L'échange de données dans son sens le plus large sera demain une nécessité absolue. Les divers problèmes et besoins évoqués précédemment dans le contexte de la messagerie multimédia, nous ont amené à définir un modèle de communication. Celui-ci a été conçu dans un contexte plus général que la transmission de données multimédias, pour répondre aux besoins très variés d'applications quelconques amenées à communiquer entre elles. Il est basé sur la circulation d'agents de communication. Nous présenterons les besoins engendrés par les problèmes de communication entre applications en nous appuyant sur des exemples. Puis nous détaillerons les caractéristiques et les originalités du modèle proposé pour répondre à ces besoins.

1. PRÉSENTATION

1.1. Contexte

La messagerie multimédia est une application typique des problèmes de communication entre applications. On se heurte à l'impossibilité de communiquer des informations multimédias entre des usagers du fait de l'hétérogénéité de leurs environnements. Le problème n'est pas spécifique à la messagerie multimédia mais à tout type d'applications désirant échanger des informations dans des environnements hétérogènes. En effet, la plupart des applications ont été construites pour communiquer par des protocoles figés, explicitement prévus dans leur code. Il s'ensuit que la plupart de ces applications ne peuvent communiquer avec le monde extérieur ou entre elles, en utilisant d'autres protocoles. Une solution pour l'avenir (et qui ne résoud certes pas le cas des applications déjà existantes) est l'adoption de normes qui permettent le dialogue entre les différents partenaires et l'échange de données complexes. Mais si une norme est certainement nécessaire, elle n'est pas forcément suffisante. En effet, on peut distinguer deux types d'informations véhiculées : les informations communes à tous les types

d'applications et propres à l'échange proprement dit, c'est à dire les informations de présentation et d'administration, et les informations qui véhiculent la sémantique spécifique à l'application. Si les premières doivent être prises en compte par une norme, les secondes sont difficiles à prévoir et ne peuvent être représentées de façon universelle. En partant des problèmes de représentation et d'échange de données multimédias entre utilisateurs, nous avons été amené à étudier un modèle de communication nécessaire au transfert d informations entre applications quelconques et les solutions que l'on peut envisager.

1.2. Exemples

Les deux exemples qui suivent, concrets et réalistes, ont pour but de montrer les divers problèmes engendrés par des besoins de communications entre applications non prévues pour cela.

1.2.1. Projet d'architecture

Cet exemple est basé sur la mise en place d'un projet architectural illustrant un problème de travail coopératif et de circulation de documents. Au départ, l'architecte conçoit les plans qu'il envoie à plusieurs bureaux spécialisés (d'ingénierie électrique, de climatisation, d'études en béton). Ceux-ci vont compléter et valider les documents selon leur compétence. L'architecte demande ensuite l'élaboration d'un devis quantitatif par un organisme adéquat («métreur»). Ces divers devis quantitatifs sont envoyés aux entreprises correspondantes (plomberie, électricité, plâtrerie) qui proposent leur devis. Il peut y avoir à ce niveau une validation du projet par un organisme de contrôle pour répondre à des besoins de sécurité. L'architecte choisit ses divers partenaires pour la réalisation du projet. Ceux-ci sont ensuite amenés à échanger des informations tout au long du suivi du chantier. Une fois le projet achevé, l'architecte fournit le plan des ouvrages exécutés. La réalisation peut donner lieu ensuite à des choix de promotion par un cabinet publicitaire.

Dans cet exemple complexe de comunication, de nombreuses données entrent en jeu : le nombre de partenaires, le matériel disponible selon les organismes, le volume et le type des données échangées, la circulation et le partage des informations. Celles-ci sont elles-mêmes fortement structurées et de types divers (textuelles, numériques, graphiques, etc). On se retrouve des problèmes de saisie et d'échanges d'informations complexes, de conversion entre logiciels pour traitement et calculs, d'aller-retour entre organismes, de modifications de prévisions et de mise en place de réunions.

1.2.2. Demande d'analyse

Prenons un second exemple ; il correspond à une demande d'analyse par un centre hospitalier. Un malade se présente à l'hopital et est reçu dans le service des urgences. Après examen, un membre qualifié du personnel prescrit une analyse. Entre-temps, le malade est transféré dans un autre service. Il faut donc faire suivre les résultats des analyses au service où se trouve le malade. A partir de ce scénario très simple, on rencontre divers problèmes. Il faut d'abord identifier le malade pour pouvoir lui associer ses analyses (au fur et à mesure que le malade se déplace, il faut faire suivre son

identificateur) ensuite choisir un laboratoire d'analyse selon des critères de proximité, de nombre d'analyses en cours ou de matériels spécifiques. Une fois les résultats disponibles, ils sont alors transmis au service demandeur, les urgences. Il est possible que les informations soient attendues sous des formes différentes sur le site demandeur et sur le site d'analyse (forme non textuelle par exemple) et ne peuvent être retraitées directement. Ceci nécessite de nouvelles saisies. L'analyse doit ensuite être transférée au service d'accueil du malade qui n'est pas forcément connu du service des urgences. Enfin, l'imputation du coût des analyses à un service et un malade spécifiques vient greffer un nouveau schéma de circulation d'informations.

Dans ce cas simple, on retrouve des problèmes de communication entre applications : lier les analyses avec le malade, récupérer les résultats, être capable de les transmettre sur le nouveau site, faire suivre des informations...

1.3. Besoins

Les exemples évoqués ci-dessus illustrent deux types de besoins engendrés par la communication d'informations entre applications :

- la communication entre des applications n'ayant pas été forcément prévues pour communiquer avec l'extérieur ou entre elles. Elles ne peuvent pas communiquer en utilisant d'autres protocoles que le leur. L'hétérogénéité des matériels et des d'informations nécessite des supports adaptés pour la génération, la manipulation et la récupération d'informations sous différentes formes de présentation (éventuellement des conversions). Cela suppose aussi de pouvoir représenter l'évolution des informations et la trace des modifications apportées pour garder un historique ainsi que le formalisme des services accompagnant ces données.

- la circulation des informations selon des schémas élaborés de communication. Elle peut obéir à des règles complexes : par exemple, elle peut nécessiter une diffusion vers plusieurs sites ou un enchaînement successif de sites. Elle peut conduire à des retours vers des sites ou obéir à des contraintes de synchronisation.

A l'heure actuelle aucune norme ne peut répondre à tous les besoins. Dans le domaine des modèles d'informations, ODA est le seul standard dont les concepts prennent en compte la description de différents types d'informations dans plusieurs étapes de traitement. Comme ODA a été développé pour des documents traditionnels, plusieurs besoins importants pour un modèle général d'informations n'ont pu encore être satisfaits et n'ont pas été considérés de manière adéquate dans les extensions [SCHU 90]. De même, la norme EDI, si elle permet de formaliser le contenus de certains documents ne permet pas la formalisation de leur circulation [KREU 91]. Il est donc nécessaire de fournir un système ouvert pouvant exprimer des besoins de communication variés : EDI , multimédia, CSCW (Computer-Supported Cooperative Work), etc, permettant de créer des outils spécialisés de communication, pour chaque problème. Ce système doit aussi autoriser des conversions de données et plus généralement des transformations complexes pouvant aller jusqu'à la mise en oeuvre de techniques de l'intelligence artificielle. Ainsi, bien que la communication entre deux applications soit difficile, un tel modèle de communication doit permettre de construire assez rapidement, et à un coût faible, un outil

adapté aux divers cas. Par ailleurs, il doit mettre l'accent sur les services qui accompagnent les étapes d'un transfert de documents. L'aspect dynamique de l'entreprise doit être pris en compte : communications avec des partenaires nouveaux, évolution des logiciels, des matériels, des besoins, etc.

Le but du modèle proposé est d'exprimer de façon simple et souple, des communications complexes et spécifiques dans des environnements hétérogènes éventuellement distribués. Le système s'appuie sur des agents actifs circulant sur le réseau.

2. MODELE DE MESSAGERS

Nous décrivons ici un modèle de communication répondant aux besoins cités ci-dessus. Les protocoles normalisés impliquent presque toujours de la part des applications qui les utilisent des scénarios figés et pauvres. Sans remettre systématiquement en cause ces protocoles, nous essayons de permettre l'expression de scénarios beaucoup plus riches, personalisés et adaptatifs. Le modèle proposé n'utilise pas uniquement des protocoles figés correspondant à des normes plus ou moins universelles, mais emploie des *agents actifs* qui permettent des communications souples et adaptatives et représentent une approche beaucoup plus ouverte vers la plupart des applications actuelles.

2.1. Composantes

Le modèle de communication fait intervenir :

- des partenaires : émetteur, destinataires, intermédiaires (individus ou programmes),
- un scénario de communication : liaison virtuelle, diffusion générale, circulation restreinte, etc,
- des données à échanger, accompagnées éventuellement d'une description de ces données (type ASN1) et même d'une méta-description définissant la syntaxe et la sémantique de cette description.

Il apparaît comme une couche "services" conçue au-dessus d'une couche "réseau". La figure (cf FIGURE 6) décrit sommairement les différents éléments de cette architecture. Sur chaque site, un *processus résident* assure l'*exécution* des agents, la *communication* de ces agents avec le marché extérieur (applications, fichiers d'une part, couche réseau de l'autre), fournit des *services* à des applications locales, répond aux commandes d'un opérateur local ou distant, et peut éventuellement réagir par des *créations* de processus. Des extensions permettent de fournir localement des services supplémentaires (traduction, accès à des matériels spécifiques, etc).

2.1.1. Couche réseau

Cette couche s'appuie le plus possible sur des outils de transport existants (messagerie : X400, internet, transfert de fichiers : ftp, ftam ou protocoles de niveau inférieur). Nous n'avons pas l'intention de créer de nouveaux outils de ce type qui n'apporteraient qu'une rigidité superflue. Nous utilisons l'outil de communication local le

plus adapté. Par exemple, lorsqu'une communication n'a besoin d'aucun élément de synchronisation ni de séquencement, alors la messagerie électronique peut être un moyen adapté. Si la garantie d'arrivée est primordiale, il est nécessaire d'utiliser des outils générant des renvois tant que l'accusé de réception n'est pas arrivé (par exemple, TCP).

2.1.2 Couche services

Cette couche permet la création d'outils ad hoc conformes aux besoins. Un aspect «service» de secrétariat permet de suivre l'évolution des agents. Il est capable de répondre aux requêtes concernant le passage et l'état d'un agent ; par exemple, "est en cours d'exécution", "est arrivé à telle heure", "est déjà sorti". C'est là une des originalités du modèle.

Les services sont simples :

- exécuter un script avec des paramètres,
- envoyer un message à un agent existant sur le système,
- utiliser un port d'entrée associé à un agent suivant un protocole convenu avec l'agent.

Ces services peuvent être requis par le gestionnaire d'agents local de diverses façons :

- une commande envoyée dans le langage de commande, soit par messagerie, soit sur un port d'entrée permanent associé à l'agent.
- une commande dans le langage d'extension, transmise par les mêmes moyens que ci-dessus
- l'utilisation d'un port de communication, géré par un agent.

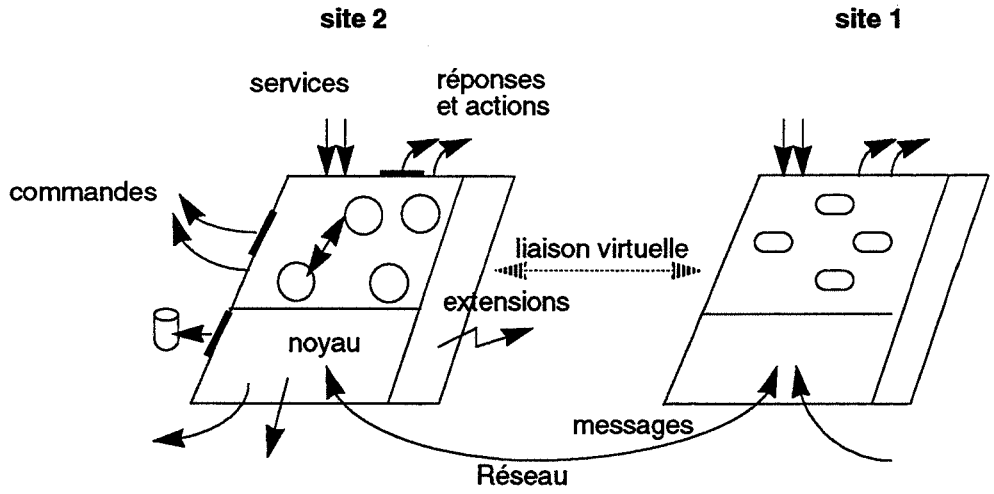


FIGURE 6 : *composantes du modèle de communication*

On constate que la couche réseau repose sur des outils existants. Aussi, dans notre modèle nous interessons-nous essentiellement à la couche services. Les besoins

d'autonomie des agents nous a amenés à étudier les modèles à acteurs. Le modèle proposé repose sur la communication et le transfert de messagers [AHRO 93a], [AHRO 93b]. Il doit permettre de formaliser des communications à caractère général entre des applications distantes et non homogènes. Il est basé sur un langage à acteurs dont les spécifications sont données en annexe (cf ANNEXE A). Nous présentons ci-après les concepts généraux des systèmes à acteurs.

2.2. Systèmes à acteurs

2.2.1. Notions générales

Le modèle à acteurs a été développé au départ pour résoudre des problèmes d'intelligence artificielle et de programmation concurrente [HEWI 77]. L'idée était de répartir les connaissances entre différents experts détenant chacun une connaissance très spécifique et coopérant entre eux plutôt que d'utiliser un système avec un seul expert formalisé par un programme raisonnant sur une base de connaissance centralisée. Le modèle s'est ensuite étendu aux systèmes distribués en particulier pour gérer des activités concurrentes. Des langages comme Act1, ABCL/1 ou Hybrid ont été développés pour gérer ces types de systèmes ouverts [YONE 86], [YONE 90], [NIER 87], [HERN 83]. Le but de ces nouveaux langages de programmation, dits langages à acteurs, est de simplifier l'expression de comportements complexes en décomposant ceux-ci en activités plus simples, réalisées par des entités autonomes comuuniquant entre elles [WYAT 92].

Le modèle acteur est basé sur le principe de l'encapsulation comme celui du modèle à objets classique mais introduit la notion d'activité autonome. Les objets, appelé acteurs, sont des agents actifs qui communiquent avec d'autres acteurs pour se confier des tâches ou échanger des informations. Un objet actif peut ainsi calculer, prendre des décisions, envoyer des messages, créer de nouveaux objets et mettre à jour sa mémoire locale. Les acteurs participent à des activités qui se créent, s'exécutent et éventuellement disparaissent. Ils évoluent donc dans un environnement dynamique. Dans un tel modèle, les connaissances et le comportement du système sont répartis parmi les différents acteurs. C'est un modèle distribué. Chacun effectue ses tâches de manière indépendante et plusieurs acteurs peuvent agir concurremment.

L'entité de base du modèle est un acteur, qui regroupe en son sein des connaissances et le moyen de les exploiter. Les acteurs peuvent être créés dynamiquement et chaque acteur a sa propre sémantique. De manière générale, il est défini par:

- des données locales, appelées *accointances* qui représentent les autres acteurs que l'acteur connaît directement (dont les proxy sur le rôle desquels nous reviendrons),

- un *comportement* qui définit les actions que l'acteur entreprend au cours de son existence, en fonction des événements qui surviennent. C'est ce comportement qui caractérise un acteur. Il est défini par un script unique et propre à chaque acteur. De manière générale, il regroupe un ensemble de procédures qui spécifient quels messages l'acteur accepte et quelles actions il doit réaliser à leur réception. Chaque procédure est ainsi spécialisée pour prendre en compte un sous ensemble de messages. Ce comportement associé à un acteur peut changer en réponse à des communications.

Les communications entre acteurs sont point à point (objet à objet) et asynchrones : un objet peut envoyer un message quand il veut, sans se préocupper de l'état ou du mode de l'objet cible. Le seul impératif est de connaitre son objet cible. Les messages sont stockés dans une boîte aux lettres associée à l'acteur. Au cours de son existence, un acteur reçoit donc une suite de messages qu'il traite en fonction de son état courant, en envoyant des messages à ses accointances, à lui-même ou à des acteurs créés spécifiquement pour traiter ce message. Un système à acteurs évolue donc par une suite d'effets résultants occasionnés par des événements survenant dans le système qui correspondent à la prise en compte des messages. Lorsque l'acteur a accès à une nouvelle information, il la propage aux acteurs qu'il connaît (ses accointances) qui à leur tour peuvent la propager et éventuellement en déduire de nouvelles informations. La connaissance est ainsi diffusée parmi l'ensemble des acteurs [MASI 90].

L'unique moyen de communication entre les acteurs est l'envoi de messages. Le mécanisme de transmission consiste en l'envoi d'un acteur (appelé le messager de la transmission) vers un autre acteur (la cible). Dans les modèles traditionnels, tout est acteur ; le message lui-même est un acteur transmis par un acteur *messager*. Cette caractéristique pose évidemment le problème de la récursion à l'infini. En effet, si l'on applique ce principe aux messagers, ils doivent eux-mêmes envoyer un message pour avoir accès au message qu'ils transmettent et pouvoir le communiquer et ainsi de suite. Pour éviter ce problème, on définit des acteurs dits *primitifs* ou "rock-bottom" ou "built in actors" qui n'ont pas besoin d'envoyer de messages ni de déléguer pour jouer leur rôle (par exemple, les opérations prédéfinies) ; L'acteur envoie donc des messages à ses accointances et ainsi de suite jusqu'à ce que les communications soient reçues par ces acteurs primitifs qui assurent ainsi les communications entre acteurs de plus haut niveau [SENT 90].

2.2.2. Modèle de Gul Agha

Le principe des acteurs a donné lieu à divers modèles à acteurs dont un des plus connus est celui de Gul Agha [AGHA 86], [AGHA 87]. Dans ce modèle, un acteur est décrit par :

- l'adresse d'une boîte aux lettres, à laquelle est associée une file qui contient les messages reçus par l'acteur. Ils sont rangés les uns après les autres et pris en compte selon leur ordre d'arrivée,

- un comportement qui représente l'ensemble des actions nécessaires pour traiter le message courant.

Les acteurs effectuent leurs communications sous forme de tâches déposées dans un système de messagerie. Celui-ci se doit d'assurer que la tâche sera délivrée à l'acteur approprié. Chaque tâche est consituée d'une étiquette pour son identification, d'une cible qui est l'adresse de la boîte aux lettres de l'acteur à qui l'on doit délivrer la communication et de la communication formée d'un nom et d'éventuels paramètres. Quand une tâche arrive à destination, elle est insérée dans la boîte aux lettres de l'acteur associé.

Chaque acteur peut avoir une ou plusieurs accointances qui sont les noms des boîtes aux lettres d'autres acteurs. Les accointances d'un acteur peuvent être fournies à la création de l'acteur et de nouvelles accointances peuvent être envoyées dans des communications. Ces accointances représentent l'état local d'un acteur. Chaque acteur a aussi un comportement qui décrit comment il doit répondre aux communications. L'acteur traite le premier message de sa boîte aux lettres selon son comportement courant. Le script accepte le message s'il le reconnaît sinon il le rejette. Pour traiter le message il peut envoyer des messages à ses accointances ou à lui-même (en créant une copie) ou à un acteur créé spécifiquement pour se charger du message. Le script spécifie aussi un comportement de remplacement qui est en fait un nouvel acteur (avec la même adresse de boîte aux lettres que le générateur) qui accepte le prochain message et agit sur celui-ci. La concurrence du système (c'est à dire la possibilité d'exécuter des tâches en parallèle) est ainsi réalisée quand un acteur spécifie un comportement de remplacement avant de répondre au message suivant reçu. La cohérence de l'ensemble est assurée par le fait que tout message envoyé sera reçu dans un temps fini. Enfin, un acteur peut déléguer les messages rejetés à un *proxy* (un autre acteur dont l'adresse de boîte aux lettres est connue de l'expéditeur). C'est le mécanisme de *délégation.* Le proxy contient habituellement des informations supplémentaires pour répondre au message et travaille pour le compte de l'acteur initial en modifiant l'état interne de ce dernier.

Toutes les tâches ayant été traitées ainsi que les acteurs devenus inutiles peuvent être détruits par le système.

2.3. Caractéristiques de notre modèle

Les modèles traditionnels comme celui de Gul Agha sont des modèles de calcul adaptés aux problèmes de concurrence massive [BAUD 91a], [BAUD 91b]. Un messager est une entité très simple qui transmet une communication de bas-niveau (opérations algébriques, valeurs d'entiers) ; c'est la seule structure de contrôle. Notre modèle est un modèle de communication ; les messages sont donc des acteurs de haut-niveau qui formalisent une activité de transfert d'informations. Nous envoyons entre deux acteurs, respectivement l'expéditeur et la cible, non pas une donnée brute mais une information pouvant être complexe, encapsulée dans un messager. Cette information va nécessiter une certaine intelligence de la part du messager. En effet, il peut être transféré de machine en machine, devoir s'authentifier (par exemple, en dialoguant avec une tierce partie qui est un service d'authentification) ou mettre en place des traitements d'informations (exemple : conversion) pour l'acteur récepteur. Toutes ces opérations que le messager doit savoir accomplir en font un acteur à part entière.

Dans les modèles à acteurs traditionnels, le messager sert uniquement à véhiculer des informations. Une fois arrivé à destination, il délivre son message et meurt. Au contraire, dans notre modèle, son activité commence seulement à ce moment : il peut obtenir le droit d'exécution, générer de nouveaux messagers ou modifier son comportement. On peut l'assimiler à une conjonction du destinataire et du messager. Ce n'est donc pas un acteur de base dont le fonctionnement est "magique" comme tous les *rock-bottom acteurs* mais un acteur comme les autres.

De plus, dans les modèles à acteurs classiques, l'environnement est très localisé, les acteurs étant plutôt utilisés pour résoudre des problèmes de parallélisme. Dans notre cas, cet environnement est largement distribué. Les messagers peuvent circuler d'un site[1] à l'autre, et acquièrent ainsi une autonomie de comportement dépendant de la machine sur laquelle ils s'exécutent. Un messager ne s'exécute que sur un seul site à la fois. Un messager est donc actif sur un seul site, mais peut être transmis d'un site à un autre. On distinguera ainsi le messager comme celui qui contient l'expression de la communication et qui transite entre les divers sites, de l'acteur qui correspond à l'environnement d'exécution d'un messager. L'arrivée d'un messager déclenche son exécution locale. Cette exécution peut conduire à l'envoi d'autres messagers, au dépôt d'informations, etc. Un messager peut ensuite rester en attente sur un site jusqu'à l'arrivée d'un message qui lui est destiné ou jusqu'à sa date de péremption. Il peut être éventuellement réactivé à ce moment.

Nos messagers empruntent des caractéristiques des acteurs du modèle de Gul Agha. Un messager est un objet actif pourvu d'un *comportement* défini par son script initial et d'un *environnement* qui lui sont propres. Chaque messager contient aussi un ensemble de données correspondant à son état courant ; ainsi tous les messagers sont dotés d'une date d'expiration qui permet de connaître leur durée de vie. L'historique de chaque messager permet de garder une trace de son évolution.

2.4. Communications entre acteurs

2.4.1. Primitives

La communication entre deux acteurs est réalisée par l'envoi de message. Un message contient en plus de la communication, une *promesse*. Celle-ci correspond à une référence à un résultat attendu et peut être tenue (immédiatement ou plus tard) ou non. Concrètement, c'est une *portion de mémoire partagée associée à un sémaphore*. Ce mécanisme permet de répondre aux besoins d'enchainement et de synchronisation d'informations. Les principes sont les suivants : un acteur ne reçoit qu'une communication à la fois ; il n'accepte la communication suivante que quand il annonce qu'il est prêt. S'il souhaite traiter une autre commuication, il délègue la tâche courante à un autre acteur plus spécialisé, d'où une organisation assez hiérarchique.

Pour réaliser ces communications, on dispose des primitives suivantes :

```
R = send (<acteur> , <message> [ , <promesse> ] )
```

Cette primitive envoie un message à l'acteur référencé avec éventuellement un promesse de résultat. Le résultat est une promesse qui sera la réponse de l'acteur. Elle correspond à celle donnée en paramètre et est fabriquée sinon. La promesse est délivrée immédiatement par le *scheduler*, et non par l'acteur, qui, éventuellement, ne recevra le message que beaucoup plus tard.

1. Un site correspond à un ensemble de machines connectées par un réseau local.

`ready ( <promesse> )` : Cette primitive permet de tester si la promesse est tenue ou non ; elle est non bloquante et rend vrai ou faux selon le cas.

`wait-some ( <p1> , <p2> , ... , <pn> )` : cette fonction attend jusqu'à ce que l'une des promesses passées en parmètre soit remplie.

`wait-every ( <p1> , <p2> , .... , <pn> )` : celle-ci attend le résultat de toutes les promesses.

`value ( <promesse> )` : attend que la promesse soit tenue et rend la valeur.

`set-value ( <promesse> , <valeur> )` : fournit une réponse à une promesse.

Ce principe de promesse permet à l'envoyeur du message de poursuivre son travail, même si l'acteur récepteur n'est pas immédiatement disponible. Grâce aux fonctions d'interrogation et d'attente, il est possible de réaliser les formes habituelles de communication des langages acteurs : synchrones, asynchrones et anticipées. Chaque acteur peut aussi se constituer un ensemble de promesses et se mettre en attente sur un sous ensemble. Ceci permet de synchroniser des acteurs. Réciproquement, une promesse peut être attendue par plusieurs acteurs.

Le système n'autorise pas le partage de promesses entre sites, mais le même effet peut être obtenu par une communication locale avec un acteur qui est le représentant de l'acteur distant. Une promesse distante peut être donc réalisée par une promesse locale à laquelle on ajoute un mécanisme (sous forme d'acteur) chargé de dialoguer avec l'extérieur.

2.4.2. Schéma de communication

Prenons le cas d'une demande de communication demandée par un client (un programme d'application P) sur un site S1 vers un client (un programme Q) sur un site S2 (cf FIGURE 7). P est prévu pour générer un fichier contenant un ensemble de pièces à commander. Le seul protocole de communication qu'il connaît est l'écriture sur fichier. Q est prévu pour satisfaire des commandes à partir de la lecture de celles-ci sur un fichier. Après connexion des deux sites sur un réseau, on essaie d'utiliser ce moyen de communication. Supposons que sur chaque site, se trouve implanté un système de messagers et voyons comment pourrait se dérouler cette communication.

Le client P exprime sa demande de communication sous forme d'une expression propre à son environnement local, par exemple sous forme d'écriture dans un fichier de la chaîne de caractères < Je, soussigné P, commande x pièces alpha à Q >. Cette expression n'est pas compréhensible sous cette forme par le client destinataire. Le gestionnaire local d'acteurs (GA) va créer un acteur représentant de ce programme P noté RP. Cet acteur va demander au GA local la création d'un messager pour traiter la demande de communication. Pour cela, le GA va traduire (par l'intermédiaire du traducteur) l'expression de communication dans le langage d'extension connu universellement par les GA, puis fournir au messager les éléments de comportement correspondant au contexte local du site récepteur. Par exemple, le code confidentiel de communication est la chaîne

"#@*!" (cryptée), la boîte aux lettres de Q s'appelle BLQ, ne commander que si le prix des articles est inférieur à 1000F. Le messager va, si besoin est, demander par envoi de messages à d'autres acteurs comment réaliser une tâche spécifique, par exemple, comment calculer une remise. Ce messager peut donc être considéré comme un acteur standard auquel on fournit tous les éléments nécessaires à son exécution sur le site destinataire. Il faut noter que P ne connait pas Q mais a seulement un point d'entrée sur le site destinataire par l'intermédiaire du GA existant.

Le GA va garder une trace du passage du messager et du délai qui lui a été imparti. Pour cela, il peut marquer le messager d'une estampille fournie par un acteur. Une fois muni des informations nécessaires, le message est transmis vers le site S2. On rappelle que l'on ne s'occupe pas du moyen de transfert des messagers entre sites.

A l'arrivée sur le site S2, le GA local va fournir au messager les données d'environnements nécessaires à son exécution. A ce niveau, le messager va s'exécuter selon la forme choisie par le gestionnaire ; il va éventuellement sous traiter une partie de sa mission à un autre messager (M') ou obtenir de l'aide de la part des acteurs résidents en communiquant avec eux (héritage de comportements, obtention de privilèges, etc). Dans notre cas, ne sachant pas comment obtenir les prix avant de commander, il demande quelle est la procédure locale et pourra ainsi obtenir le mécanisme de lecture des prix dans la base de données locale.

On peut noter que le messager qui circule est un messager "léger" c'est à dire composé de très peu de code personnel puisqu'il n'est pas nécessaire de doter le messager des éléments d'environnement connus de tous les sites (par exemple, le langage d'extension). Quand il doit s'exécuter, le GA local lui fournit l'environnement et les ressources nécessaires à cette exécution.

Cet exemple très simple montre que l'essentiel du problème réside dans la circulation des informations entre les divers partenaires. La formalisation de ces protocoles d'échanges doit être souple, et dynamiquement modifiable. L'approche par messagers est une solution envisageable à moindre coût pour prendre en compte l'ensemble des besoins même si elle nécessite au départ la spécification des scripts spécifiques (description des acteurs RP et RQ).

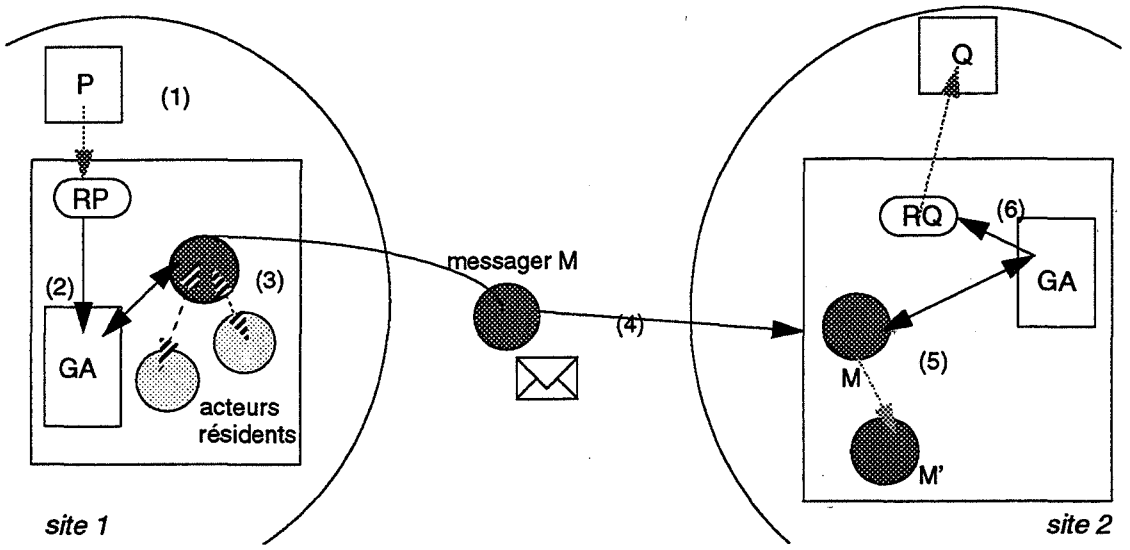


FIGURE 7 : *principe de communication par messagers*

3. SYSTEME DE MESSAGERS

3.1. Architecture

Le modèle que nous proposons, se compose de plusieurs entités formant un *système de messagers*. Il est présent sur chaque site voulant établir une communication. Chaque système est constitué de plusieurs parties : un *noyau* qui regroupe les fonctions et les opérations de base avec un ensemble d'acteurs résidents et des entités externes au noyau et comprises dans le système de messagers(cf FIGURE 8).

- Le noyau

Il regoupe toutes les fonctionnalités minimales d'un système de messagers. Il est constitué des éléments suivants :

- Un traducteur du langage d'extension utilisé par les acteurs pour formaliser leur comportement, auquel est associé un mécanisme d'ordonnancement (scheduler) qui génère les tâches demandées par les acteurs et contrôle leur déroulement, leur séquencement et leur synchronisation,

- Des acteurs résidents définis initialement mais qui peuvent être créés ou modifiés au fur et à mesure que le système évolue. Parmi ces acteurs, on peut trouver des acteurs susceptibles d'envoyer des événements à d'autres acteurs, de réveiller un acteur, de fournir les primitives locales de base d'entrées/sorties, ou encore les services «secrétariat».

Indépendamment de ce noyau de base, on trouve toutes les fonctions que celui-ci n'a pas besoin de connaître directement mais qui peuvent être demandées par les acteurs. Ces fonctions peuvent être implantées sous forme d'acteurs et être ainsi créées

dynamiquement par d'autres acteurs. Elles ne sont pas résidentes dans le noyau (comme les acteurs résidents) car elles ne sont pas considérées comme indispensables et minimales. Ce sont par exemple, des fonctions d'authentification, d'annuaire, de statistiques ou de comptabilité. Dans le cas de l'authentification, lorsqu'un messager se présente sur un site, il entrera librement si aucune fonction d'authentification n'existe et devra passer par la procédure de contrôle d'identité dans le cas contraire. On peut remarquer ici que la procédure sera plus ou moins rigide selon les sites.

- Le gestionnaire d'acteurs

Le *gestionnaire d'acteurs* est de manière générale, un superviseur de tous les acteurs existants dans le système. Il est chargé de l'implémentation des fonctions de création et de destruction des acteurs du système. Il permet de garder la trace du passage des messagers et délivre les messages aux destinataires. Il est aussi responsable de la "lyophilisation" des messagers (extraction des données universelles et connues de tous) pour leur transfert vers d'autres sites et dans le sens inverse de leur "gonflage" avec les données locales pour leur exécution. Le GA est en fait un acteur (ou un ensemble d'acteurs résidents).

- Les autres entités

Ces entités sont externes au noyau ; elles comprennent :

- les éléments d'environnement local nécessaires à l'exécution des acteurs. Ce sont des ressources spécifiques au site local (par exemple, existence d'un périphérique audio, de couleurs pour l'affichage, de certaines polices de caractères).

- *l'auberge* qui correspond à un environnement d'exécution où se trouve un messager en cours d'exécution Dans ce cas, on parlera de messager "actif" ou d'acteur

- *l'entrepôt* qui regroupe les messagers inactifs en attente d'un événement les concernant (acteurs "dormants" ou "en attente") et les messagers ayant été traités en partance vers d'autres sites Il correspond typiquement à un espace disque dans lequel sont entassés les acteurs devenus inactifs

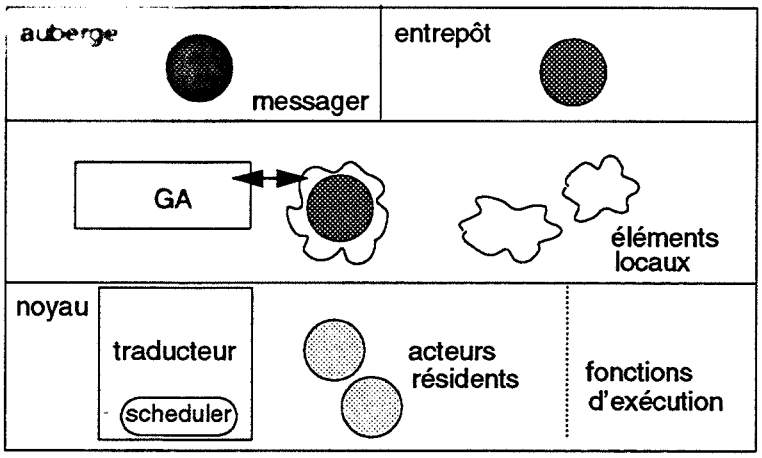


FIGURE 8 : *système de messagers*

3.2. Propriétés

- Adaptation à l'environnement

Un acteur doit pouvoir déclencher une suite ou une composition de comportements élémentaires. De plus, il doit pouvoir changer de comportement pour s'adapter à ce qu'on lui demande (par exemple, si l'on remplace une pièce par une autre référence équivalente). Il peut obtenir de l'aide pour réaliser localement un travail dans le cas où il ne connaît pas le moyen de le faire. Par exemple, il peut demander comment on effectue localement une fonction spécifique et agir selon la réponse sur son comportement. Ce mécanisme permet à un acteur de s'adapter à l'environnement dans lequel il vient d'arriver. En effet, l'acteur n'a pas besoin de savoir comment est réalisée localement une action, il suffit qu'il soit capable d'envoyer une demande d'exécution à l'acteur approprié. Cet acteur résident doit être capable de répondre à sa demande de façon locale. Il peut aussi recevoir un signal envoyé par un autre acteur (par exemple, un acteur résident) pour lui indiquer qu'un événement exceptionnel vient d'arriver (arrêt de la machine de fabrication). Enfin, un acteur peut s'abonner à des services ou obtenir des privilèges. Par exemple, il peut demander à être réveillé par un autre messager en cas d'attente d'informations, ou être autorisé à modifier le comportement d'autres acteurs. Ces services et privilèges ne sont valables que localement et disparaissent au moment où le messager n'est plus géré par le gestionnaire local.

- Délocalisation

Contrairement aux modèles traditionnels dans lesquels les acteurs sont utilisés surtout dans des environnements fortement couplés (contexte de parallélisme) [BOUD 91], nos messagers sont répartis sur des machines faiblement couplées.

Un messager est localisé sur une seule machine à la fois mais peut envoyer des clones sur une autre machine pour répondre par exemple, à un besoin de diffusion d'informations (demande de votes). Il peut aussi être transféré d'un site à un autre par un phénomène de *migration*. Ceci consiste en la création d'un nouveau messager sur le site récepteur avec les caractéristiques du messager de référence sur le site originel et en la suppression de ce dernier. Le messager créé a le même comportement que son modèle de référence ; il n'y a donc aucune différence observable entre les deux.

- Sous-traitance

Dans les modèles à acteurs traditionnels, Il n'y a pas de notion d'héritage ; chaque acteur est totalement autonome et contient toute la connaissance nécessaire pour remplir son rôle. Le seul moyen pour un acteur d'acquérir de nouvelles connaissances est d'utiliser un mécanisme de délégation. La délégation est une forme d'héritage puisque l'acteur à qui est déléguée une tâche a accès aux informations détenus par l'acteur délégant. On établit une relation de dépendance mutuelle entre des acteurs qui peuvent ainsi partager des comportements [LIEB 86].

Dans notre cas, si un acteur ne peut traiter une communication directement (par exemple, si l'acteur n'a pas le droit d'exécuter la tâche demandée), il peut confier tout ou partie de sa tâche à un autre acteur existant ou créé expressement pour cela. Il peut si

nécessaire lui communiquer aussi la promesse transmise par l'acteur à qui doit être délivrée la réponse. Nous utilisons alors un mécanisme de *sous-traitance*. Contrairement à la délégation, l'acteur sous traitant est indépendant de l'acteur principal et agit indépendemment de son environnement. Cependant, l'acteur principal peut transmettre au sous traitant des références (pointeurs) sur certaines entités de son environnement, ce qui permet à l'acteur sous-traitant de modifier effectivement l'état de l'acteur principal.

3.3. Implantation

La transmission "physique" des messagers repose sur des outils de communication asynchrones existants (par exemple, la messagerie électronique).

Dans un système de messagers, le noyau sera implanté dans un langage d'implantation propre à la machine ; c'est un langage compilé proche de la machine, efficace mais peu souple (par exemple le langage C).

Les acteurs sont programmés dans un langage d'extension interprété et de plus haut niveau qui permet de définir dynamiquement de nouvelles fonctions au sein du système [GIRA 91]. On en trouvera en annexe une description plus détaillée (cf ANNEXE A). L'utilisation d'un langage de programmation puissant pour traduire les comportements apporte la souplesse d'adaptation au système.

4. AVANTAGES DU MODELE

Notre modèle est un modèle de communication par agents reposant sur un langage à acteurs. On utilise un modèle à objets qui apporte les avantages bien conus de développement modulaire et d'encapsulation. Le langage sous jacent utilisé est un langage à acteurs plutôt que par classes [UNGA 87]. En effet, l'utilisation d'acteurs présente plusieurs avantages :

- Le premier point est l'autonomie apportée par l'approche acteur contrairement à une approche objet pure. L'utilisation de classes implique un mécanisme descriptif centralisateur. Une instance se réfère à sa classe pour déterminer son comportement : ce n'est manifestement pas envisageable lorsque l'instance est en activité sur une partie du réseau très éloignée de celle où la classe est définie. Il n'est pas possible non plus de concevoir un ensemble de classes figées. Les besoins des utilisateurs, qui sont très divers impliqueraient la création locale de classes ad hoc, et la diffusion universelle de leurs descriptions. Dans le modèle acteur, au contraire, chaque acteur est autonome et se suffit à lui-même puisqu'il contient tous les éléments nécessaires à son fonctionnement.

- Un tel modèle gagne fortement en souplesse. En effet, il est adaptable aux spécificités des applications qui doivent communiquer entre elles. L'approche par acteurs permet de définir des messagers ayant un comportement particulier selon les besoins, alors que dans un modèle à classes, la représentation d'un comportement spécifique entraîne la création d'une nouvelle classe (avec souvent un petit nombre d'instances). Si la conception des scripts peut être complexe selon le cas à traiter, elle reste néanmoins souple et adaptée aux spécificités requises. De plus, si cette conception est difficile à mettre en oeuvre, il est

presque certain que la greffe du nouveau cas directement dans l'application existante soit encore plus complexe.

Ces divers avantages permettent de penser que notre modèle apporte un confort et une souplesse supplémentaires par rapport aux approches classiques par objets. De plus, l'utilisation d'acteurs dans notre modèle permet de répondre à d'autres besoins particuliers :

- Dans notre modèle, un acteur peut demander comment s'effectue une tâche localement ou demander sa réalisation par un autre acteur. Ceci évite d'inclure dans chaque messager tout le comportement nécessaire. Notre acteur s'adapte ensuite à l'environnement dans lequel il vient d'arriver.

- Un acteur est un objet dynamique qui peut modifier son comportement en créant d'autres acteurs. A chaque nouvelle création, il est possible de garder une trace et de représenter ainsi l'historique de l'échange entre applications. Dans ce type de modèle, un acteur peut réagir à un message en changeant son comportement. La composante historique joue donc un rôle important, ce qui n'est pas le cas, par exemple, en programmation fonctionnelle. Dans un cas comme celui de la demande d'analyse médicale, il est possible de garder les transferts du malade pour transmettre les résultats d'analyse au service concerné en créant un acteur chargé du suivi du malade.

- Il est possible de mettre l'accent sur les services qui sont formalisés par des messagers.

- Il n'y a pas de règles de circulation liées aux messagers. Tout est paramétrable par programmation des acteurs. En particulier, il n'existe pas de séquentialité présumée ; les messagers peuvent agir dans n'importe quel ordre et concurremment. Dans un exemple comme celui du projet architectural, si les divers documents sont véhiculés par des messagers, ceux-ci peuvent transiter entre de nombreux partenaires et prendre en compte les modifications et retours arrières éventuels.

En résumé, notre modèle de communication permet :

- d'adapter les logiciels entre eux,
- de gérer des scénarios de circulation plus élaborés.

Il apparaît aussi comme complémentaire vis à vis des normes existantes puisqu'il permet leur utilisation en servant de moyen d'interchange entre elles dans la mesure du possible. Par exemple, si une application n'accepte que des documents EDI et qu'une application souhaite communiquer avec elle mais n'est capable que de générer un fichier, il est possible de traduire les informations en un document EDI par un acteur représentatif capable de générer ce type de document.

La technique des messagers recouvre divers domaines de communication dont la messagerie multimédia. Nous allons montrer dans les deux chapîtres suivants en quoi elle est particulièrement adéquate dans ce type d'application en nous appuyant sur les spécificités des informations transmises.

CHAPITRE 4

Messages multimédias

Un message multimédia est un ensemble d'informations complexes réunies dans une seule structure. Parmi ces informations, on distingue celles qui sont relatives à la présentation du message, essentiellement son architecture spatio-temporelle et celles qui sont liées au comportement du message et aux manipulations effectuées sur le message. Ce chapitre a pour but de présenter ces deux aspects. Nous ne nous intéresserons pas aux types des données véhiculées, mais essentiellement à la structure du message et à son comportement.

1. ARCHITECTURE SPATIO-TEMPORELLE

1.1. Présentation

De façon générale, un message multimédia regroupe dans une structure unique, en l'occurrence le corps du message, des entités d'informations auxquelles est associée une certaine forme de représentation. On désignera par la suite ces entités sous le nom d'*objets multimédias*[1]. Ce sont, par exemple, un paragraphe de texte, une image ou un enregistrement sonore. Dans un message multimédia, ces objets sont articulés autour de deux notions de base qui sont l'espace et le temps. Le premier correspond à la disposition des différentes parties dans l'espace d'affichage du message[2], le second reflète l'ordonnancement des objets dans le temps. On ajoute ainsi une autre dimension à la notion classique d'espace rencontrée dans n'importe quel document. Ceci est particulièrement vrai dans un message où l'ordre de restitution est significatif ; par

1. Dans l'ensemble de ce chapitre, le terme objet sera pris dans le sens d'entité physique ou logique sans rapport avec les objets acteurs décrits précédemment.

2. On associera le terme de message au corps du message en faisant exclusion de l'en-tête.

exemple une annotation vocale d'une image impose que l'image soit affichée avant le son. Cet aspect est aussi primordial dans un message où l'on inclut des séquences vidéos ou des images animées qui sont des objets fortement dépendants du temps.

Un objet multimédia peut être un élément simple comme une image statique ou un texte ou encore un élément plus complexe comme une vidéo. En fait, une vidéo est un objet regroupant du son couplé à des images fixes qui sont affichées en séquence. Il est donc toujours possible de décomposer un objet complexe en éléments simples. On définit comme *élémentaire* tout objet ne pouvant être décomposé en sous éléments (par décomposition en intervalles de temps). Un objet regroupant plusieurs objets élémentaires est qualifié de *composite*. On ne se référera par la suite qu'aux objets élémentaires.

Outre la combinaison de l'espace et du temps, un message multimédia peut inclure une certaine interactivité. On peut ainsi prévoir un "dialogue" avec le destinataire du message au travers de composants graphiques (exemple : boutons) qui déclenchent des actions. Ces composants ne véhiculent pas d'information par eux-mêmes, mais déclenchent une action qui peut influer sur la présentation du message ou sur l'environnement de l'utilisateur. On appellera ces composants des *objets actifs*.

Prenons l'exemple suivant d'un cas simple de message multimédia interactif comportant trois objets multimédias : un texte, une image et un son, ainsi qu'un objet actif représenté par un bouton (cf FIGURE 9).

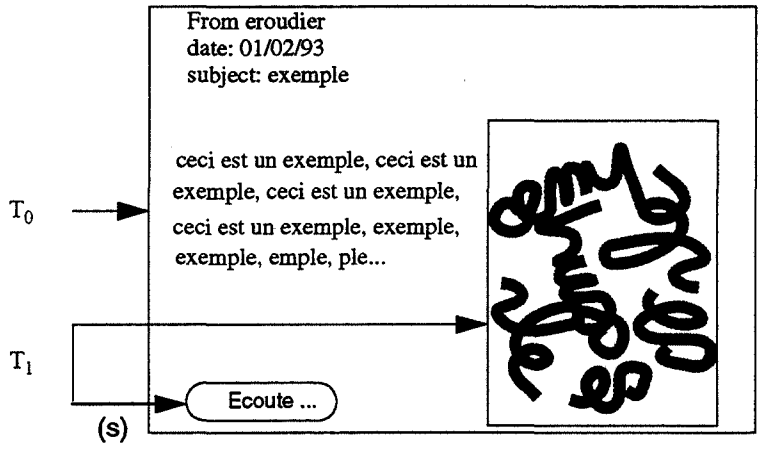


FIGURE 9 *: message multimédia interactif*

Supposons que dans cet exemple, le message se déroule temporellement de la façon suivante :

t_0 : affichage du paragraphe de texte,

t_1 : affichage du graphique et du bouton d'interaction.

quand on clique sur (s), la partie sonore est jouée.

Si on sépare l'aspect visuel, de l'aspect temporel du message, on obtient deux représentations différentes (cf FIGURE 10).

La première figure schématise la structure spatiale du message. Chaque objet multimédia est représenté par un bloc d'affichage disposé dans la fenêtre. Chaque bloc correspond au plus petit rectangle englobant de la surface affichée de l'objet correspondant. Leur agencement respecte l'aspect visuel du message.

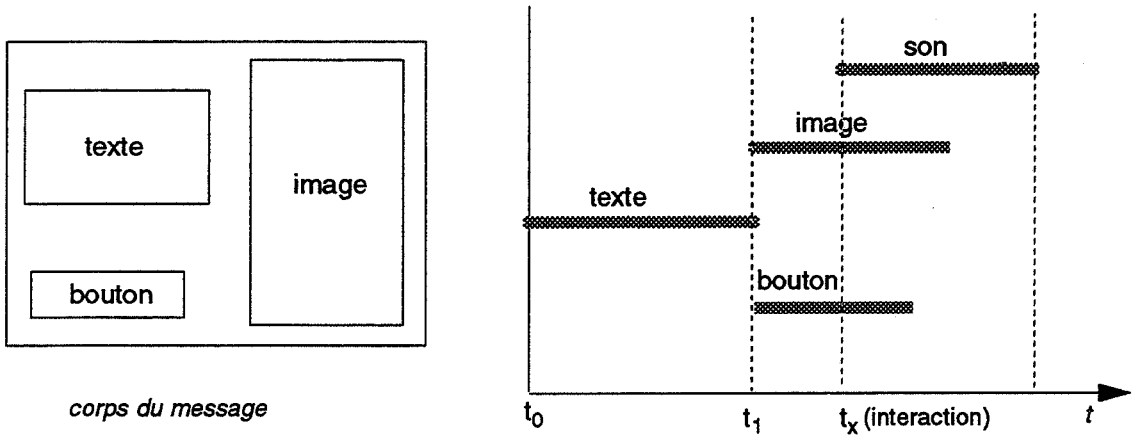


FIGURE 10 *: organisation spatiale et temporelle du corps du message*

Le second schéma formalise la structure temporelle du message. Cette figure permet de visualiser les relations temporelles des différents composants du message entre eux. Chaque flèche correspond à la restitution, à partir d'un instant t_i, d'un objet. Nous voyons ainsi que la restitution des objets *bouton* et *image* est simultanée puisque leur affichage est déclenché en parallèle ; elle est séquentielle par rapport à l'objet *texte*, la terminaison de ce dernier entraînant l'affichage de chacun. La production de l'objet *son* est indépendante des autres objets car elle est déclenchée lorsque l'utilisateur clique sur le bouton. Cette interaction est représentée par l'instant t_x.

Nous venons de voir qu'un objet multimédia est une entité logique ayant deux représentations, l'une visuelle, l'autre temporelle. Seuls les objets de type son (voix, musique) ont une unique représentation dans le temps. Un message multimédia peut donc être vu comme un assemblage d'objets multimédias véhiculant des informations reliés les uns aux autres par leurs structures spatiales et/ou temporelles auxquelles s'ajoutent des interactions utilisateurs ou des actions prédéfinies

1.2. Architecture spatiale

La composition spatiale d'un document multimédia consiste à assembler des informations dans une représentation spatiale qui est fixe à un instant donné. Elle comporte deux aspects, la hiérarchie spatiale du document, explicitée par les relations spatiales des objets entre eux et les transformations spatiales appliquées aux objets.

1.2.1. Relations spatiales

Si on suppose deux objets élémentaires multimédias (A et B) ayant chacun une représentation visuelle (B_a et B_b) et un espace d'affichage (E) (par exemple une fenêtre), on définit les relations spatiales suivantes (cf FIGURE 11) :

- A et B sont spatialement indépendants ; l'affichage de l'un n'est pas relié à l'affichage de l'autre. C'est le cas où chaque objet est affiché dans une fenêtre différente ; l'espace d'affichage de A est distinct de l'espace d'affichage de B. Si les objets appartiennent au même espace d'affichage ($E_a = E_b = E$) alors leurs blocs correspondants sont disjoints. L'espace d'affichage de B correspond à l'espace d'affichage total excepté le bloc associé à l'objet A ($E_b = E - B_a$).

- A et B sont spatialement dépendants ; la position spatiale de l'un dépend de la position de l'autre. Ceci n'est possible que si A et B ont le même espace d'affichage. Dans ce cas, les objets ont une partie d'affichage commune ; leurs blocs sont alors tout ou en partie superposables (B_a est inclus dans B_b ou l'inverse).

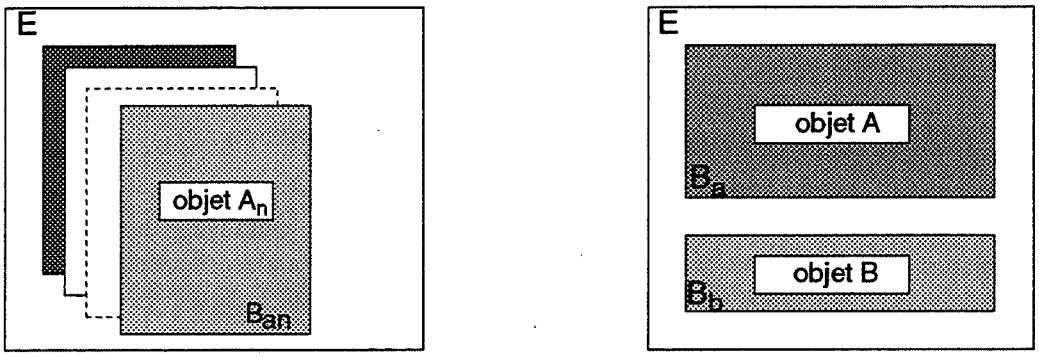


FIGURE 11 *: objets superposables ou disjoints*

Ces deux schémas sont des exemples, il existe ensuite de multiples variantes selon la disposition des objets : à gauche, à droite, au-dessus, en-dessous. Le premier exemple correspond à une séquence animée dans laquelle plusieurs images se surperposent les unes aux autres pour donner l'impression de mouvement (principe du dessin animé). Le second cas est une configuration classique de n'importe quel document comme par exemple, la disposition d'une image à côté d'un texte.

1.2.2. Transformations spatiales

Ces transformations sont dites spatiales car elles agissent sur l'architecture spatiale du document. Elles permettent de modifier dynamiquement la structure spatiale et sont ponctuelles dans le temps. Ce sont, par exemple, des transformations telles que le déplacement d'objets à l'intérieur d'un espace d'affichage ou entre espaces d'affichage, l'effacement et le ré-affichage d'objets, etc. Ainsi, une transformation spatiale peut agir

sur la hiérarchie spatiale du document. D'autres transformations plus complexes peuvent être envisagées comme la mise à l'échelle, ou le grossissement d'un objet [LITT 91].

1.3. Synchronisation

La synchronisation est nécessaire pour assurer l'ordre temporel des événements[1] dans n'importe quel système multimédia. Il existe deux aspects du problème : la définition des relations temporelles entre les objets, et la restitution des données selon ces relations. Nous nous intéresserons ici surtout au premier cas, le second étant un autre problème lié essentiellement à des caractéristiques d'implantation que nous verrons par la suite.

Les contraintes de synchronisation incluent trois types de contrôles de présentation qui sont considérés comme suffisants pour la synchronisation multimédia : ordre séquentiel, simultané et indépendant. Le premier correspond à deux séquences temporelles qui se suivent, la terminaison de l'une entraînant le commencement de l'autre. Le second type de contrôle consiste en deux séquences qui sont parallèles (début et/ou fin identiques). Le contrôle dit indépendant indique que les séquences ne sont pas reliées les unes aux autres et qu'elles se déroulent donc de manière autonome. C'est le cas par exemple de séquences déclenchées par des événements asynchrones (entrées clavier, souris).

1.3.1. Concepts généraux

Le mécanisme qui consiste à relier les objets selon un ordonnancement temporel est lié aux concepts d'actions et d'événements.

- Un *événement* est le déclencheur d'une action. Pour synchroniser les objets, on définit des points de repère dans le temps. Ceux-ci sont des événements particuliers nommés *points de synchronisation*. Ils correspondent le plus souvent au début ou à la fin d'une action ou à un événement spécifique (interaction de l'utilisateur, objet particulier). Par exemple, on peut définir des points de synchronisation tels que `begin`, `end` ou `interrupt`. Les points de synchronisation peuvent être déterminés comme des unités de temps selon un système temporel de référence ou en termes de relations temporelles avec d'autres actions. En effet, un point de synchronisation d'une action est souvent égal ou relatif à celui d'un autre action.

- Une *action*, de façon générale, est quelque chose qui s'exécute. Dans notre contexte, une action correspond essentiellement à la restitution d'un objet multimédia (affichage ou production de sons). Les actions pour lesquelles il n'existe pas de points de synchronisation durant leur exécution sont dites *atomiques*. Un ensemble d'actions élémentaires est une *action composite*. Par exemple, afficher 10 fois une image fixe est une action composite -répéter 10 fois- d'actions élémentaires -afficher l'image-. On distingue aussi les actions *ponctuelles* qui ont une durée d'exécution quasi-immédiate (cas des affichages) et les actions *continues* qui ont une durée définie dans le temps.

1. Ici événement a le sens traditionnel de "quelquechose qui arrive".

A cela correspondent des objets qui peuvent être classés selon leur temps de vie dans le message. On distingue les objets *persistants* et les objets *non persistants*. Les premiers correspondent à des objets dont la représentation a une durée plus ou moins longue mais qui restent présents dans le message comme par exemple une image statique. Un objet non persistant est un objet qui a une durée mais qui disparaît une fois sa restitution terminée comme par exemple une séquence sonore.

1.3.2. Relations temporelles

On associe à une action un intervalle de temps qui correspond à sa durée. Pour deux intervalles, il existe un nombre fini de relations temporelles qui sont : `before`, `meets`, `during`, `overlaps`, `starts`, `ends`, `equals` et leur inverse [LITT 90]. On peut classer ces relations en deux groupes : les relations séquentielles et les relations parallèles. Soient deux actions A et B auxquelles on associe un intervalle de temps délimité par deux points de synchronisation : le point de départ et le point de terminaison. Il existe six types principaux de relations (cf FIGURE 12).

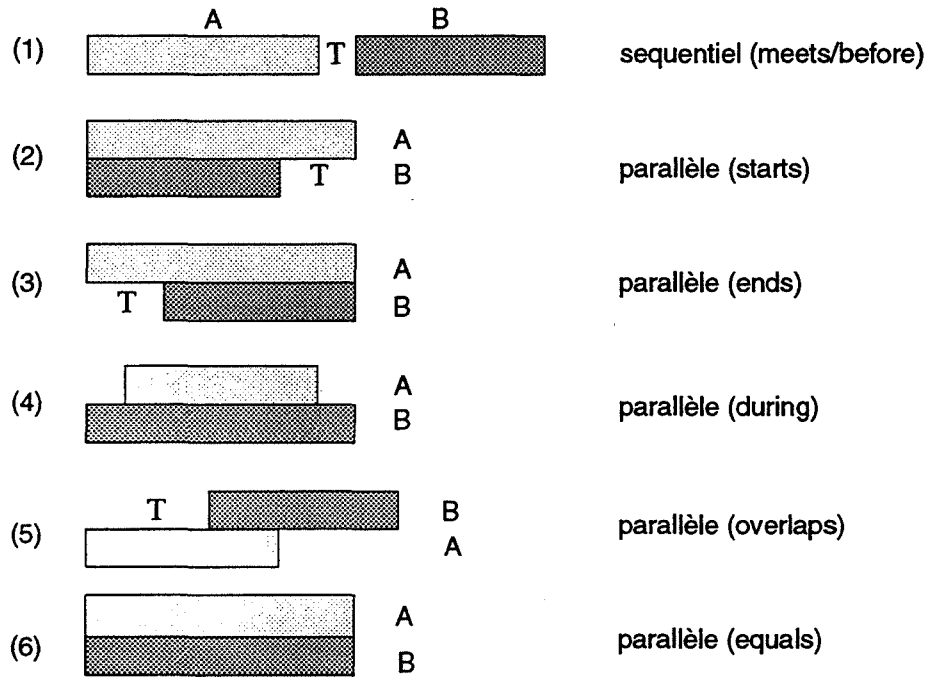


FIGURE 12 : *relations temporelles entre deux intervalles de temps*

(1) L'action B n'est lancée que lorsque A est terminée. S'il n'y pas de délai prévu (T) entre A et B, le point de départ de B est le point de terminaison de A, sinon A et B sont indépendants.

(2) A et B sont exécutées en parallèle et leurs points de départs sont identiques. L'action suivante débute lorsque A ou B est terminée.

(3) A et B sont exécutées en parallèle et leur points de terminaison sont identiques. Dans ce cas B commence après un délai T. L'action suivante débute quand A et B sont terminées.

(4) A et B sont exécutées en parallèle et leur points de départ et leur points de terminaison sont différents. A commence après B ; l'action suivante débute lorsque A ou B est terminée.

(5) A et B ont une partie de leurs exécutions en parallèle et leur points de départ et leur points de terminaison sont différents. B commence sans attendre la fin de A. L'action suivante débute lorsque B est terminée.

(6) A et B s'exécutent concurremment. Leurs points de synchronisation sont identiques.

A partir de ces relations temporelles de base, on peut obtenir de nouvelles synchronisations en incorporant des délais T entre les actions.

En utilisant ces relations temporelles entre les objets, on peut définir un modèle de synchronisation comme celui proposé par [YAVA 92]. Ce modèle utilise un graphe direct appelé CSG (Conversation Synchronisation Graph) pour décrire les relations temporelles entre les unités d'informations multimédias. Les noeuds du graphe représentent les unités d'informations comme les images et les mémoires d'écrans vidéo. Chacune est associée à un intervalle de temps (début et durée), le point de départ pouvant être relatif à d'autres unités. Les unités d'information sont synchronisées par leur placement dans le graphe les unes par rapport aux autres et selon les autres types de noeuds qui sont des points de synchronisation. Il est possible aussi de formaliser la structure temporelle des messages par des réseaux de Pétri comme le propose Little [LITT 90], mais l'ajout d'interaction utilisateur est difficilement représentable.

2. COMPORTEMENT

Nous venons de décrire les différentes informations de présentation véhiculées par les messages multimédias. Outre les spécificités spatio-temporelles, un message comporte aussi des informations spécifiques de son comportement. Si un message est le plus souvent une source d'informations, il peut aussi avoir un rôle à jouer selon l'application ou le destinataire.

2.1. Messages actifs

Les messages actifs sont des messages qui agissent dans l'environnement dans lequel ils se trouvent pour, par exemple, exécuter des programmes ou lancer d'autres messages. L'intérêt des messages actifs est de générer automatiquement des actions prédéfinies. Nous avons vu que pour cela, on peut utiliser des objets actifs qui entraînent une action particulière. Ces objets actifs, tels que nous les avons définis ont une représentation visuelle ou non, et sont associés à un ensemble d'actions asynchrones. Les possibilités engendrées par les messages actifs sont multiples. Prenons par exemple, le cas du questionnaire (pour enquête ou statistiques). L'expéditeur prévoit dans son message une suite de questions et de réponses possibles. Ces questions peuvent prendre différentes

formes : texte, questions orales. Les réponses sont prévues à l'avance dans le message et guident la suite du comportement du message. Par exemple, elles peuvent fournir au fur et à mesure de nouvelles informations ou activer de nouveaux messages (demandes d'informations supplémentaires sur le sens d'une question). Le questionnaire rempli peut être envoyé automatiquement à un ou plusieurs destinataires chargés de récolter et de traiter les réponses.

Les messages actifs posent le problème de la sécurité. Si l'on offre le pouvoir de réaliser un travail général et utile, on donne obligatoirement le pouvoir de causer des dommages. On ne peut jamais connaître les conséquences entraînées par l'exécution d'un message actif. Ces problèmes de sécurité ne sont pas liés qu'aux seuls messages dits "actifs" mais aussi à des messages n'entrainant pas d'actions (messages "passifs") mais qui sont tout aussi nocifs pour une entreprise ou une personne. C'est le cas, par exemple, d'une commande importante qui entraîne des investissements conséquents pour une société et qui se révèle fausse par la suite. Pour assurer une certaine sécurité, cela suppose de pouvoir authentifier de manière absolue l'expéditeur et, en cas de problèmes, savoir quel message est responsable de quelle action ; d'où la nécessité de garder un historique à moyen ou long terme des messages (provenance, contenu, déroulement de l'exécution et résultats s'il y a lieu).

2.2. Devenir des messages

Le plus souvent les messages sont amenés à circuler entre deux ou plusieurs partenaires (par exemple, appartenant à une liste de distribution). Il est donc utile de connaître où le message doit être envoyé et de manière générale ce qui doit en être fait. Le message peut transiter entre différents organismes, donner lieu à d'autres transmissions ou être simplement stocké automatiquement. La plupart du temps, ce type d'informations n'est pas fournie dans le message ; c'est le destinataire qui est responsable de la suite à donner à ce message. Il est donc tout à fait envisageable de formaliser ce devenir de manière automatique. Par exemple, un message peut contenir un renvoi automatique à l'expéditeur, une fois le message consulté. Il est possible d'ajouter des informations sur les parties modifiées ou ajoutées (de quelle manière et par qui). On peut garder ainsi l'historique du message.

Un exemple d'un tel message est la constitution et le suivi d'un dossier médical. Au départ le message est constitué d'informations générales sur le patient puis il est complété au fur et à mesure par le laboratoire d'analyse, les divers médecins, les spécialistes ou l'administration. Le message va ainsi transiter entre divers partenaires selon un ordre pré-établi ou non, chacun pouvant ajouter ses propres remarques et de nouvelles informations (nouveaux médicaments prescris, résultats d'analyses). L'ensemble des modifications doivent être sauvegardées avec le type de données, la date, la provenance, etc, de manière à constituer le suivi du dossier. Un médecin peut envoyer certaines parties du message à un confrère (par exemple, un détail de radiographie sélectionné) pour confirmer un diagnostic et générer ainsi un nouveau message.

3. SCÉNARIO DE MESSAGES

3.1. Référence cinématographique

Un message multimédia est un assemblage complexe d'informations sémantiques (les informations véhiculées) et d'informations de présentation ou de comportement du message. Il doit donc pouvoir formaliser des expressions du genre *"image à gauche du texte"*, *"graphique accompagné de voix"* ou encore *"si réponse ok faire suivre à Untel"*. Ces relations correspondent à ce que l'on peut définir comme un scénario.

Au sens cinématographique, le scénario sert de base de travail au metteur en scène pour réaliser le film [REIS 68], [JONE 74]. C'est un document qui précise l'histoire, les décors, les acteurs et leurs rôles. Il peut aussi contenir des informations sur la variété des sons, les répliques des artistes, les effets de lumière, le nombre et la position des caméras, les détails des entrées dans le film, etc. En fait, le scénario contient toutes les données qui peuvent être importantes pour le réalisateur. Le public voit un film comme une entité unifiée et linéaire, tandis que le réalisateur conçoit le film comme la génération de plans individuels et d'éléments sonores et leur assemblage. Un plan ou prise de vue contient quatre aspects : le point de vue que l'image représente (la perspective), comment cet environnement est capturé par la caméra et le son (enregistrement), ce qui se passe (le contenu), et les relations avec l'ensemble de la scène (le contexte).

exemple de scénario de prises de vues :

Plan	*caméra*	*décor*	*action*	*son*
1	caméra 1 - pos A Fred	la maison de Zoé le seuil.	Fred frappe à la porte et entre	toc toc porte qui grince
2	caméra 2 - pos D Zoé	l'entrée	Zoé : «bonjour Fred»	

On retrouve les mêmes concepts dans un message multimédia. Chaque plan correspond à une action effectuée dans l'espace et/ou le temps : affichage d'une image, production d'un son et reflète une situation à un instant donné. De la même façon, le scénario d'un message multimédia détermine les objets composant le message, l'aspect visuel et le déroulement du message et les actions à réaliser. Il existe donc une certaine analogie entre les deux domaines [DAVE 91]. Ce type de scénario est cependant plus simple à concevoir par le nombre restreint de données entrant en jeu. Une autre différence essentielle provient des interactions possibles de l'utilisateur qui sont inexistantes dans un scénario de cinéma. Certains jeux télévisés dans lesquels le spectateur agit sur le déroulement de l'action ou sur les personnages constituent un exemple de ce qui pourrait être réalisé.

3.2. Exemple de scénario

Prenons un exemple de scénario plus ou moins complexe de message multimédia incluant divers types de médias et des interactions de l'utilisateur (cf FIGURE 13). Dans

cet exemple, le scénario se déroule pendant un temps *t* sur deux dispositifs de sortie : l'un visuel et l'autre auditif. Plusieurs types de médias sont en jeu : des textes, des images fixes, du son seul et une séquence vidéo qui regroupe une suite d'images couplées avec un son. Le scénario indique le dispositif de sortie utilisé par chaque objet média et la position de chacun dans le temps. Il indique aussi la synchronisation des différents types d'objets multimédias les uns par rapport aux autres de façon séquentielle ou en parallèle. Dans le cas d'objets séquentiels, la restitution d'un objet peut être déclenchée par un autre objet (exemple : texte 2, image N° X) ou sur une action de l'utilisateur. Dans le cas d'objets simultanés, plusieurs objets sont synchronisés à partir d'un point de départ (exemple : images de 1 à n + son). Il faut noter que deux objets séquentiels peuvent être séparés par un certain délai. Un scénario peut être interrompu pour laisser la place à un autre, par exemple sur une réponse du destinataire ; il est possible ainsi d'avoir une suite de scénarios pour un même message, chacun étant déclenché selon certains critères.

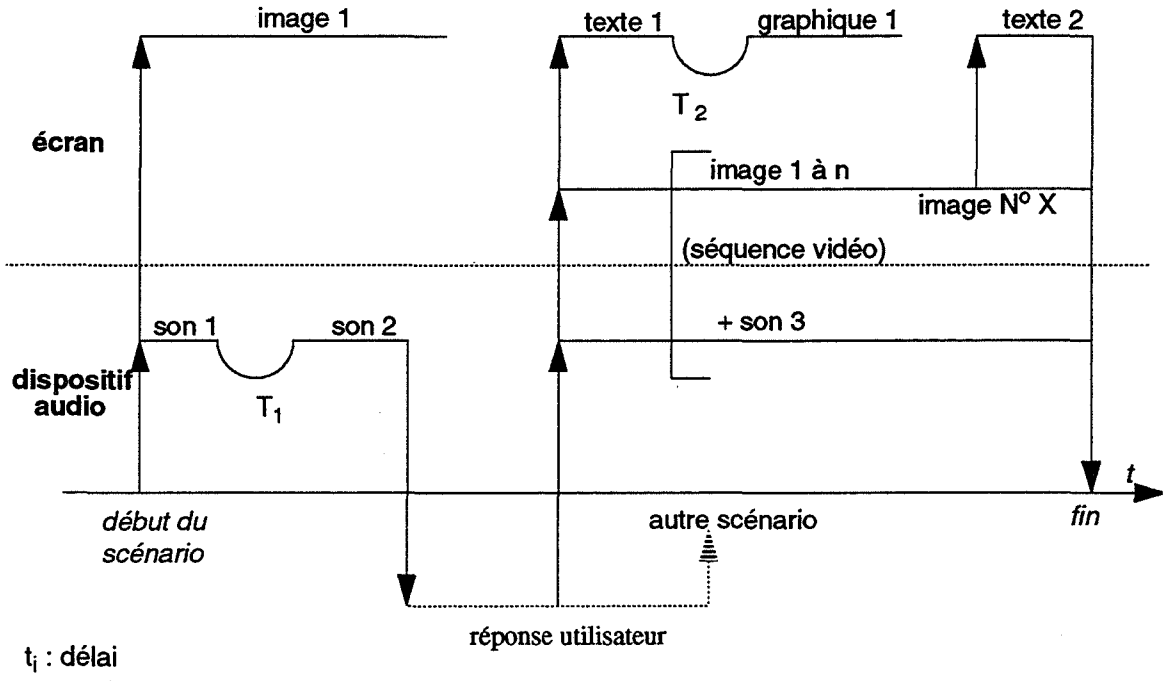


FIGURE 13 : *exemple de scénario*

La notion de scénario tel que le montre cet exemple formalise parfaitement l'architecture du message multimédia car il représente de manière précise les relations spatiales et temporelles des composants du message. Il peut aussi représenter facilement les transformations sur les objets et les transitions entre eux ou un quelconque comportement du message. Nous allons maintenant voir comment il est possible de représenter ce type de scénario et en quoi l'approche par scripts est adaptée.

CHAPITRE 5

Représentation

Ce chapitre fait la synthèse des deux chapitres précédents. Le premier décrit le modèle de communication proposé ; le second détaille les caractéristiques des messages multimédias. Nous allons maintenant montrer en quoi le modèle est adapté à la représentation des messages multimédias tels que nous les avons décrits. Pour cela, nous étudierons les diverses approches possibles pour prendre en compte tous les aspects des messages manipulés.

1. CHOIX D'UNE REPRÉSENTATION

Il s'agit de représenter le scénario du message c'est à dire son architecture spatio-temporelle, les interactions et les actions possibles. Ceci peut être réalisé de plusieurs façons.

1.1. Description structurée

Une solution est de l'inclure dans le message lui-même par une description structurée. Cette solution a été choisie dans le système expérimental MMS *(Multimedia Mail System)*. Le modèle développé dans ce système permet au créateur du message de contrôler le séquencement des différentes parties. Cette coordination est réalisée en incorporant explicitement des contrôles de séquences dans le corps du message. Le message multimédia est construit à partir de descripteurs de présentation. Chaque descripteur est une liste de propriétés de séquencement définie par un nom de propriété. Trois types de propriétés sont définis : séquentiel, simultané et indépendant. Par exemple, si une figure est accompagnée d'une voix, les éléments de présentation seront regroupés sous la propriété "simultané". La valeur de la propriété est une liste d'éléments de présentation qui regroupent l'objet multimédia et des informations d'affichage (représentation spatiale) ou, pour des messages plus complexes, une autre liste de descripteurs de présentation.

L'information de contrôle spatial décrit la zone correspondant à l'objet dans la fenêtre du message, relativement à son origine.

Le même type de solution est proposé pour la présentation des objets multimédias dans les documents ODA. Le standard prend en compte la disposition des différentes parties du document au travers des structures physiques ("layout structure"). Des extensions au niveau de ces structures ont été proposées pour prendre en compte les aspects temporels [HOEP 91]. La synchronisation du document est exprimée par de nouveaux attributs associés aux objets primitifs ("basic layout objects") ou composites ("composite layout objects") relatifs à la disposition des éléments. Les objets de base ont ainsi des attributs indiquant le type temporel de contenu (statique ou dynamique selon que la portion de contenu qu'il représente est dépendante ou non du temps) ou leur durée. Les objets composites contiennent des attributs de synchronisation dont les valeurs déterminent un déroulement séquentiel, parallèle ou répétitif de leurs subordonnés. Les structures logiques des documents ne sont pas pour l'instant directement concernées par les aspects temporels.

Dans les deux cas, l'architecture du message correspond à ce que l'on peut appeler un "arbre décoré". Le message multimédia est représenté comme une arborescence d'objets dont la racine est le corps du message et les feuilles les objets élémentaires associés à un type de médias. A chaque feuille, on associe des attributs de disposition et de séquencement qui reflètent les relations spatio-temporelles du message (cf FIGURE 14).

Si l'on se reporte à l'exemple de scénario (cf FIGURE 13), il est évident que sa représentation par une structure arborescente serait difficile à mettre en oeuvre. Une description rigide et complexe n'est pas adaptée à nos besoins. En effet, nous voulons transmettre des messages interactifs dans lesquels le destinataire peut agir. Nous avons de plus besoin d'exprimer des contraintes sur la façon dont ces messages doivent être restitués et ce qui doit être fait avec le message. D'autres approches sont envisageables, l'approche par objets et l'approche par messagers. Etudions les deux possibilités.

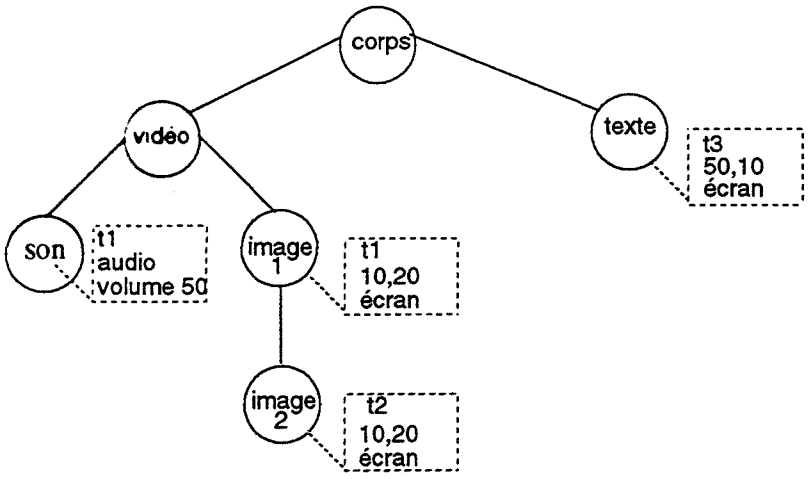


FIGURE 14 : *structure arborescente de représentation de scénario*

1.2. Approche objets

L'approche objet est souvent utilisée dans les modèles de composition multimédia. En effet, la vision objet est perçue comme naturelle dans le domaine multimédia. Elle est, par exemple, utilisée dans [GIBB 91] et [de MEY 92] pour résoudre les problèmes de synchronisation entre entités multimédias.

Voyons une approche objet que l'on pourrait imaginer à partir des caractéristiques des messages que nous avons vues au chapitre précédent. Nous venons de voir que les messages manipulés comportent différents objets multimédias (textes, images et sons), l'organisation spatiale du message et la synchronisation des différentes parties. S'il semble assez aisé de représenter les objets multimédias avec une approche objet, il est plus difficile de concevoir une configuration permettant d'englober l'ensemble de ces notions. Nous pouvons répertorier trois ensembles d'objets manipulés dans les messages. Ce sont :

- les objets du message : texte, graphiques et images, sons, événements qui correspondent à des entités logiques et/ou physiques.
- les objets "relations" qui représentent les relations et les contraintes entre les objets ci-dessus.
- les objets "machines" qui sont les dispositifs d'entrées/sorties.

Reprenons ces types d'objets en détail. Dans la première catégorie, on trouve deux sous-ensembles d'objets qui sont :

- *les objets multimédias :* Ce sont les parties du message qui correspondent à une entité logique comme un paragraphe ou une image. Chacun est associé à un type de média. Ils constituent la base des messages. Ces objets sont soit des objets élémentaires correspondant à une forme unique de représentation temporelle ou spatiale, soit des objets composites regroupant plusieurs objets élémentaires.

- *les objets actifs* : Ceux-ci peuvent avoir ou non une représentation visuelle dans le message (bouton ou événements). Ils peuvent agir sur la présentation du message, sur son déroulement mais aussi entraîner des actions au niveau du devenir du message (ce qui doit être fait avec le message) ou toute autre action voulue par l'expéditeur. Dans ce cas, ils sont spécifiques au message.

Dans la deuxième classe, sont répertoriés trois types d'objets "relations" selon le type de relation et les objets concernés :

- *les contraintes sur les objets :* elles correspondent à des spécifications sur un objet. On distingue les contraintes physiques qui sont dépendantes du type de représentation de l'objet. Par exemple, un son entraîne la spécification du volume sonore et éventuellement une représentation graphique pour avertir le destinataire (icône). A ces contraintes physiques, s'ajoutent des contraintes logiques comme les dépendances logiques entre objets. Par exemple, un commentaire texte peut accompagné une figure, si l'on déplace la figure, il faut prendre en compte également le déplacement du texte.

- *les contraintes sur les médias :* ce sont des contraintes entraînées par un type de média sur des entités physiques. Par exemple, un son entraîne la synchronisation du dispositif de sortie puisque deux sons ne peuvent être joués en même temps.

- *les références à des objets* : un objet peut être référencé pour être ré-utilisé par la suite dans le message. Par exemple, le même objet peut être affiché dans deux fenêtres ou à deux intervalles de temps différents. Ceci est surtout le cas pour des objets multimédias élémentaires.

La dernière classe contient un seul type d'objet :

- *les objets "média"*: ce sont les périphériques d'entrée/sortie tels que le clavier, l'écran, le microphone. C'est par eux que transitent les interactions de l'utilisateur de même que l'affichage des composants visuels et la production de sons.

Il est parfaitement envisageable de concevoir des classes et des objets pour représenter l'architecture spatio-temporelle et même les actions des messages. Il est aussi possible de décrire précisément les interactions et le comportement spécifique d'un message en utilisant un modèle-vue-contrôleur (MVC). Le principe est de formaliser une situation par un modèle dont on a, à un instant donné, une certaine vue, en fait une représentation possible du modèle. Le contrôleur permet de prendre en compte des informations en provenance de l'extérieur (organes de saisie) et modifier le modèle, modifications qui sont alors observables dans la vue. Le lien de dépendance reliant le modèle à ses vues permet ainsi d'informer de façon quasi-transparente les représentations des changements subis par le modèle. La triade M.V.C. est bien adaptée à des besoins de prototypage [KRIE 92] et peut être appliquée à nos messages multimédias. Le modèle représente le message en cours pour lequel un utilisateur a une certaine vision et sur lequel il peut agir par l'intermédiaire du contrôleur. Le contrôleur gère les événements d'entrée et permet de déclencher de nouveaux scénarios. Néanmoins dans des cas complexes de messages, le mécanisme de dépendance apparait comme limité.

A priori, une approche objet à classes est donc possible mais n'est pas forcément simple, ni nécessairement bien adaptée à nos besoins. En effet, cela supposerait de définir des classes d'objets spécifiques avec des méthodes suffisamment souples et puissantes pour prendre en compte tous les comportements possibles d'un message. Si cela peut être fait pour un type de comportement bien défini comme, par exemple, faire suivre un message, cela devient vite impossible à prévoir pour des actions complexes et qui peuvent évoluer dans le temps. Cela suppose aussi de transmettre les caractéristiques de chaque classe avec le message pour pouvoir restituer les objets sur la machine réceptrice. Par ailleurs, un objet est souvent unique en son genre, et il n'est pas vraiment utile ou judicieux de créer une classe pour lui seul. L'approche objet à classes n'est donc pas forcément bien adaptée aux messages multimédias que nous aimerions transmettre.

Nous nous sommes donc intéressés à une approche plus souple en utilisant des agents actifs autonomes. Ceux-ci sont représentés par des messagers qui empruntent des concepts aux modèles acteurs.

1.3. Approche messagers

Nous avons introduit au chapitre 3, un modèle général de communication entre applications. Le modèle proposé doit offrir les moyens de communiquer entre des applications, malgré le caractère spécifique lié au matériel ou aux protocoles de chacune. Ce modèle est construit autour de messagers qui assurent les besoins de communication. Le messager est constitué des objets à transmettre, accompagnés par un script. Ces derniers sont générés par l'application sans souci de transmission.

L'utilisation des scripts dans les applications informatiques est chose courante [WOOD 91]. On peut citer parmi ceux-ci, le langage HyperTalk d'HyperCard qui permet de créer de applications interactives performantes de manière simple [MAGR 90]. Par rapport aux formes structurées, les scripts ont l'avantage d'être souples, extensibles et aisément intégrables dans une application. Bien qu'en général ils soient formalisés sous forme de programmes, ils sont souvent plus faciles d'utilisation car leur syntaxe est simplifiée par rapport à la syntaxe souvent très rigide des langages traditionnels. Un script est un ensemble de fonctions associées à un objet qui décrit son comportement en dehors de celui qui lui est naturel. Le script est exécuté par l'objet lorsque qu'une certaine situation se produit ; c'est souvent l'envoi d'un message ou un événement particulier qui déclenche son exécution.

Dans notre cas, le script reflète les relations spatio-temporelles entre les parties du messages et s'appuie sur les entités que nous avons préalablement définis : entités[1]"multimédias", entités "relations", entités "contraintes" et entités "médias". Il représente les actions et les contraintes que l'expéditeur veut spécifier sur son message. Ces actions peuvent être des spécifications sur la position ou l'ordre des entités multimédias, des contraintes spécifiques au type de média, des interactions de l'utilisateur, des informations supplémentaires sur l'environnement utilisateur ou des actions spécifiques. Le script spécifie la position dans l'espace des entités multimédias à un instant donné ; il peut indiquer aussi leurs déplacements dans le même ou dans un autre espace d'affichage. Dans le temps, il indique les relations temporelles entre les diverses entités. Il peut incorporer des attentes, arrêter ou redémarrer une partie de message, répéter des actions. L'expéditeur peut inclure dans le script des interactions spécifiques (demandes de réponses) ou des actions déclenchées par des événements.

1.3.1. Exemple

Reprenons l'exemple du scénario du chapitre précédent (cf FIGURE 13) et supposons qu'il corresponde à une présentation multimédia des activités d'une entreprise de fabrication automobile qui souhaite informer ses divers partenaires. On désire obtenir un message qui débute par une présentation de la société avec une vue générale et un commentaire sonore de bienvenue. Si le destinataire souhaite en savoir plus sur les investissements, chiffre d'affaires, etc., il peut sélectionner un autre scénario par une interaction particulière (exemple : touche du clavier), sinon la présentation se poursuit par

1. On parlera ici d'entités plutôt que d'objets pour les distinguer des objets que sont les messagers acteurs.

l'affichage simultané d'un texte explicatif et d'une vidéo expliquant la fabrication d'un véhicule. Sont données aussi quelques indications techniques de conception sous forme de graphiques 3D. La sélection d'une image (par arrêt sur image) permet d'obtenir des informations complémentaires sur une étape de fabrication. La présentation se termine en donnant les coordonnées nécessaires à tout renseignement supplémentaire.

Imaginons maintenant le script qui lui correspond. Ce script est construit à partir des objets suivants:

- des entités multimédias élémentaires et composites : texte 1, texte 2, image 1, images vidéos, son 1, son 2, son 3, correspondantes aux vues de la société, aux explications sonores et à la séquence animée expliquant la fabrication des produits,

- des entités médias : écran (composé de deux fenêtres) et dispositif audio,

- des entités actives (question et réponse de l'utilisateur).

Pour concevoir le script correspondant à ce scénario, on a défini les mots clés : *début, fin, faire, afficher, jouer* (actions de restitution), *volume, sur, sous, dessus, à droite, à gauche* (spécifications de position ou de production) *avec* (synchronisation parallèle), *si... alors* (condition), *quand* (point de synchronisation), *pour... de...à...* (répétition). Ces mots clés sont des exemples ; de nombreuses variantes sont possibles. Le script résultant est un pseudo-algorithme qui exprime le scénario du message (cf FIGURE 16). On peut remarquer que bien que le scénario soit complexe, le script obtenu est simple et concis. Cet exemple montre qu'un formalisme par script est plus adapté que toute autre structure figée, car il apporte la puissance et la souplesse nécessaires à la conception de messages.

1.3.2. Fonctionnement

On obtient ainsi un principe de fonctionnement simple (cf FIGURE 15). A partir d'objets prédéfinis, le script formalise leur assemblage dans l'espace et le temps et indique toutes les actions concernant le message multimédia. Il accompagne les données sémantiques du message, qui est ensuite affiché chez le destinataire par l'exécution du script.

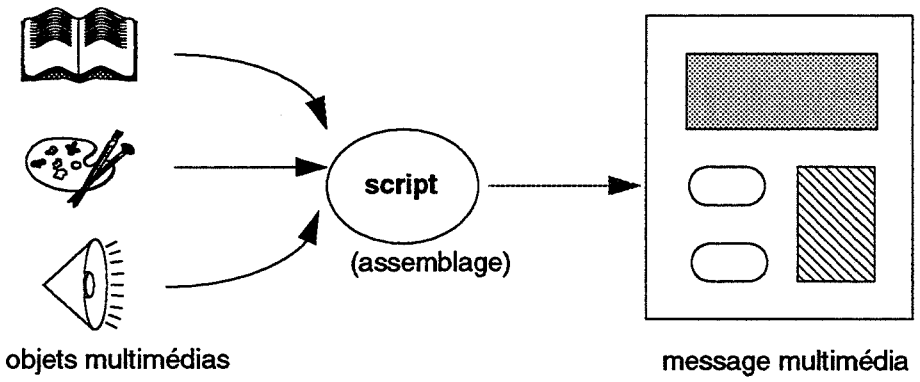


FIGURE 15 : *fonctionnement général*

```
DEBUT

  AFFICHER image 1 SUR fenêtre 1
  AVEC
  JOUER son 1 SUR sortie audio VOLUME 10

  QUAND FIN son 1 FAIRE
      ATTENDRE T1
      JOUER son 2 SUR sortie audio VOLUME 10

  QUAND FIN son 2 FAIRE
      question utilisateur
      SI réponse "continuer"
     ALORS
        AFFICHER texte 1 SUR fenêtre 1 SOUS image 1
        AVEC
        POUR i DE 1 A n AFFICHER image i SUR fenêtre 2 DESSUS image i-1
        AVEC
        JOUER son 3 SUR sortie audio VOLUME 20

        QUAND fin texte 1 FAIRE
            ATTENDRE T2
            AFFICHER graphique 1 SUR fenêtre 1 A DROITE texte 1

        QUAND image N0 X FAIRE
          AFFICHER texte 2 SUR fenêtre 2 DESSOUS image X
FIN
```

FIGURE 16 *: scénario exprimé par un script*

2. IMPLANTATION DU SCRIPT

2.1. Langage intégré

Le scénario de l'auteur est traduit en un langage de programmation qui est intégré dans l'application. Les langages intégrés sont des langages d'extension conçus pour ajouter de nouvelles spécifications algorithmiques à une application. Betz, [BETZ 88] les définit comme des environnements miniatures de programmation incorporés dans une application qui permettent à l'utilisateur de configurer et d'étendre les possibilités de son application. Ils sont donc utilisés dans des applications qui ont besoin de plus de flexibilité que ne leur en fournit l'approche traditionnelle. Des exemples typiques de langages d'extension sont

les langages de macros qui sont utilisés dans de nombreux types d'application telles que les tableurs, les traitements de textes ou les bases de données [GATE 87]. Les macros sont intéressantes pour ajouter des fonctions à une application mais ont l'inconvénient d'être non standardisées et ainsi trop dépendantes de leur application (tous les langages de macros sont différents) et de ne pas avoir la puissance, c'est à dire l'efficacité, et la souplesse des langages traditionnels (qui sont des langages universels au sens de Turing). Un autre exemple plus élaboré de langage intégré est celui que l'on trouve dans un produit comme Hypertalk, le langage d'Hypercard qui permet de générer des applications spécifiques.

Dans notre cas, le langage encapsulé a pour but de représenter l'architecture du message et de configurer l'environnement utilisateur. Puisque le script est envoyé avec le message, et doit pouvoir s'exécuter sur différents matériels, il doit être transmis sous une forme symbolique. Il peut donc être, soit interprété par le destinataire, soit compilé puis exécuté. Cependant, il doit respecter la sémantique des scripts des messages. La meilleure façon de garantir celle-ci est d'incorporer l'interprète (ou le compilateur) de scripts à l'application concernée. Il est inclus dans le système qui contrôle l'ensemble.

2.2. Choix du langage

2.2.1. Caractéristiques nécessaires

Le langage implantant les scripts des messages doit répondre à un certain nombre de caractéristiques propres aux langages d'extension. Il doit être facile à apprendre avec une syntaxe simple, être souple et puissant plutôt qu'efficace [CFI 91a]. Il doit pouvoir déclencher les fonctions propres à l'application et comporter les éléments traditionnels des langages de programmation. Deux choix sont possibles : un langage interprété ou un langage compilé. Le premier est plus facile à utiliser et permet d'avoir une solution dynamique, le second offre de meilleures performances. Dans notre application, les scripts seront créés interactivement sous l'application ou directement exécutés à partir de messages les contenant. Ceci conduit à une approche de type interprété plutôt que compilé, avec l'interprète inclus dans l'application. Le choix d'un interprète est justifié par le fait que l'on veut pouvoir modifier interactivement le script du message, exécuter des parties du script dans un ordre différent ou indépendamment. Il est possible de modifier l'environnement de l'utilisateur en cours de restitution de messages ou d'ajouter de nouvelles fonctions pour traiter certains messages. Un compilateur offre une approche trop rigide pour ce type de spécifications qui nécessite que les scripts soient exécutés dynamiquement et séparément du reste de l'application.

2.2.2. Langages possibles

Nous avons choisi un langage de programmation "universel" car nous avons besoin de disposer de structures de contrôle et d'accéder aux ressources du système. Plutôt que de créer un autre langage, nous avons choisi un langage existant. En effet, parmi les langages présents dans le monde informatique, plusieurs répondent à nos besoins ; il n'est donc pas nécessaire d'en concevoir un autre. De plus, choisir un langage existant permet d'avoir un

langage robuste et évite aux utilisateurs qui le connaissent un nouvel apprentissage. Analysons les différents langages interprétés qui correspondent le mieux à nos besoins :

❒ *APL* est un langage qui allie puissance et concision mais il nécessite un interpréteur conséquent. La syntaxe est très simple mais demande un ensemble de caractères spécifiques.

❒ *Forth* est un langage puissant mais manque de structure de données et est peu connu de la communauté informatique.

❒ *Hypertalk* (le langage utilisé dans Hypercard d'Apple), fournit des possibilités intéressantes pour des systèmes multimédias et hypermédias mais sa syntaxe n'est pas suffisamment formelle pour que des programmes puissent être générés automatiquement.

❒ *Postscript* [Post] est un puissant langage de description de page mais il ne fournit pas les caractéristiques de temps réel nécessaires. De plus, il est complexe à implémenter et fournit des possiblités qui sont trop disjointes de celles nécessaires à l'utilisateur d'un système de messagerie multimédia.

2.2.3. Choix de Lisp

Les langages cités ci-dessus n'étant pas adaptés à nos besoins, nous avons choisi un langage d'extension basé sur Lisp. Parmi l'ensemble des langages possibles, Lisp répond le mieux aux exigences d'un langage d'extension [CFI 91b] car :

- il est facile à intégrer dans une application,
- il garantit la portabilité s'il est intégré dans le système,
- c'est un bon langage procédural,
- c'est un langage connu par la plupart des personnes travaillant dans les domaines de l'informatique,
- il est extensible et raisonnablement efficace.
- il offre une manière souple et puissante de description d'actions.

De plus, les systèmes Lisp ont prouvé leur efficacité pour le prototypage d'environnement et sont souvent utilisés comme plates-formes pour la construction de systèmes de gestion d'interface utilisateurs (UIMS) [BECK 91]. Enfin, Lisp est un langage interprété, standard et reconnu.

Le système Lisp choisi (GLisp) fait partie du système InTalk qui a été spécialement conçu pour s'intégrer avec des programmes C. Le système InTalk [GIRA 92] est un outil permettant le développement d'applications paramétrables et extensibles grâce à l'introduction d'un véritable langage de commande interne à l'application permettant de faire fonctionner celle-ci en mode interactif, de créer des fichiers de commandes, etc. Il permet donc d'inclure un langage script dans une application développée dans un langage compilé tel que C. Ce type d'architecture hybride (cf FIGURE 17) permet d'allier la puissance du premier et la souplesse du second. Le langage compilé est utilisé pour des parties qui ont besoin d'être exécutées rapidement ou répétitivement. Il permet aussi

d'accéder à des algorithmes spécifiques ou à des ressources du système. Le langage script est utilisé pour configurer l'application ou l'environnement et ajouter de nouvelles fonctions. Celui-ci reste sous le contrôle de l'application. De plus, l'implantation des scripts en un langage Lisp fait qu'ils sont entièrement compatibles. En effet, toute machine ayant l'interprète GLisp peut exécuter le script tout en respectant sa sémantique. Enfin, GLisp est plus performant (d'un facteur de 2,5 à 40) que d'autres Lisp du domaine public (XLISP, MIT_Scheme, SCM, PSI, etc).

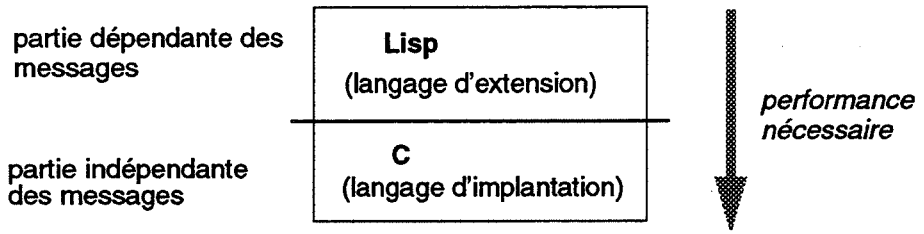


FIGURE 17 : *architecture hybride*

3. CONCLUSION

Nous avons vu que les messages multimédias ne peuvent pas être assimilés à des documents multimédias statiques car ils sont assujettis à des contraintes spatiales mais aussi temporelles. De plus, des interactions et des actions (prédéfinies ou aléatoires) sont possibles.

Reprenons l'exemple du message envoyé aux participants d'un congrès au Japon. Ce message comporte toutes les informations nécessaires sous forme d'images, de graphiques et de voix. Ces divers renseignements s'organisent les uns avec les autres pour former un ensemble cohérent. Le message commence par une page d'informations générales et le planning du congrès. Viennent ensuite des plans pour s'orienter, des photographies des sites à visiter, des phrases de prononciation, etc. Le destinataire peut ensuite remplir les formulaires d'inscription qui sont alors automatiquement envoyés (par sélection d'un menu) aux organisateurs.

Prenons un autre exemple, futuriste, d'une publicité envoyée par messagerie électronique. Sous un titre alléchant (pour ne pas être détruit sans être lu), le message a pour but de présenter un nouveau produit. Dès que le destinataire a sélectionné le message, une image couleur d'un visage apparait annonçant verbalement le nouveau produit. Suivent ensuite les caractéristiques et les qualités du produit avec images et graphiques à l'appui. Des commentaires sonores accompagnent le défilement des informations. Une petite séquence animée montre divers cas d'utilisation du produit. L'utilisateur peut demander automatiquement l'envoi d'informations supplémentaires en choisissant dans un menu les spécificités qui l'intéressent. Le message se termine par un bon de commande pré-établi (adresse de l'expéditeur, références) avec la promesse d'une réduction sur le prix si la commande arrive sous les 10 jours.

Pour formaliser de tels messages, nous avons besoin d'une grande souplesse de description, d'automatismes, de mécanismes d'historique, d'interactions avec l'utilisateur, et d'actions pré-établies. Ceci est rendu possible en formalisant les spécificités des messages par des scénarios. Ceux-ci sont représentés sous forme de scripts accompagnant les composantes du message. Le message apparait ainsi comme une entité unique, un *messager* qui véhicule les informations et comportements nécessaires à la présentation interactive des informations du message. Le modèle proposé permet donc de représenter les caractéristiques de ce type de message en offrant la souplesse et la puissance requises.

Pour illustrer cette approche, nous avons développé un outil de messagerie multimédia. Un des objectifs principaux du projet est de prendre en compte l'architecture spécifique des messages multimédias au moyen d'un scénario qui coordonne la présentation du message. Il est représenté par un script traduit dans un langage de programmation Lisp. La partie suivante présente la réalisation de cette application.

UNE RÉALISATION : 4M III

6. Présentation générale
7. Modules fonctionnels
8. Interface utilisateur
9. Scripts 4M

CHAPITRE 6

Présentation générale

Nous venons de définir un modèle de communication basé sur des messagers et nous avons tenté de montrer son adéquation aux problèmes de représentation des messages multimédias. Une application de messagerie multimédia a été développée pour illustrer cette approche. Ce chapitre présente les concepts mis en oeuvre lors de la conception de cette application : les objectifs du projet, les contraintes à respecter et les choix d'implantation effectués. Nous verrons dans les chapitres suivants les détails de la réalisation.

1. PRÉSENTATION

1.1. Le projet

Le projet intitulé 4M (MultiMedia Mail Manager) consiste en la conception et en la réalisation d'un UA multimédia. Le terme UA (User Agent) est emprunté aux Recommandations du CCITT sur la norme X400. L'agent utilisateur d'une messagerie est un processus d'application pour la préparation, l'envoi et la réception d'un message. Il peut permettre aussi de trier ou stocker des messages dans des répertoires. L'UA correspond à la partie accessible d'un système de messagerie par l'utilisateur final. Il sert d'interface entre l'usager et le système de transfert de messages qu'il rend transparent à l'utilisateur. Parmi les agents utilisateurs les plus utilisés, on peut citer *Mail* et *elm*.

L'UA multimédia proposé doit permettre à un utilisateur de préparer, émettre et recevoir des messages comportant d'autres types de médias en particulier des images et des sons, et de les restituer sur une machine différente. Ce projet est basé sur le modèle théorique que nous avons vu précédemment dont il reprend les concepts généraux. Les messages manipulés par l'outil sont des messages multimédias interactifs dont la

présentation spatio-temporelle et le comportement sont formalisés par un script. Ce script est traduit dans un dialecte de Lisp.

Outre la validation du modèle proposé, cette réalisation doit servir d'illustration des besoins de l'utilisateur vis à vis d'une messagerie multimédia, et en particulier prendre en compte les points suivants :

- structure des messages (et plus précisément leurs aspects spatiaux et temporels),
- intégration du système dans l'environnement de l'utilisateur,
- interface utilisateur (comment les messages sont créés et affichés).

1.2. Caractéristiques

Le projet 4M se caractérise par la simplicité d'utilisation et la légèreté de l'implantation.

Un message multimédia ne doit pas être considéré comme un document multimédia. Le document multimédia est souvent un document complexe comprenant de nombreux types de médias et une structuration particulière. Il est souvent composé de plusieurs pages incluant des images et des graphiques, des commentaires sonores, éventuellement des références à d'autres documents, des liens hypermédias, un glossaire, etc. Il demande donc une longue préparation. Au contraire, le message est par définition une facilité de communication. Il comporte en général moins d'informations ; les messages sont très souvent des notes brèves, des explications ou des réponses, dans lesquelles l'expéditeur veut simplement inclure une image ou un passage sonore. Il doit donc pouvoir être composé facilement et rapidement et permettre d'inclure aisément des informations visuelles et auditives. Nous voulons donc offrir aux utilisateurs de machines diverses la possibilité d'émettre et de recevoir des messages multimédias complexes de façon aussi simple et transparente que possible.

Nous avons vu qu'il existe un petit nombre de systèmes de messagerie expérimentaux et commerciaux. Tous sont des produits finis et complets. Ils permettent à leurs utilisateurs d'émettre et de recevoir des messages multimédias plus ou moins complexes. Mais ces systèmes sont lourds et nécessitent un investissement de la part de l'utilisateur. De plus ils ne sont pas compatibles entre eux. Notre objectif n'est pas de réaliser un nouveau système mais plutôt de créer un prototype léger qui puisse fonctionner sur des machines hétérogènes sans nécessiter d'investissement important de la part des utilisateurs.

1.3. Objectifs

Le projet 4M est conçu pour illustrer l'approche de notre modèle tout en respectant les objectifs fixés.

• Il doit permettre la création d'un message multimédia à un coût (en temps et manipulation) proche de celui d'un message uniquement textuel. Pour cela, on privilégie l'aspect graphique de l'interface avec l'utilisateur et l'emploi maximum des outils propres à l'utilisateur pour composer son message.

• Le système doit formaliser la représentation d'un message multimédia en prenant en compte la structure du message et les interactions de l'utilisateur. L'implantation est réalisée par un script traduit en un programme Lisp et transmis avec le contenu du message.

• Le système doit être souple et extensible, l'utilisateur pouvant ajouter de nouvelles fonctions soit dans son message soit dans son propre système pour modifier son environnement.

• Ce projet doit fonctionner dans un environnement hétérogène pour permettre à de nombreux utilisateurs d'échanger des messages.

2. DESCRIPTION

2.1. Principe de base

Le projet 4M se présente comme un ensemble d'outils permettant de créer et de recevoir des messages multimédias ayant les caractéristiques que nous avons vues précédemment. Puisque nous nous intéressons principalement à la structure des messages multimédias et à leur réception, nous ne construirons ni d'éditeurs spécifiques ni d'outil multimédia intégré pour créer les différentes parties des messages. On peut citer plusieurs raisons pour ne pas fournir un éditeur de messages multimédias. La première raison est que les personnes emploient des outils différents selon leur besoins. Par exemple, certaines personnes utilisent simplement "*Mail*" ou un équivalent pour taper leur message et l'envoyer, d'autres le préparent avec leur éditeur de texte préféré (*vi*, *emacs*, *edit*) ou un outil graphique (*mailtool* de OpenWindows). Pour des types de médias plus complexes, comme des textes formattés ou des images, il existe de très nombreux outils spécialisés. Chacun est plus performant dans son domaine qu'un outil intégré. De plus, les utilisateurs sont en général très réticents à l'idée de changer d'environnement de travail. Une approche plus adaptée est de leur permettre d'employer leurs propres outils dans leur propre environnement pour créer les messages. Nous voulons également que le système reste ouvert à de nouveaux types de médias ou de nouveaux formats. L'utilisation d'outils externes plutôt qu'un système intégré facilite leur inclusion. Cependant, le système ne permettra de reconnaître que certains types de formats courants car nous ne nous occuperons pas des conversions inter-formats.

Les différentes parties du message sont donc créées de façon externe. La présentation et le comportement du message sont formalisés par un script. Il est transmis avec les différentes parties du message dans un message unique. Pour permettre une création simplifiée des scripts, on fournit un langage de commande construit sur le langage d'extension. Ce langage de commande est exprimé de manière textuelle ou gestuelle. La première forme d'expression est fournie par des fonctions spécialisées prédéfinies respectant la sémantique Lisp ; un outil graphique de conception automatique de script corrrespond à la forme gestuelle du langage de commande.

Les différentes parties sont alors assemblées pour constituer une entité unique, que l'on peut qualifier d'*agent actif*, d'*acteur*, ou ici, de *messager*. Le message est donc représenté

par un messager qui contient les différents comportements nécessaires à la présentation interactive des informations du message. Ces comportements sont prédéfinis (composition de texte, affichage d'images, restitution de sons, assemblage, synchronisation, interaction avec le client), ou construits spécifiquement pour ce message. La restitution du message multimédia est réalisée par l'exécution du script sur le site récepteur. Le messager va créer un ensemble d'acteurs coopérants chargés de restituer le message. Chaque tâche transmise par le messager entraîne la création d'un acteur ou l'envoi d'un message à un acteur spécialisé auquel est confiée la tâche en question. Chaque acteur est libre de créer d'autres messagers pour venir à bout de sa mission. L'ensemble des tâches est géré par un gestionnaire de tâches qui s'occupe de la création, de l'exécution séquentielle ou simultanée et de la destruction des acteurs réalisant les tâches nécessaires.

2.2. Fonctionnement

La figure (cf FIGURE 18) illustre le fonctionnement général de l'application. Le processus est le suivant :

1. création des parties de message et du script sur la machine expéditrice stockés sous forme de fichiers,
2. assemblage des fichiers et informations d'adressage pour former un message transmissible,
3. transmission du message,
4. réception sur la machine réceptrice, démultiplexage en parties distinctes dans des fichiers,
5. exécution du script pour donner le message multimédia.

Une fois le message reçu, le destinataire peut dérouler son message dans un ordre différent ou modifier certains paramètres de restitution du message. Il doit aussi être capable d'interrompre des messages longs à n'importe quel moment et permettre de se protéger contre des messages potentiellement dangereux dans le cas de messages actifs qui exécutent des programmes sur la machine réceptrice.

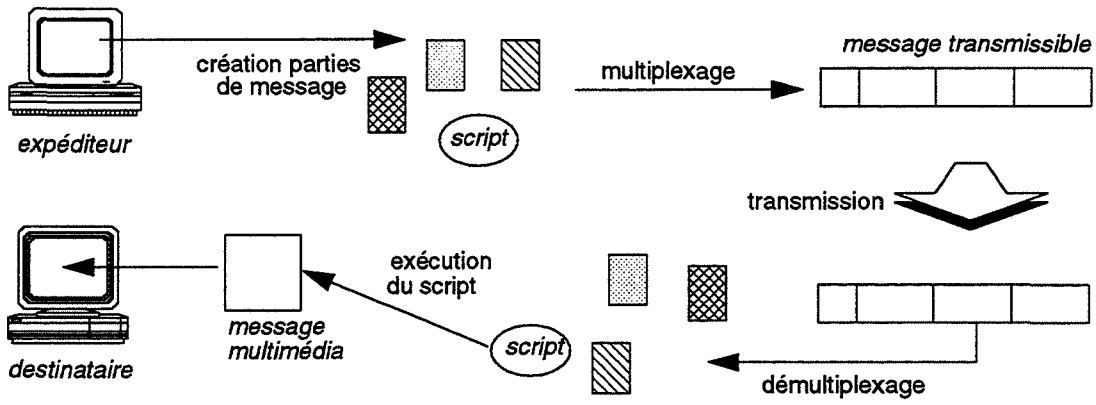


FIGURE 18 : *fonctionnement général*

2.3. Choix effectués

2.3.1. Logiciels

Le projet étant la réalisation d'un UA multimédia, nous ne nous intéresserons pas au transfert des messages qui sera réalisé par le logiciel local de transport. Une fois le message composé, il sera transmis au module responsable du transfert du message de façon transparente à l'utilisateur. De même, on ne s'occupera pas de la façon dont les messages arrivent. Un transcodage (pour obtenir une forme transmissible ou une forme affichable) peut être nécessaire mais il est effectué de chaque côté de façon transparente et n'affecte pas le reste du fonctionnement.

Le système n'offrirera pas non plus des fonctionnalités élaborées de gestion ou d'archivage de messages qui ne font pas partie des objectifs fixés. Seules quelques options de base classiques seront envisagées. Les fonctions principales de l'interface seront la composition et la restitution des messages.

2.3.2. Techniques

Le projet 4M est conçu pour fonctionner dans un environnement hétérogène. On connaît la multitude de machines et systèmes existants, souvent incompatibles entre eux, aussi nous sommes-nous fixés un certain nombre de contraintes pour notre prototype, qui reflètent en grande partie des choix propres à notre laboratoire.

La première est l'utilisation d'Unix comme système d'exploitation. Déjà largement utilisé dans le monde des stations de travail et il est probable qu'il sera dans le futur disponible sur presque tous les ordinateurs. On peut déjà trouver chez Apple, Atari et IBM des produits comme respectivement *AUX*, *ATX* et *AIX*.

La seconde contrainte se situe au niveau de l'interface avec l'utilisateur. Nous avons choisi comme support le système de fenêtrage *X Window*. Ce choix a été guidé essentiellement par le fait que X est devenu un standard commercial sur lequel un grand nombre de constructeurs ont investi et sur lequel on trouve la plupart des styles d'interface. Il est disponible sur la plupart des systèmes Unix et aussi sur de nombreux ordinateurs individuels. La plupart des sociétés fournissent du matériel spécialement conçu pour supporter le protocole X. De plus, X11 est construit sur un protocole conçu pour être indépendant des périphériques. Bien que conçu sur Unix, X peut être implanté sur n'importe quel système d'exploitation.

La construction de ce prototype nous permet de valider nos idées tout en imposant au système des contraintes relativement faibles. Cependant, les programmes ont été écrits afin d'encapsuler autant que possible ces spécificités (Unix et X11) dans l'éventualité d'un portage sur d'autres systèmes (MacOS, Windows, etc).

2.4. Architecture du système

Le MultiMedia Mail Manager implante un système de messagers composé de quatre modules principaux : le module qui réalise le dialogue avec l'utilisateur (UIM), le module de composition des différentes parties du message (MCM), le module de traitement des messages 4M (MPM) et le module de production des messages (MRM). Ces modules forment l'architecture générale du système (cf FIGURE 19).

- Le module UIM (User Interface Module) est l'interface graphique qui réalise le dialogue entre l'utilisateur et les fonctionnalités du système. Ce module est relié aux modules de composition et de restitution auxquels il fournit l'interface graphique.

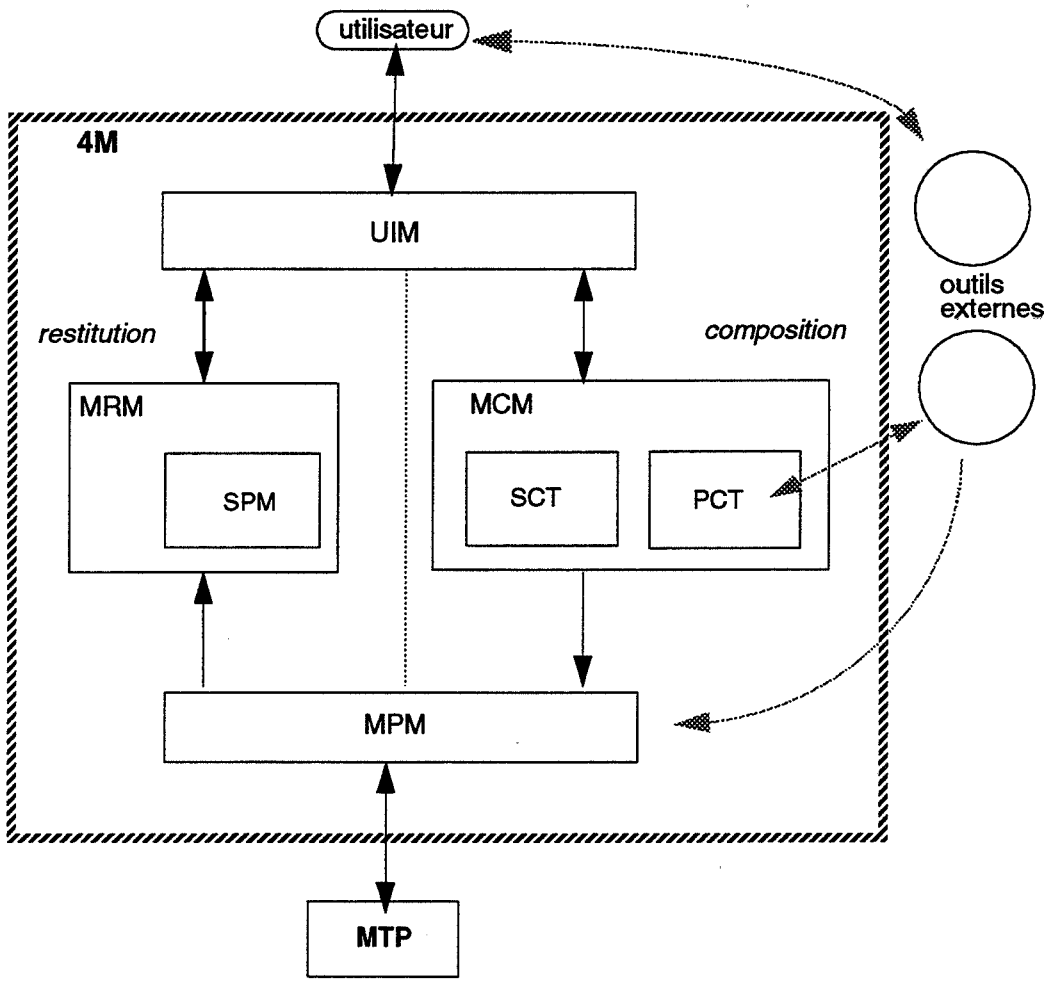
- Le module MCM (Mail Composition Module) est utilisé lors de la composition des messages. Il comporte deux outils, l'un pour la création du script (SCT), l'autre pour récupérer les objets multimédias créés avec des outils externes (PCT).

- Le module MRM (Mail Rendering Module) correspond à la restitution des messages. Il comporte le sous-module d'exécution des scripts des messages 4M (SPM).

- Le module MPM (Mail Production Module) a pour rôle de fabriquer le message sous forme transmissible à partir des composants fournis par le module PCM. Il est aussi responsable de la fonction inverse, c'est à dire du découpage du message reçu en différentes parties pour la restitution. Il communique avec le système responsable du transfert des messages (MTP) auquel il transmet le message à envoyer. Ce programme achemine les messages jusqu'aux destinataires et fournit au MPM les messages qui arrivent.

Les modules PCM et SPM sont indépendants car ils sont utilisés soit en composition soit en réception de messages. Les communications entre les modules MRM ⇔ MPM et MCM ⇔ MPM, sont réalisées au moyen de fichiers qui correspondent aux divers composants du message. L'utilisateur peut employer des outils externes pour créer les différentes parties du message ; ces composants sont fournis sous forme de fichiers, soit au sous-module PCT qui permet de les visualiser avant leur inclusion dans le message, soit directement au module MPM pour la conception du message transmissible.

Nous allons reprendre en détail les caractéristiques et le fonctionnement de chacun des modules dans les chapitres suivants.



4M : MultiMedia Mail Manager
MPM : Mail Processing Module
MTP : Mail Transfert Program
MCM : Mail Composition Module
PCT : Part Composition Tool
SCT : Script Composition Tool
MRM : Mail Rendering Module
SPM : Script Processing Module
UIM : User Interface Module

FIGURE 19 : *architecture générale*

CHAPITRE 7

Modules fonctionnels

L'architecture du projet 4M repose sur divers modules que l'on peut séparer en deux groupes : les modules fonctionnels ainsi appelés car ils réalisent chacun une partie du fonctionnement de l'application et l'interface utilisateur que nous verrons ultérieurement. Ce chapitre présente les divers modules selon leur fonction principale : la composition, la restitution et la transmission de messages. Nous ne détaillerons pas l'utilisation de ces modules du point de vue de l'utilisateur, mais expliciterons plutôt les principes et les caractéristiques du fonctionnement de chacun.

1. MODULE DE COMPOSITION (PCM)

1.1. Principe

Ce module est chargé de la composition des messages multimédias 4M à partir de parties de messages existantes ou à créer. Pour des raisons matérielles et logicielles, nous nous sommes restreints à des types d'objets textes, sons et images fixes. Le rôle du module de composition est essentiellement de rassembler les différents objets et de concevoir le script de l'expéditeur. Ce sont les rôles respectifs des sous-modules PCT et SCT. Ainsi, la composition se déroule en deux phases qui sont la *collecte d'informations*, et *la création du script.*

- le processus de "collecte" consiste à créer ou récupérer des représentations d'informations et à les stocker sous forme de fichiers.

- la seconde étape est la conception du script associé aux différentes parties du message. Comme nous l'avons vu dans la partie précédente, ce script définit le séquencement et les méthodes selon lesquelles les différents types de données sont affichés ou entendus, la synchronisation entre eux et les interactions extérieures. Ce script peut être créé par

l'utilisateur, soit en utilisant un éditeur de texte tel que *vi* ou celui fourni par le module PCT, soit en utilisant l'outil graphique SCT.

Une fois les différentes parties de message créées, l'expéditeur spécifie, dans les champs prévus à cet effet, l'adresse de son destinataire, éventuellement le sujet du message et la liste des fichiers contenant le script et les parties composant le message (cf FIGURE 20). L'ensemble des informations est fourni ensuite au module MPM qui s'occupe de créer le message transmissible. Si, lors de l'envoi du message, aucun script n'est spécifié, l'application crée un script par défaut (simplifié) que l'utilisateur peut inclure dans son message.

1.2. Sous module PCT

Il permet la consultation des différents objets multimédias que l'expéditeur veut inclure dans son message. Il est séparé en trois outils d'aide distincts, chacun manipulant un type de média. Puisque l'on privilégie l'emploi de programmes externes appropriés à l'environnement de l'utilisateur, ces divers outils ne sont pas très sophistiqués. Ils servent uniquement d'aide à la composition du message.

- Le premier est un éditeur de texte simplifié (style *xedit*) pour charger, créer ou modifier un fichier texte. Il est surtout utilisé pour envoyer des messages textes ou pour concevoir un script. Cet outil permet de saisir ou de visualiser un texte sans sortir de l'application. Considéré comme l'outil par défaut, il est appelé automatiquement à toute composition de messages (cf FIGURE 20).

- Le deuxième outil est un outil de visualisation d'images sous divers formats (bitmaps X11, rasterfile Sun, macpaint). L'expéditeur peut ainsi choisir la ou les images qu'il souhaite envoyer. Il peut aussi sélectionner une partie d'une image et l'enregistrer dans un nouveau fichier (cf FIGURE 21).

- Le troisième outil permet d'écouter des sons et de la voix pré-enregistrés sous divers formats (par exemple, Sun, Mac). Il est aussi possible, si la machine dispose de périphériques audio d'entrée/sortie, d'enregistrer de nouveaux sons (cf FIGURE 22).

Ces deux derniers outils reconnaissent automatiquement le format, respectivement, des images ou des sons fournis en entrée. Ils sont accessibles directement à partir du module principal de composition

Les trois parties de ce module manipulent des objets représentés sous forme de fichiers selon certains formats. Le problème de passage d'un format à un autre est extrêmement courant, et de nombreux convertisseurs ou formats existent déjà. Aussi, nous avons préféré employer des formats de représentation simples et très courants plutôt que de définir des formats propres au Multimedia Mail Manager. Pour chaque type de média, on utilise un principe de format pivot qui permet de limiter le nombre de convertisseurs entre formats. De plus, cette technique permet de rester ouvert à d'autres formats existants par simple adjonction d'un convertisseur dans le format approprié. Les formats pivots utilisés sont la représentation ASCII pour le texte, la structure d'image X11 et le format son SUN. Divers convertisseurs de formats d'images et de sons internes à l'application ou externes permettent d'obtenir une des représentations compréhensibles par l'UA. Il est envisagé

d'utiliser d'autres formats pivots plus riches et mieux adaptés ; ODA pour le texte, JPEG et PBM pour les images en sont des exemples.

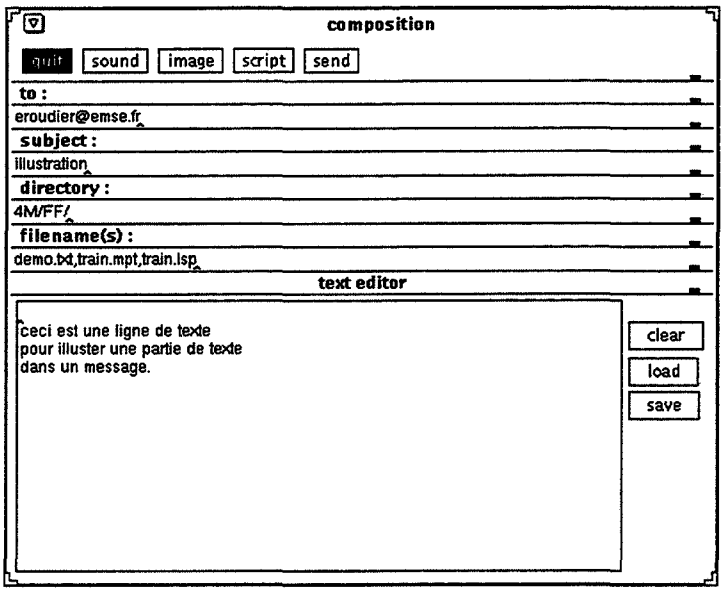


FIGURE 20 : *composition de messages - éditeur de textes*

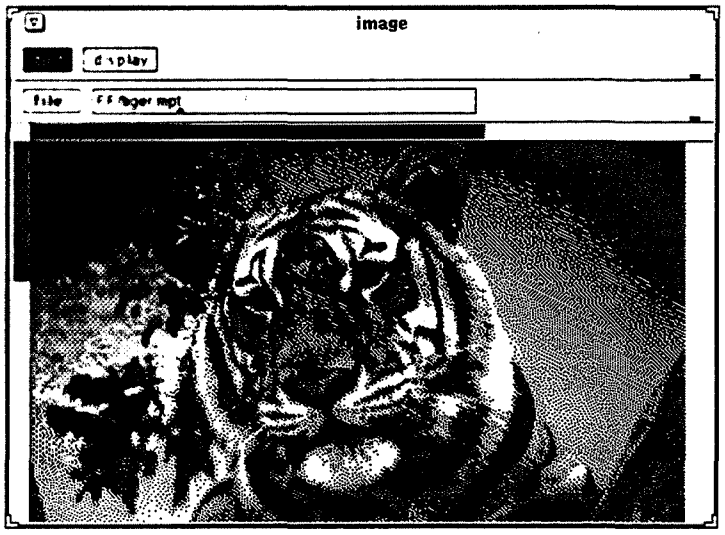


FIGURE 21 : *outil de visualisation d'images*

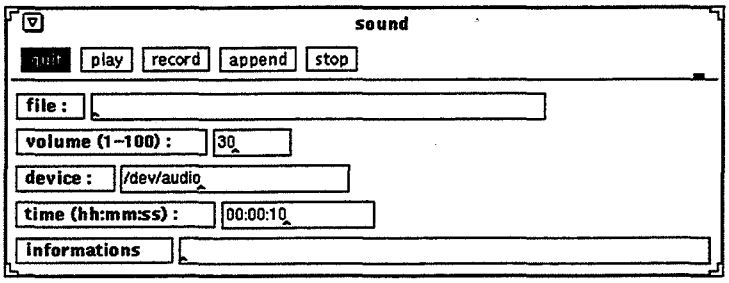


FIGURE 22 : *outil de production de sons*

1.3. Sous module SCT

Outre la sélection des objets du message, l'utilisateur doit composer le script associé. Pour cela, il a à sa disposition un certain nombre de fonctions que nous analyserons plus en détail dans le chapitre suivant. Pour simplifier sa création, en particulier pour les utilisateurs inexpérimentés ou ne connaissant pas la syntaxe, ce sous module propose un outil graphique de composition de script (cf FIGURE 23).

La conception se déroule en trois phases. Tout d'abord, l'utilisateur définit les objets de son message (fenêtres, textes, sons, boutons) qui sont représentés sous forme iconique. Il détermine leurs éventuels attributs : taille des fenêtres, nom du fichier contenant l'objet multimédia, action associée à un bouton, etc. Ces objets sont ensuite disponibles pour concevoir l'architecture spatio-temporelle du message (principe de tableau de bord). Les relations spatiales sont obtenues en disposant les objets dans les différentes fenêtres prédéfinies. Le déroulement du message est spécifié en les positionnant selon une échelle de temps. Si les spécifications spatiales des objets sont définies, alors le déroulement du message est créé automatiquement de manière séquentielle. L'ordre de composition n'est pas strict ; à chaque étape, l'utilisateur peut créer, détruire ou déplacer les icones dans le tableau de bord ou dans les fenêtres de spécification d'architecture. Si une icone représentant un objet est détruite, toute spécification spatiale ou temporelle relative à cet objet est aussi automatiquement supprimée.

Le script est généré automatiquement à partir des objets définis et des positions spatiales et temporelles spécifiées. L'utilisateur peut le visualiser ou le modifier puis le sauvegarder.

Certaines fonctions de script plus difficiles à spécifier graphiquement ne sont pas encore accessibles par le module de composition de script. Ce sont principalement des fonctions agissant de façon dynamique sur le message : déplacement de parties de messages, effacement de fenêtres, etc. Elles peuvent être ajoutées dans le script par l'utilisateur au moment de la sauvegarde.

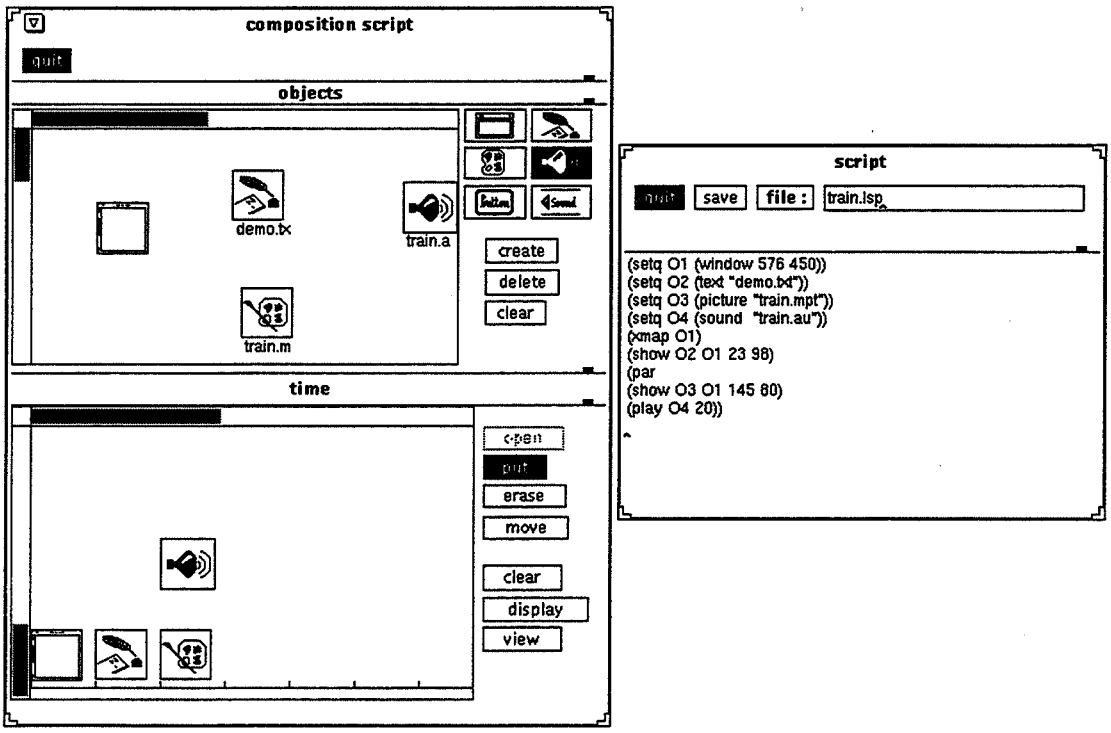


FIGURE 23 : *sous-module de composition de script*

2. MODULE DE RESTITUTION (MRM)

2.1. Principe

Ce module est responsable de la restitution des messages. Comme le module de composition, il manipule des fichiers qui proviennent du module MPM. Ces fichiers correspondent aux parties du message, ils sont temporaires et créés à chaque restitution. Il est possible d'autoriser l'utilisateur de les stocker de manière définitive pour leur réutilisation. Pour éviter à l'usager de changer d'UA, l'application permet de consulter des messages textes tels que ceux gérés par les outils *mail* ou *elm*. Lors de la restitution, on distingue les messages de type 4M (multimédias) et ces messages textes, grâce à des en-têtes spécifiques (cf page 97). Chaque type de message entraîne un traitement différent.

2.2. Restitution

2.2.1. Messages textes

La restitution des messages de type non 4M est réalisée par simple affichage dans une fenêtre de type texte avec ascenseurs, de l'en-tête et du corps du message (cf FIGURE 24).

Le destinataire peut consulter et sauvegarder le message ou le réutiliser pour composer un autre message.

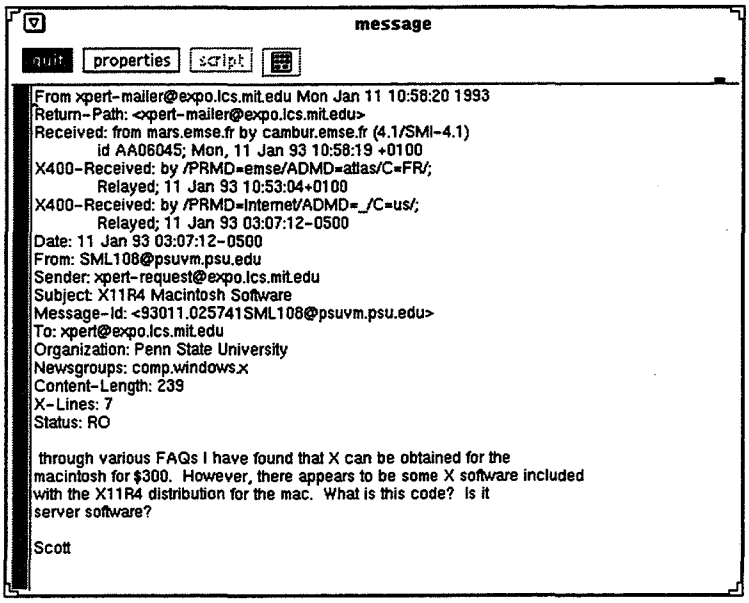


FIGURE 24 : *message texte*

2.2.2. Messages 4M

Ils sont restitués de la façon suivante :

- L'en-tête du message est affichée indépendamment dans une fenêtre de type texte identique à celle des messages textes.

- Le corps du message est restitué par exécution du script contenu dans le message. Le fichier correspondant est fourni en entrée à l'interpréteur GLisp et le message se déroule dans l'ordre spécifié dans le script[1]. Les fenêtres spécifiées s'ouvrent pour afficher les textes et les images. Si le message est simplement sonore, aucune fenêtre autre que celle de l'en-tête n'est alors présente. Le son est produit sur le périphérique de sortie. Si celui-ci n'existe pas sur la machine réceptrice, un message avertit le destinataire que la partie sonore se trouvant à cet endroit dans le message ne peut être entendue (cf FIGURE 25).

Les messages actifs qui exécutent des programmes posent évidemment le problème de la sécurité. En effet, incorporer la puissance d'un langage de programmation "universel" dans un message entraîne automatiquement des risques. Dans le cas de messages multimédias dans lesquels on définit des actions, celles-ci peuvent modifier, voire détruire certaines caractéristiques de l'environnement récepteur. Par exemple, un message peut contenir un script entraînant l'effacement de la boîte-aux-lettres ou d'autres fichiers, des

1. L'exécution des scripts est détaillé au chapitre 9.

modifications de droits d'accès, etc. On a vu aussi des fichiers postscripts, envoyés par réseau, qui modifiaient des caractéristiques de l'imprimante jusqu'à la rendre inutilisable dans certains cas. Il existe différents moyens pour sécuriser ce type d'envoi qui répondent à divers niveaux de sécurité. Le premier niveau correspond à la prise de connaissance des actions contenues dans le message avant leur génération. Le libre choix est laissé au destinataire qui peut vérifier ou modifier le message. Le second niveau autorise l'exécution d'actions déterminées par le destinataire. Éventuellement, un accord commun peut être passé entre l'expéditeur et le récepteur sur les actions autorisées. Une dernière solution est de fournir au script différents environnements d'exécution où les actions potentiellement dangereuses peuvent être filtrées par l'utilisateur lui-même..

Pour résoudre le problème de la sécurité de ces messages actifs, nous avons choisi la première solution. On offre ainsi la possibilité au destinataire de visualiser le script contenu dans le message avant son exécution. Il peut prendre connaissance des spécifications du script et décider de le lancer ou non. Dans ce dernier cas, la restitution du message est abandonnée. Il peut aussi le modifier, dans la mesure du possible, par exemple supprimer les fonctions susceptibles d'être dangereuses. Ce contrôle préalable est le comportement par défaut, il peut être annulé par l'utilisateur via un paramètre de restitution ; dans ce cas, il n'y a plus aucun contrôle avant l'exécution du script.

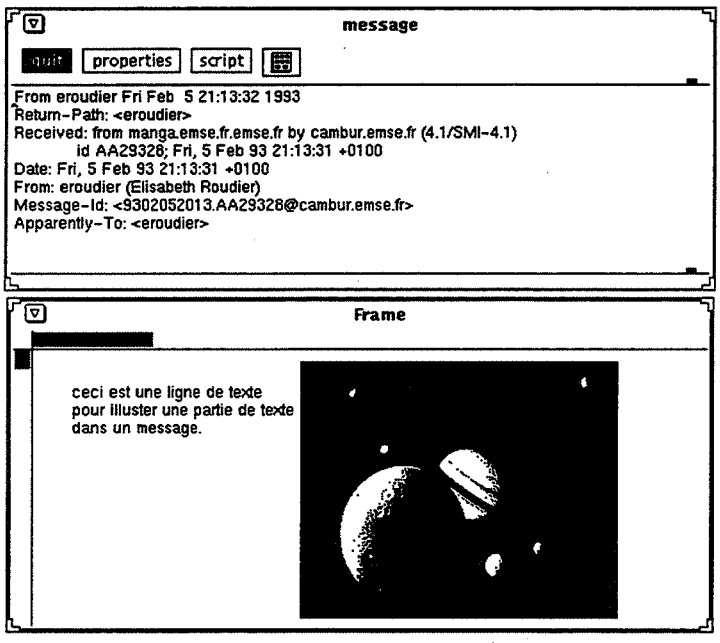


FIGURE 25 : *message 4M*

2.3. Outils complémentaires

L'utilisateur dispose de trois outils complémentaires lui permettant d'agir sur le message en cours de restitution.

• Le premier est un outil de configuration de paramètres tels que la police de caractères utilisée par défaut pour l'affichage des textes, le volume sonore et l'affichage automatique des scripts des messages pour contrôle préalable. Seul le premier paramètre est valable pour les messages textes. Une fois modifiées, les valeurs de ces paramètres seront valables pour tous les messages lus par la suite jusqu'à un nouveau changement. Elles prennent effet immédiatement sur le message en cours qui est ré-exécuté s'il y a modification effective. Si nécessaire, l'utilisateur peut revenir aux valeurs initiales des paramètres (cf FIGURE 26).

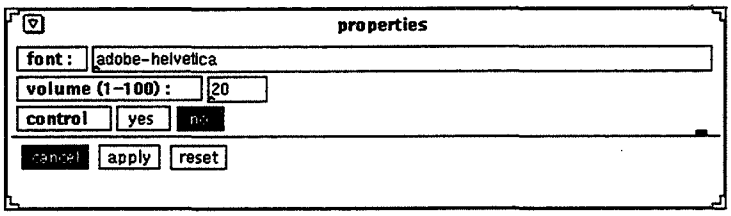


FIGURE 26 : *outil de configuration*

• Le second outil est une télécommande qui agit sur l'enchaînement du message. Grâce aux fonctions disponibles, l'utilisateur peut revenir sur le déroulement de son message, consulter des séquences, réexécuter le script entièrement ou en partie. Les fonctionnalités possibles sont des fonctions classiques de tels outils comme : retour arrière, marche avant, arrêt, pause, continuation, avance rapide, retour au début (cf FIGURE 27).

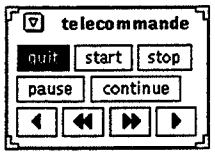


FIGURE 27 : *outil de télécommande*

• Le dernier outil est un éditeur de texte pour visualiser le script du message. Le destinataire peut le modifier selon ses choix et le réexécuter par l'outil de télécommande. Cependant les modifications de ce script ne sont définitives que s'il a été prévu dans le script que le destinataire doit renvoyer le message avec ou sans modification. Sinon, celui-ci garde pour l'instant son script initial, tel que l'a conçu l'expéditeur. Dans tous les cas, l'utilisateur peut, s'il le souhaite, revenir au script originel du message (cf FIGURE 28).

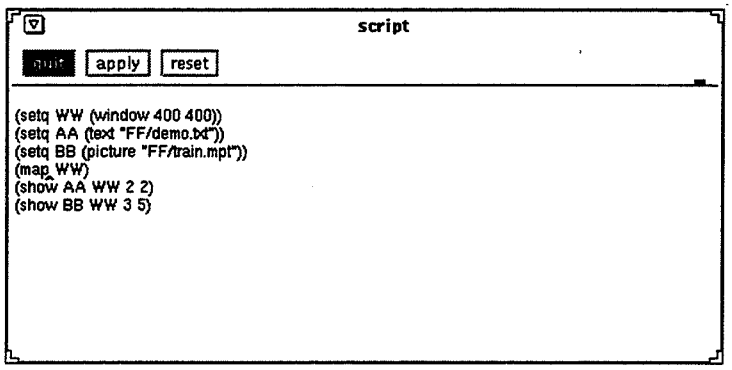


FIGURE 28 : *visualisation du script*

3. MODULE DE TRANSMISSION (MPM)

3.1. Principe

Ce module est responsable de la création du message sous forme transmissible. Les différentes parties du message ainsi que le script sont regroupés dans un message unique qui est traduit en une forme transmissible incluant des informations d'expédition. A la réception, le module agit dans le sens inverse en découpant le message reçu en différents fichiers temporaires.

Si l'expéditeur ne spécifie pas de fichier-script dans la liste des fichiers à transmettre, l'application lui demande s'il souhaite incorporer un script par défaut dans son message. Celui-ci est créé de manière simple et automatique à partir des autres fichiers spécifiées. Si le script existe, et avant d'envoyer son message, l'expéditeur peut prendre connaissance de l'aspect final de son message en l'exécutant. Une fois satisfait de l'aspect de son message, il confirme l'envoi. Le module crée alors le message transmissible correspondant et le passe au système responsable de l'acheminement des messages.

3.2. Format des messages 4M

Un message est composé d'une enveloppe et d'un contenu L'enveloppe contient un ensemble de champs, par exemple les références de l'expéditeur, des destinataires, la priorité du message, etc, dont certains sont utilisés pendant le transfert du message. Le contenu représente l'information que l'expéditeur veut transmettre aux destinataires.

Pour reconnaître les messages textes des messages 4M, une en-tête spécifique indique au début du corps du message que le message en cours est de type 4M. A l'heure actuelle, les messages 4M sont transmis sous forme texte, les parties de messages non textuelles sont compressées et encodées à l'expédition puis décodées et décompressées à la réception. Pour chaque partie de message, une en-tête indique le type de données, le nom du fichier originel et le nombre de caractères. Si la partie de corps de message est une image ou un son, l'en-tête spécifie aussi les méthodes de codage et de compression

employées (cf FIGURE 29). La technique de compression/décompression utilisée permet de véhiculer des messages moins volumineux, d'un rapport de 20 à 50%.

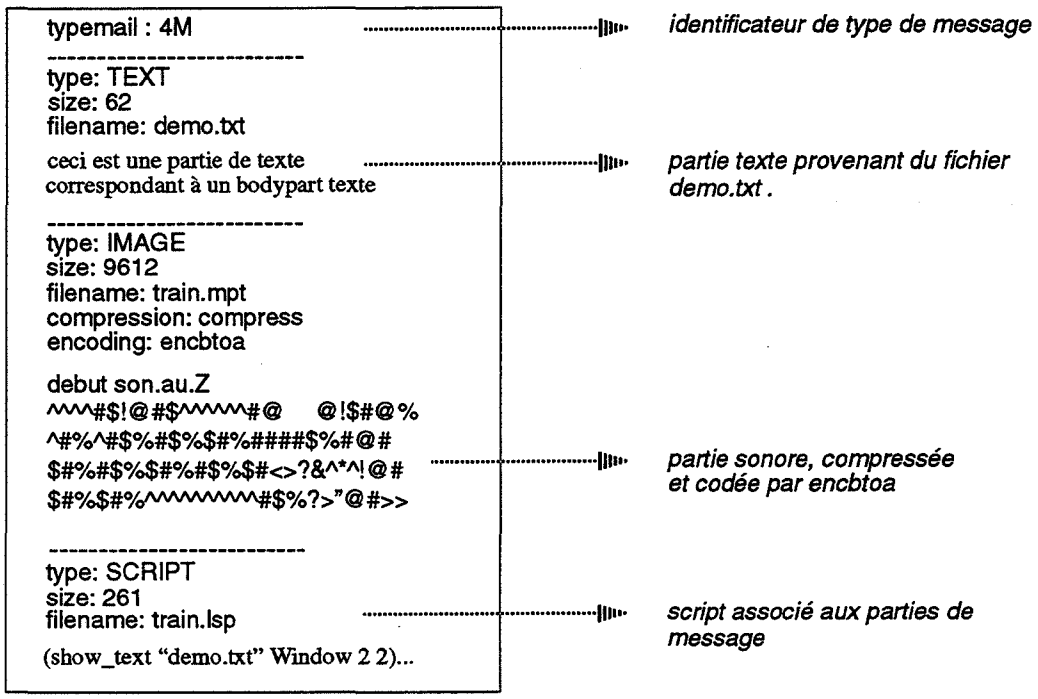


FIGURE 29 : *exemple de message 4M*

3.3. 4M et X400

Les messages 4M utilisent actuellement un format de transmission qui leur est propre mais il est envisageable de se conformer à la norme X400. En effet, cette norme permet la définition de corps de message contenant plusieurs parties (body-parts), chacun pouvant représenter un type différent de contenu : texte, teletext, etc. Un identificateur donne la nature du body-part. Certains identificateurs sont déjà standardisés (Texte IA5, Voix, etc.) d'autres peuvent être déterminés pour des applications spécifiques ou pour un emploi privé.

Pour transmettre des messages complexes composés de données et de scripts tout en respectant les formats X400, nous pouvons définir un identificateur de body-part privé. Cet identificateur est unique et indique que le body-part qui suit a une forme spécifique de type 4M. Le body-part lui-même est privé et dépend de l'application, ce qui permet de garder le format existant des messages 4M. A la suite de cet identificateur, nous pouvons spécifier des informations supplémentaires comme le codage utilisé ou la technique de compression qui peuvent être utilisés de manière générale pour différents médias (cf FIGURE 30). Cette approche permet d'avoir un seul identificateur X400, plutôt qu'un identificateur spécifique par type de body-part reconnu par notre système de messagerie. On évite ainsi des problèmes de cohérence et de gestion d'identificateurs par rapport à

ceux définis dans la norme. De plus, c'est une solution souple et ouverte puisqu'il est possible d'inclure dans n'importe quel message des types de body-parts déjà définis ou d'autres identificateurs privés correspondant à d'autres systèmes de messagerie. On peut avoir ainsi des messages X400 contenant à la fois des body-parts de type 4M et des body-parts de type commun (connus par la norme) ou conformes à d'autres applications.

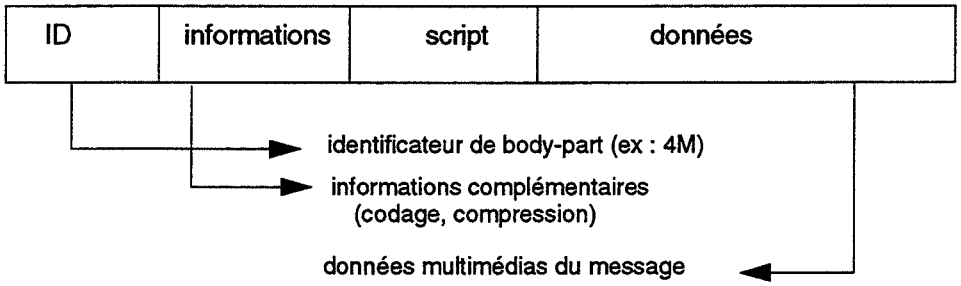


FIGURE 30 *: format X400 d'un message 4M*

4. AUTRES FONCTIONNALITÉS

Ces fonctions ne font pas partie d'un module spécifique. Elles sont fournies à l'utilisateur pour répondre à certains besoins et sont accessibles au niveau supérieur de l'interface (cf FIGURE 31). Ce sont des fonctions de gestion de messages et de dialogue avec l'interpréteur GLisp.

4.1. Gestion des messages

Ces fonctions permettent de gérer les messages de l'utilisateur. Ce sont essentiellement des fonctionnalités d'édition et d'archivage de messages, classiques aux outils de messagerie :

- parcours de la liste des en-têtes de messages et sélection de messages,
- suppression d'un ou plusieurs messages, récupération du dernier message supprimé,
- transfert ou copie d'un ou plusieurs message dans un classeur de messages[1],
- création et mise à jour (suppression effective des messages) de classeur,
- lecture d'un classeur ou de la boîte-aux-lettres courante de l'utilisateur.

1. On appelle classeur, un fichier ordinaire servant d'archivage de messages en les regroupant par thèmes ou selon d'autres critères.

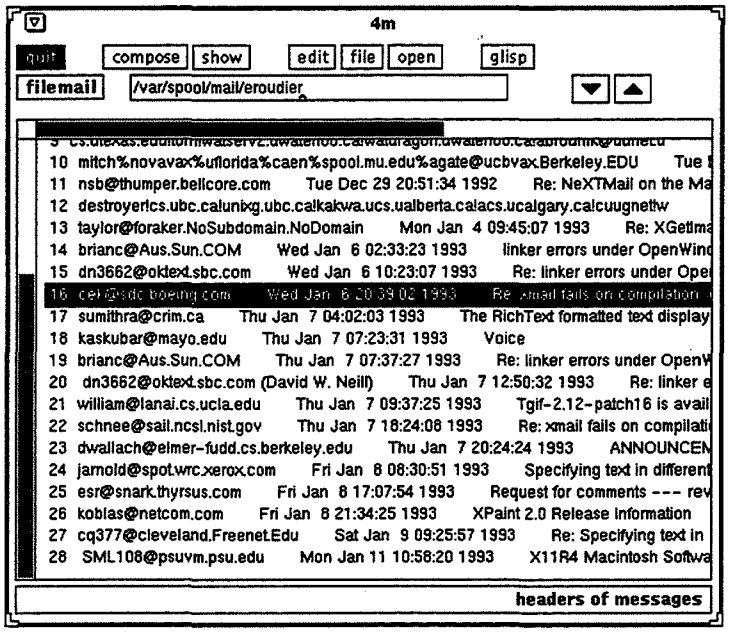


FIGURE 31 *: fenêtre principale*

4.2. Interface avec GLisp

Une fenêtre de dialogue avec l'interpréteur GLisp permet de saisir ou de charger des programmes Lisp qui sont exécutés interactivement par l'interprète. Ces programmes peuvent être, par exemple, des tests de script à envoyer ou toute spécification Lisp. Ils peuvent être exécutés pas à pas, expression lisp par expression Lisp ou comme un tout. Une zone d'entrée avec des fonctionnalités d'éditeur de texte simplifié permet la saisie des expressions. Les résultats ou les messages d'erreurs sont imprimés dans une zone de sortie (cf FIGURE 32).

Cette interface avec le système GLisp permet d'utiliser les fonctions de script définies au chapitre 9 dans un autre contexte que l'envoi de messages. Ceci permet, entre autres, de pouvoir configurer son environnement, son interface, définir de nouvelles fonctions, créer des programmes en exécutant l'ensemble des spécifications par l'interprète.

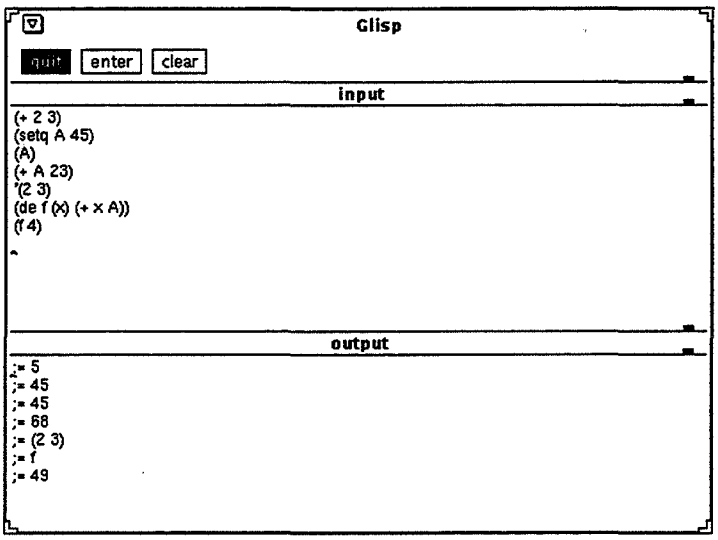


FIGURE 32 : *interface GLisp*

CHAPITRE 8

Interface utilisateur

L'outil 4M est un agent utilisateur de messagerie, et donc par définition, une interface entre l'utilisateur et le système de transfert de messages. L'interface utilisateur a un rôle particulier puisqu'il sert d'intermédiaire entre l'utilisateur et les autres modules de l'application. En effet, toutes les communications entre l'utilisateur et les modules fonctionnels passent au travers de l'interface qui représente une part importante de l'outil. Cette interface est simple, avec des possibilités limitées, sa conception n'étant pas l'objectif principal du projet.

1. PRÉSENTATION

Une interface interactive est la partie logicielle chargée de la communication des informations entre l'utilisateur et un système informatique. L'interface accepte les entrées de l'utilisateur vers l'ordinateur et fournit les sorties de celui-ci vers l'utilisateur.

La qualité de l'interface dépend de ce que l'usager doit connaître pour comprendre ce qu'il voit, et des actions qu'il doit entreprendre pour obtenir les résultats escomptés. L'aspect extérieur d'un logiciel est ce que l'utilisateur perçoit directement du système et qui conditionne son opinion ; le confort d'utilisation et la facilité d'apprentissage déterminent l'acceptation ou le rejet du logiciel [COUT 90]. Ainsi, les produits actuels tendent à toucher une large population d'utilisateurs, novices ou spécialistes, ce qui conduit les développeurs à consacrer une part importante du logiciel à l'interface utilisateur.

L'interface de l'outil 4M, comme nous avons déjà pu le voir, est une interface graphique. Le dialogue entre l'utilisateur et l'UA est réalisé au travers de fenêtres, d'icônes, de menus et autres gadgets fournis par l'interface. Ceci est rendu possible grâce à un environnement multi-fenêtres qui fournit de nombreuses possibilités pour créer de

telles interfaces. En effet, les systèmes de fenêtrage fournissent à l'utilisateur un environnement convivial dans lequel il peut ouvrir des fenêtres, se déplacer à l'intérieur ou appeler des fonctions à l'aide d'objets graphiques. Ils sont d'ailleurs devenus une caractéristique commune de la plupart des systèmes informatiques.

2. CONCEPTION DE L'INTERFACE

La conception et la construction d'une interface sont des tâches complexes et coûteuses La difficulté provient de la nature de l'interface qui requiert des compétences et des techniques multiples. Elle implique aussi bien des problèmes d'ergonomie que de méthodologie de conception et d'implantation. Ces tâches obéissent à certaines règles dont l'une des plus importantes est la séparation entre les fonctionnalités du système et l'interface interactive. La difficulté principale est d'aborder la réalisation selon une logique qui tient d'avantage compte de la façon dont l'utilisateur se représente l'information manipulée que des problèmes d'implémentation. De plus, les interfaces multimédias doivent fournir à l'utilisateur le moyen d'intégrer divers types de médias tout en réduisant la charge de connaissance et de compréhension nécessaires pour les manipuler [LAUR 90], [COUT 91].

2.1. Outils d'aide

Dans le but de simplifier la conception des interfaces, des outils d'aide sont disponibles [COUT 86]. Il en existe trois catégories principales qui sont les squelettes d'application, les générateurs d'interface et les boîtes à outils.

- Un squelette d'application regroupe sous la forme d'une architecture réutilisable et extensible, un noyau d'exécution et des services communs à un ensemble de systèmes. Le développement d'une application consiste à compléter les trous du squelette et à redéfinir les parties prédéfinies inadaptées au cas particulier de l'application. Il met donc à la disposition du programmeur une architecture prête à l'emploi. On peut citer le squelette d'application MacApp comme exemple [MacA 90].

- Un générateur d'interface est un ensemble d'outils permettant la génération automatique d'applications interactives exécutables à partir de spécifications de haut-niveau. Il aide à créer et à gérer tous les aspects de l'interface pendant sa création et son utilisation.

- Une boîte à outil est une bibliothèque de procédures pour le développement d'interface utilisateurs. L'éventail des fonctions offertes regroupe la gestion des événements physiques de bas niveau tels que l'affichage de tracés ou les entrées clavier et celle d'entités de dialogue de haut niveau tels que les menus et les ascenseurs. Ces fonctions peuvent être classées en deux catégories : les fonctions de gestion du poste de travail et celles de gestion du dialogue. Les premières ont pour but de cacher le fonctionnement physique du système, les secondes offrent les entités de dialogue.

Pour concevoir cette interface, nous avons choisi le dernier type d'outil d'aide, en l'occurrence, le système X Window System[1] [POUN 89], [SHEI 86]. Malgré la mauvaise modularité et la complexité du fonctionnement [LEE 91], ce type d'outil offre la flexibilité, l'extensibilité et la portabilité que nous cherchions.

Le système X11 est construit sur un modèle à événements. Le modèle à événements a émergé de la réalisation de systèmes graphiques qui considèrent les périphériques comme des sources d'événements. Ceux-ci peuvent être générés, soit par ces périphériques, soit par l'interface elle-même, soit encore par l'application. Quand un événement est généré, il est envoyé à un ou plusieurs gestionnaires d'événements qui sont susceptibles de le reconnaître et de le traiter. L'avantage du modèle réside dans sa capacité à décrire un dialogue multifil, c'est à dire que l'utilisateur peut maintenir plusieurs dialogues en parallèle (dans plusieurs fenêtres appartenant ou non à la même application). Il a la possibilité de passer d'un dialogue à un autre à tout moment de l'interaction. Ce type d'interaction nous permet de pouvoir composer une réponse à un message tout en visualisant le message originel, de modifier des paramètres en cours, d'afficher plusieurs images, etc.

2.2. Caractéristiques

L'interface conçue est une interface graphique qui combine plusieurs styles de dialogue : du type WYSIWIG, langage de commande et manipulation directe.

- WYSIWYG. Ce style qualifie un système interactif graphique où la représentation sur l'écran correspond à la réalité de l'application. Il correspond à la plupart des fonctionnalités de l'interface.

- Manipulation directe. Les objets, les attributs et les relations sur lesquelles on peut effectuer des opérations sont visibles sur l'écran ainsi que les actions sur les représentations visuelles. Ce style de dialogue crée des interfaces puissantes et faciles à utiliser [NANA 90]. Il est présent essentiellement dans l'outil de composition de script.

- Langage de commande. C'est la forme traditionnelle de dialogue avec l'ordinateur. Facile à utiliser et extensible, les erreurs sont néanmoins fréquentes. Le langage de commande est utilisé dans l'exécution des scripts, et de manière générale, dans les dialogues avec l'interprète Lisp.

Cette interface présente les caractéristiques habituelles à ce genre d'outil. Nous nous sommes efforcés d'en simplifier l'utilisation par les quelques principes simples qui suivent.

2.2.1.Organisation des fenêtres

La fenêtre principale s'ouvre au lancement de l'application, les autres fenêtres sont lancées après sélection d'une option représentée par un bouton ou un menu. Chaque fenêtre peut être quittée par sélection du bouton de sortie. Il n'y a pas de restriction sur le nombre de fenêtres ouvertes de l'application.

L'ensemble des fenêtres est hiérarchisé. La fenêtre mère est la fenêtre principale, ensuite se trouvent les fenêtres correspondant à chaque module puis aux sous-modules et ainsi de suite (cf FIGURE 33). Ceci permet en sortant d'une fenêtre principale de quitter

1. Des compléments sur X sont fournis dans l'annexe B.

toutes les fenêtres appartenant à sa descendance. Ainsi, le fait de quitter la fenêtre principale supprime toutes les fenêtres de l'UA.

Toutes les fenêtres correspondant à un niveau supérieur (c'est-à-dire à une fonction principale de module ou d'outil) sont iconifiables, de même que certaines fenêtres comme celle de la télécommande. En effet, pour faciliter son utilisation, il est souhaitable qu'une fenêtre de ce type soit accessible de manière plus ou moins indépendante de la hiérarchie auquelle elle appartient.

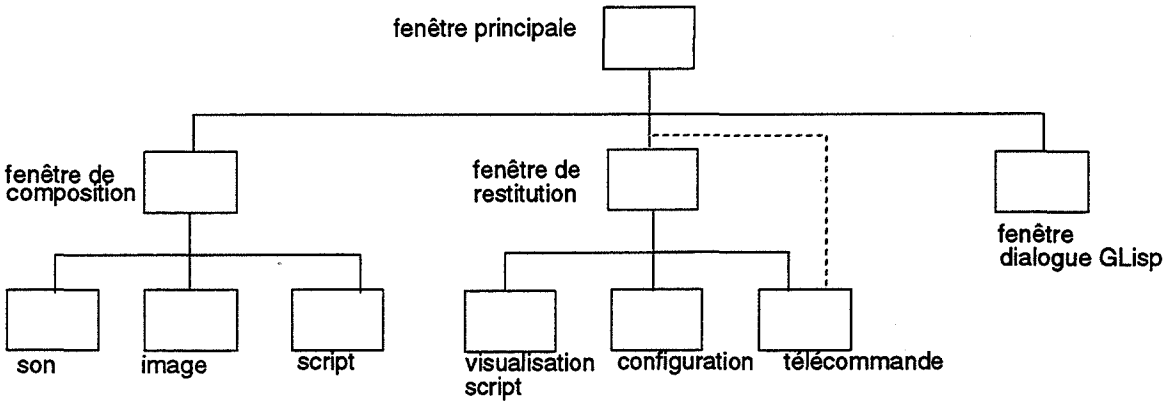


FIGURE 33 : *hiérarchie des fenêtres iconifiables*

Dans la mesure du possible, toutes les fenêtres ont le même aspect et la même structure généraux. En général, on trouve dans la partie supérieure les boutons de commande alignés horizontalement, puis verticalement les champs d'entrée/sortie et les zones de texte. Les boutons de sortie sont toujours placés en haut à gauche de la fenêtre. Les objets de dialogue correspondants aux options non disponibles à un moment donné sont reconnus par un aspect grisé. Les zones de texte ou de visualisation graphique comportent des ascenseurs pour le défilement des informations (cf FIGURE 20)

2.2.2. Messages d'erreur

Les messages d'erreur ou d'avertissement sont affichées dans des fenêtres séparées et non iconifiables. L'ouverture de ce type de fenêtre bloque les entrées de l'utilisateur ; un simple clic à l'intérieur fait disparaître la fenêtre et redonne la main à l'utilisateur.

2.2.3. Configuration de l'interface

- Des accélérateurs ont été mis en place pour faciliter certaines manipulations. Par exemple, la sélection et la lecture d'un message sont lancées sur un double clic, certaines touches du clavier permettent des raccourcis de commandes.

- Les attributs des objets (taille des fenêtres, polices de caractères, etc,) sont regroupés dans un fichier externe à l'application qui peut être modifié par l'utilisateur selon ses préférences (cf 3.2. Aspects de portabilité page 108).

- Il existe une version anglaise et une version française de l'interface. Tous les libellés des boutons, des menus et des titres ainsi que les messages d'erreur sont concernés. Une variable standard d'environnement permet d'obtenir automatiquement au lancement de l'application l'une ou l'autre version.

Remarque : Les fonctions de script restent définies en langue anglaise puisqu'elles sont interprétées par GLisp sous cette syntaxe. On pourrait disposer de paramètres de lancement pour GLisp autorisant l'utilisation d'une autre langue.

De plus, par l'utilisation de fonctions de scripts, l'utilisateur peut configurer son interface, par exemple, ajouter des fenêtres et des boutons avec des actions associées et par extension, utiliser toute fonction de base Xlib. Il suffit de donner en paramètre un nom de fichier contenant le script à exécuter. Ceci est explicité dans les fonctions de script page 120.

2.3. Vue d'ensemble

Le schéma (cf FIGURE 35) représente l'enchaînement des fenêtres principales de l'interface. Pour ne par surcharger le schéma, ne sont pas représentées les fenêtres de dialogue (entrées de noms de fichiers, valeurs de paramètres, informations diverses) ni celles correspondant aux messages d'erreurs. Les flèches correspondent à l'association d'un bouton à l'ouverture d'une fenêtre.

3. IMPLANTATION

L'interface a été implantée en utilisant le système de fenêtrage X Window System [X11 90]. Une caractéristique importante de X par rapport à de nombreux autres systèmes de fenêtrage est qu'il ne définit pas un style particulier d'interface. Au contraire de systèmes comme Microsoft Window ou l'interface du Macintosh, il fournit des mécanismes pour supporter de nombreux types d'interfaces mais laisse le choix au programmeur. Celui-ci a à sa disposition des primitives d'opérations sur les fenêtres mais peut choisir l'aspect et le comportement (Look and Feel) de son interface. Les boutons, menus et autres composants ne sont pas fournis pas X mais par l'ensemble de widgets utilisé. C'est cet ensemble qui détermine l'apparence de l'interface : style des boutons, menus... ainsi que l'enchaînement des widgets et leur comportement. Au niveau de l'interface, s'est posé donc le choix d'un toolkit.

3.1. Choix effectués

Le produit a été conçu dans l'objectif d'être implanté sur des machines de constructeurs différents. La portabilité est donc un impératif. De façon à rester indépendant d'un quelconque constructeur, la version initiale a été développée en utilisant la couche Intrinsics et les Athena Widgets (cf FIGURE 34). Ces deux ensembles font partie de la distribution de X. La couche Intrinsics est garantie compatible avec la couche de base Xlib. L'ensemble des Athena widgets, développé par le MIT, est moins complet que certains autres mais a l'avantage d'être disponible et indépendant d'un constructeur ou d'un environnement graphique. L'application peut donc fonctionner sur toute machine

Unix ayant un serveur X et la distribution X11. Bien que la première adaptation utilise cet ensemble de widgets, la possibilité de concevoir d'autres styles d'interfaces a été envisagée. En particulier, pour fournir à l'utilisateur une interface respectant un "Look and Feel" plus répandu comme OSF/Motif ou Open Look. Une première version sous OSF/Motif a été ainsi développée.

3.2. Aspects de portabilité

Pour faciliter le passage d'un style d'interface à un autre, certaines précautions ont été prises. Toutes les spécifications relatives à l'interface sont séparées de l'application exceptées les fonctions X considérées comme standard comme les primitives de base Xlib utilisées dans les affichages et certaines fonctions de la couche Intrinsics. En effet, l'utilisation du protocole X implique qu'un programme X utilisant la bibliothèque de base Xlib soit portable sur toute machine ayant un serveur X. De plus, la distribution de X assure la standardisation des couches de base Xlib et les Intrinsics. Tout ce qui est spécifique à l'ensemble de widgets utilisé (i.e. Athena Widget Set) est traité de façon particulière. La gestion des fenêtres de l'application est assurée par un programme unique, le changement de style d'interface nécessite donc la réécriture d'un seul fichier. Au niveau des interactions entre l'application et l'interface, toutes les fonctions particulières aux widgets ou manipulant leurs attributs sont redéfinies séparément de l'application. L'utilisation d'un autre ensemble de widgets implique la redéfinition de ces fonctions mais sans modification des spécifications propres à l'application. Cette redéfinition est plus ou moins simple selon les nouveaux widgets employés et leurs caractéristiques. Elle consiste en fait à associer aux fonctions prédéfinies, les actions correspondantes des widgets. Dans la plupart des cas, le changement est trivial.

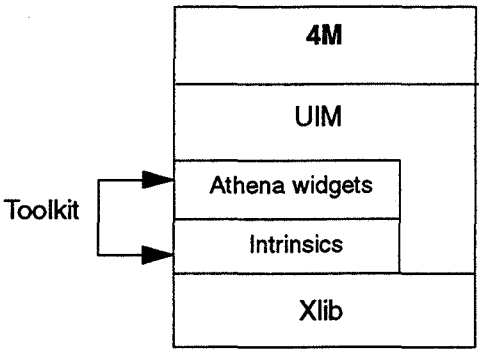
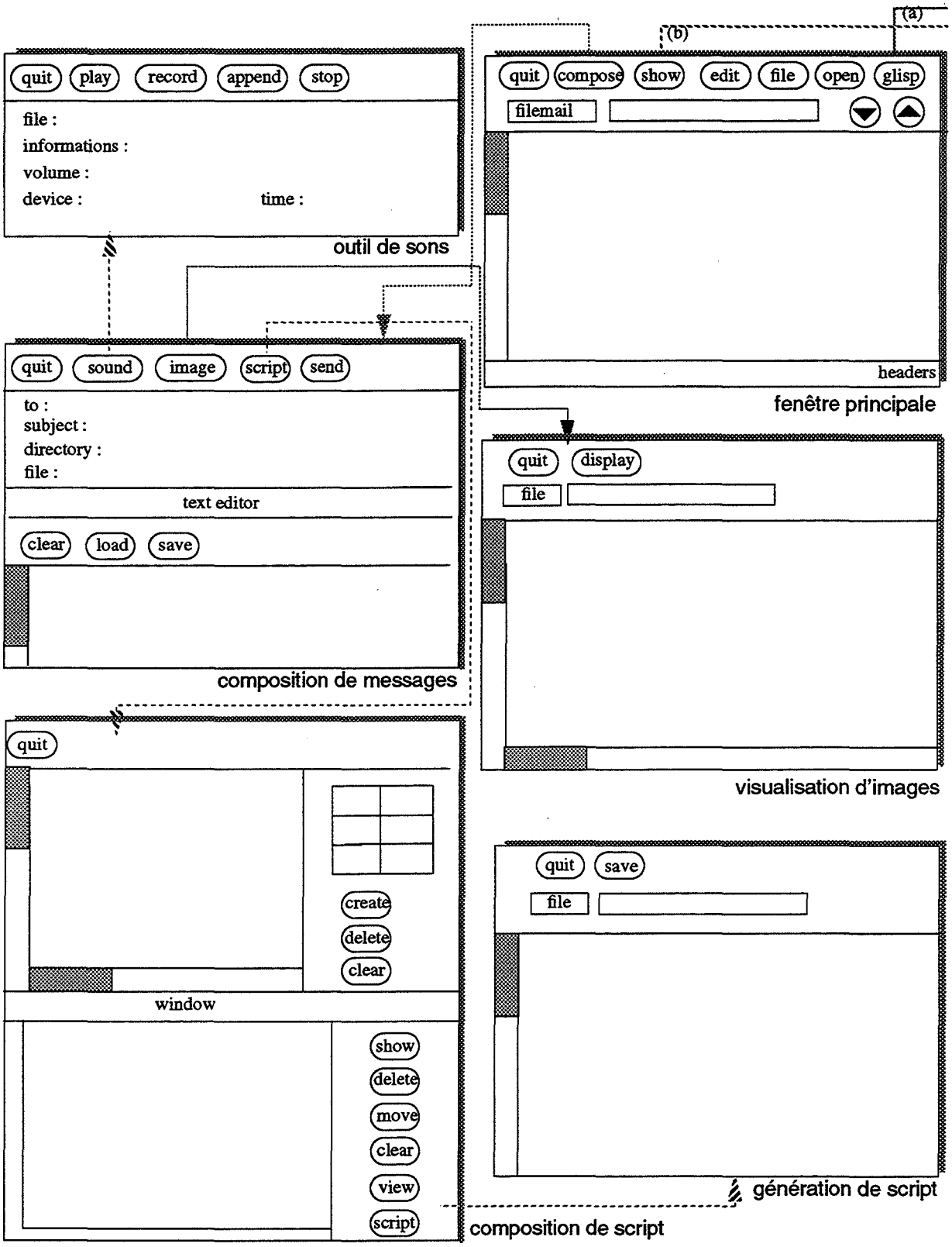


FIGURE 34 : *architecture de l'interface*

Les attributs des widgets reflétant l'aspect de l'interface (taille des fenêtres, messages, labels, fonts, etc) sont définis dans un fichier de ressources X, chargé automatiquement au départ de l'application par le Resource Manager. Ce gestionnaire de ressources permet de spécifier et de modifier les caractéristiques propres à l'interface de façon indépendante de l'application. Il est donc possible de modifier ces attributs selon les widgets employés ou

pour satisfaire certaines préférences de l'utilisateur. C'est ce principe qui permet d'utiliser la version française.

Les divers aspects évoqués ci-dessus ont montré leur utilité pour le portage de l'application sur un matériel HP. Ce portage a conduit à récrire l'interface en utilisant l'ensemble de widgets Motif de façon à rester cohérent avec l'environnement graphique HP et fournir une version plus universelle. Cette nouvelle version a été conçue de manière à respecter la logique de la première au niveau de l'aspect des fenêtres et de leur enchaînement.


(a)
(b)
quit
play
record
append
stop
file :
informations :
volume :
device :
time :
outil de sons
quit
compose
show
edit
file
open
glisp
filemail
headers
fenêtre principale
quit
sound
image
script
send
to :
subject :
directory :
file :
text editor
clear
load
save
composition de messages
quit
display
file
visualisation d'images
quit
create
delete
clear
window
show
delete
move
clear
view
script
composition de script
quit
save
file
génération de script

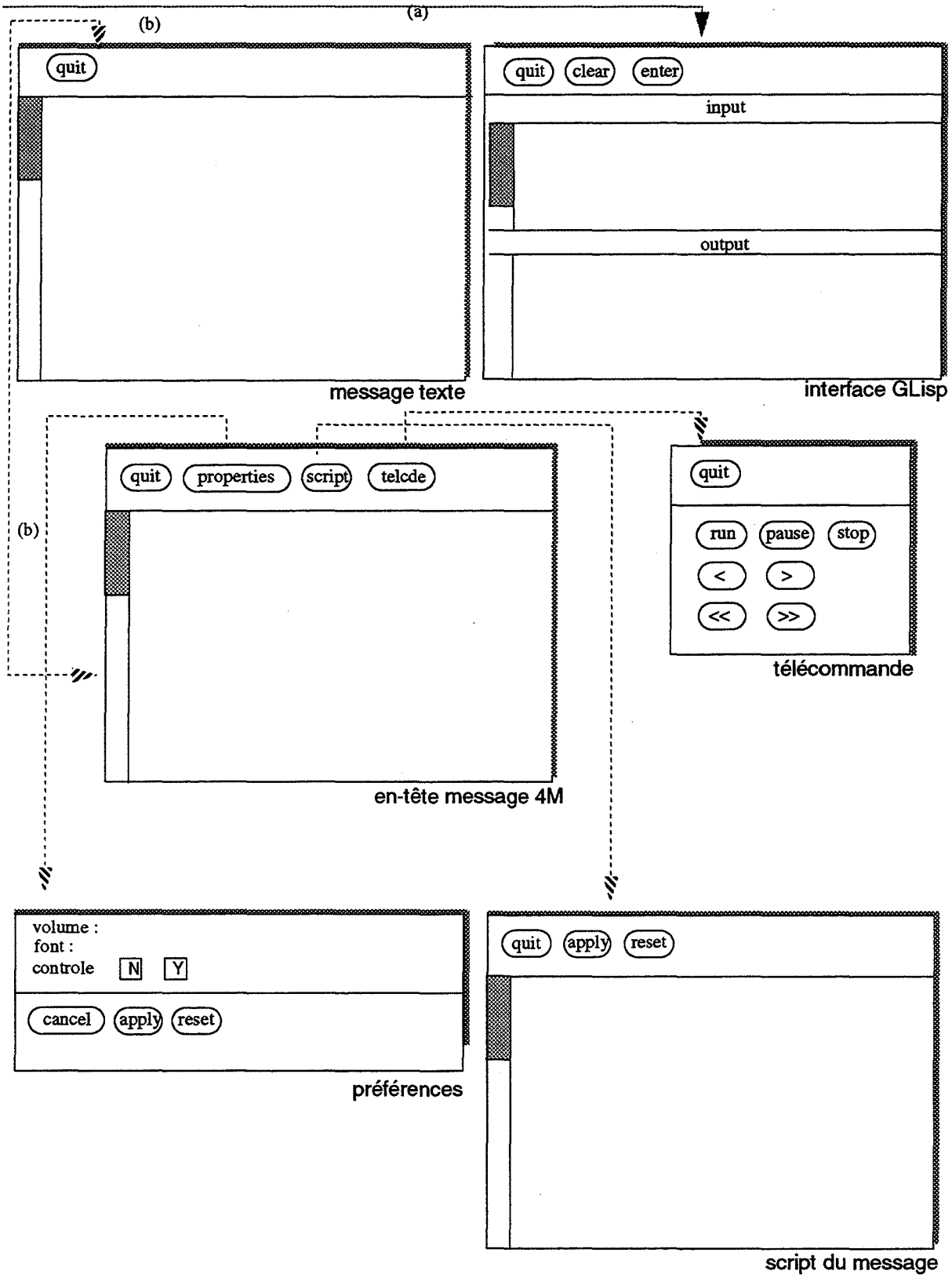


FIGURE 35 : *enchaînement des fenêtres*

CHAPITRE 9

Scripts 4M

Nous venons de présenter l'architecture générale du système et son fonctionnement. Les divers outils proposés permettent ainsi de composer et de restituer des messages multimédias. La partie la plus intéressante de l'application réside dans ces messages 4M. Nous l'avons dit, les messages manipulés par l'outil sont des messagers, c'est à dire qu'ils véhiculent à la fois les informations du message et son comportement sous la forme d'un script. Pour faciliter la tâche de l'utilisateur lors de la composition des scripts, nous fournissons un langage de commande dont la forme la plus utilisée est un ensemble de fonctions simplifiées à partir du langage d'extension GLisp. Ce chapitre décrit ces diverses fonctions et leur exécution ; il termine en montrant quelques exemples de messages ainsi composés.

1. FONCTIONS DE SCRIPT

Des fonctions spécifiques sont fournies à l'utilisateur pour la composition des scripts des messages. Pour respecter l'optique des langages de commande, nous avons choisi de définir des fonctions de script aussi simples que possible et manipulant peu de paramètres. L'ensemble des fonctions décrites ci-après est suffisant pour créer des messages multimédias interactifs.

On distingue trois types de fonctions :

- les fonctions de présentation visuelle et auditive du message,
- les fonctions de synchronisation,
- les fonctions auxiliaires comme celles de modifications de l'environnement de l'utilisateur ou du comportement du message.

Il est possible de définir ensuite ses propres fonctions au fur et à mesure des besoins, directement dans les scripts transmis. Ces fonctions sont écrites en GLisp.

1.1. Langage d'extension GLisp

1.1.1 Caractéristiques

GLisp est un des langages de programmation du système InTalk [GIRA 92]. Comme nous l'avons vu, le système InTalk consiste en un ensemble de bibliothèques qui permet d'incorporer un langage script dans une application développée dans un langage compilé, tel que C. Le système Intalk fournit deux langages de commande qui peuvent être implantés à l'intérieur d'une application : CTalk et GLisp. Le premier est un langage structuré de haut-niveau dont la syntaxe est voisine de C. Il est traduit dans une forme intermédiaire, GLisp, dont la syntaxe est très proche de Lisp. Le langage GLisp peut également être utilisé comme langage de commande dans l'application ainsi obtenue. Dans les deux cas, les fonctions disponibles dans le système sont identiques. Parce que Lisp nous semble d'un accès plus aisé à l'utilisateur néophyte (et aussi parce que la couche CTalk n'était pas disponible au début du projet), nous avons choisi la version GLisp plutôt que la version CTalk.

GLisp est un petit système Lisp expérimental, formé d'un noyau minimal basé sur des concepts simples. Ses caractéristiques sont :

- *la portabilité* ; Il est écrit entièrement en C et fait peu d'hypothèses sur le matériel sur lequel il tourne. Il a été porté sur des machines et des systèmes d'exploitation divers : Sun 3/60, Sparc stations Sun 4, Macintosh, Atari, PC et Alpha.

- *un espace mémoire réduit* ; Bien qu'il représente une implantation complète du langage, il n'ajoute que 150 à 350 kilobytes à l'application qui l'intègre.

- *l'intégration dans les applications* ; Conçu comme un outil destiné à simplifier l'écriture d'une application il fournit l'accès aux structures et types de données C, aux variables et fonctions de l'application. Il permet aussi l'accès aux types et aux structures de données Lisp à partir de l'application.

- *l'efficacité* ; Bien qu'il ne soit pas aussi efficace qu'un système Lisp complet muni de son compilateur associé, il offre un compromis raisonnable en vitesse. GLisp n'est pas un interpréteur traditionnel mais plutôt un générateur de "threaded code". Chaque fonction définie est traduite en une liste d'adresses de procédures exécutables ; le code est ensuite exécuté par une boucle écrite en C. Cette technique offre un bon compromis entre l'efficacité et la portabilité.

GLisp est un dialecte de *Scheme* [SPRI 92] de par sa simplicité et l'utilisation de la liaison lexicale ; il fournit aussi une combinaison de fonctions de *common-lisp* et *Le_Lisp*. Il offre un grand nombre de structures de contrôle (cond, if, while, when, repeat, catch, throw, etc.) et optimise les fonctions récursives terminales.

1.1.2. Syntaxe des fonctions

La syntaxe et la sémantique du langage GLisp ont été choisies relativement proches de celles de Common Lisp, dans les limites du raisonnable. Les autres influences sensibles sont Scheme et Le_Lisp. Les fonctions utilisées dans les scripts respectent la syntaxe Glisp, elles ont donc la forme :

```
(fonction [paramètre 1] [paramètre2]... [paramètre n])
```

Une fonction peut rendre une valeur significative, dans ce cas, la fonction GLisp `setq` permet de l'affecter à une variable. Selon la fonction utilisée, il est possible d'utiliser pour les paramètres des valeurs par défaut de façon à simplifier certaines spécifications.

```
(setq var (fonction par 1 par 2... par n))
```

1.1.3. Interface avec 4M

L'interprète GLisp est intégré dans l'application 4M. Des fonctions d'initialisation sont lancées au départ puis toutes les fonctions de script, que ce soit un script de message ou non, sont données en entrée à l'interpréteur GLisp qui est chargé de les exécuter. Pour cela, on définit des tampons d'entrée/sortie dans lesquels sont chargées les expressions à exécuter. En effet, en plus des types internes qui correspondent aux objets primitifs, tableaux, structures et aux différents types de fonctions, GLisp manipule des structures de données identiques aux flots de CommonLisp qui représentent des objets qui peuvent être des sources ou des destinations de données (chaînes, fichiers, fenêtres). Des fonctions sont disponibles pour lire et écrire des données formattées ou non à partir de ces flots de données. GLisp renvoie les résultats dans les tampons de sortie. C'est l'application qui gère l'impression éventuelle des résultats et des messages d'erreur dans les fenêtres de l'interface.

1.2. Fonctions de présentation

Les fonctions de présentation concernent la restitution du corps du message. Elles sont classées selon les objets manipulés qui sont les fenêtres et les parties de message.

1.2.1. Fonctions sur les fenêtres

Dans les fonctions ci-après, "idf_window" correspond à l'identificateur de fenêtre rendu par la fonction de création.

- `(window lg ht)` : crée la fenêtre de largeur `lg` pixels et de hauteur `ht` pixels (mais ne l'affiche pas). Ces valeurs doivent être comprises entre 0 et 100. La valeur 0 est la valeur par défaut ; dans ce cas, la fenêtre a une taille d'environ 300 par 300 pixels. Les dimensions sont ajustées à la taille réelle de l'écran, ce qui permet d'adapter les tailles des fenêtres selon le dispositif d'affichage existant. Cette fenêtre comporte un cadre et des ascenseurs horizontal et vertical pour le défilement des informations. Cette fonction renvoie un identificateur de fenêtre (`idf_window`) nécessaire aux fonctions

de manipulation décrites ci-après.

`-(unmap idf_window)`: ferme la fenêtre référencée pas l'identificateur. La fenêtre est toujours existante mais n'est plus visible.

`-(xmap idf_window)` : affiche la fenêtre référencée par l'identificateur, dans le cas d'un premier appel après la fonction de création. Si la fenêtre a été fermée auparavant par une fonction unmap, elle redevient visible à l'écran.

`-(clear idf_window)`: efface le contenu de la fenêtre référencée par l'identificateur.

`-(destroy idf_window)` : détruit la fenêtre référencée par l'identificateur.

Remarque : la destruction d'une fenêtre n'implique pas la suppression des textes et images contenus qui peuvent être référencés ailleurs.

1.2.2. Fonctions sur les objets multimédias

Dans les fonctions suivantes, les positions x et y sont des coordonnées dans l'espace de la fenêtre ; le point d'origine (0,0) correspond au coin supérieur gauche de la fenêtre. Toutes les coordonnées sont exprimées en pixels. Pour simplifier leur utilisation elles sont limitées entre 0 et 100. La valeur 0 peut être utilisée comme une coordonnée en x ou en y, le positionnement est alors automatique (cf 1.2.5. Positionnement page 118).

`-(show_text filename idf_window x y)` : affiche le texte contenu dans le fichier de nom `filename` dans la fenêtre `idf_window` aux positions `x` (horizontale) et `y` (verticale).

`-(show_picture filename idf_window x y)`: affiche l'image contenue dans le fichier `filename` dans la fenêtre `idf_window` aux positions `x` et `y`.

`-(show_button label idf_window x y)`: affiche un bouton avec le libellé label dans la fenêtre `idf_window` aux positions `x` et `y`.

`-(show_warning string idf_window x y)`: affiche le message d'avertissement sonore donné par la chaine de caractères `string` dans la fenêtre `idf_window` aux positions `x` et `y`.

`-(play_sound filename volume)`: joue le fichier sonore de nom filename au volume spécifié. Ce volume doit être compris entre 0 et 100. La valeur 0 n'est pas significative et est utilisée comme valeur par défaut. Dans ce cas, le volume est égal à 20. Il en est de même si la valeur donnée n'est pas comprise dans les limites fixées. Ceci permet d'adapter une échelle au volume réel propre à chaque machine.

Remarque : ces fonctions ne renvoient pas de valeur significative, elles ne sont donc utilisables que pour des affichages simples. Il est impossible de manipuler ensuite les parties de message correspondantes. Les fonctions de création et de manipulations suivantes permettent de réaliser des actions plus complexes.

1.2.3. Fonctions de création

Pour manipuler des parties de messages, on leur associe un identificateur rendu par une fonction de création. Elle consiste à associer un composant du message (fichier contenant

un objet multimédia ou objet actif) à une structure identifiable. On note "idf_part" cet identificateur.

`-(text filename)` : crée un composant de type texte associé au fichier de nom `filename` et renvoie un identificateur.

`-(picture filename)` : même principe avec un fichier image.

`-(sound filename)` : même principe un fichier son.

`-(button label)` : crée un bouton avec le libellé `label` et renvoie un identificateur.

`-(warning string)` : crée un avertissement avec le libellé `string` et renvoie un identificateur. Par défaut la chaîne de caractère est "there is a sound here" (dans la version anglaise).

Remarque : La différence entre les avertissements et les boutons est simplement visuelle (cf FIGURE 36).

FIGURE 36 : *bouton et avertisseur*

`-(action idf_part action)` : associe une `action` (sous forme d'expression GLisp) à un objet actif référencé par `idf_part`. Cette fonction n'est valide que pour les composants de type bouton ou avertissement. Elle est exécutée lorsque l'utilisateur clique sur la représentation graphique correspondante.

1.2.4. Fonctions de manipulation

Les fonctions suivantes sont des fonctions génériques dans le sens où elles agissent sur des objets divers selon leur type de données. Ceci est rendu possible par la spécification du paramètre `idf_part` correspondant à l'identificateur associé au composant rendu par la fonction de création. Dans les fonctions suivantes `idf_window` correspond à l'identificateur de la fenêtre, `x` et `y` aux coordonnées en pixels dans l'espace de la fenêtre à partir du point d'origine supérieur gauche.

`-(show idf_part idf_window x y)` : affiche le composant référencé par l'identificateur dans la fenêtre correspondante aux positions *x* et *y*. Selon le type de composant, cette fonction appelle l'une des fonctions de restitution décrites au paragraphe correspondant page 116.

`-(move idf_part idf_window x y)` : translate le composant référencé aux nouvelles positions `x` et `y` dans la fenêtre correspondante à `idf_window`.

`-(erase idf_part idf_window)` : efface le composant de la fenêtre où il se trouve.

`-(translate idf_part idf_window`$_a$ `idf_window`$_b$ `x y)` : transfert le composant de la fenêtre (a) à la fenêtre (b) aux nouvelles positions `x` et `y`. Cette fonction équivaut aux actions successives `(erase idf_part idf_window`$_a$`)` et

(show idf_part idf_window$_b$ x y).

Remarque: Les quatre fonctions ci-dessus ne sont valides que pour des composants de type visuel. Les actions `move`, `erase` et `translate` agissent sur un seul composant affiché dans une fenêtre. S'il y a plusieurs occurrences du même composant (même identificateur) dans une fenêtre, ces actions ne fonctionneront que sur la première.

`-(play idf_part volume)` : produit le son référencé par l'identificateur avec le `volume` spécifié. Cette fonction appelle en fait la fonction `play_sound`.

`-(delete idf_part)` : détruit le composant spécifié. Il disparaît de toutes les fenêtres dans lesquelles il est affiché.

`-(inactive idf_part)` : inactive un composant de type bouton ; l'action enregistrée pour le composant n'est plus disponible.

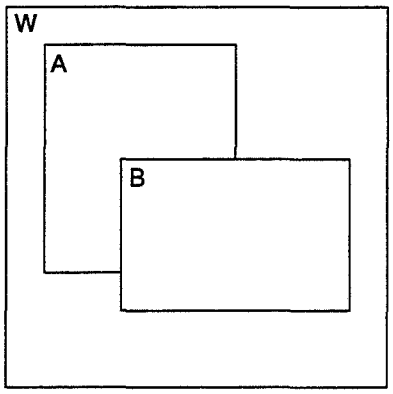
`-(active idf_part)` : active un composant de type bouton. C'est l'état par défaut lors de la création de l'action.

1.2.5. Positionnement

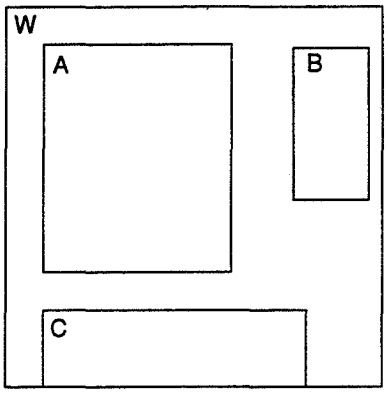
Le positionnement des composants du message est spécifié dans toutes les fonctions de présentation au moyen de coordonnées. Il est dépendant d'une variable d'environnement (OVERLAP) qui permet à l'utilisateur d'autoriser ou non la superposition des parties du message composé. Par défaut, cette variable est fausse (superposition interdite). L'affichage obéit aux principes suivants :

- Si OVERLAP est vraie et que les positions en x et y sont comprises entre les limites autorisées, alors le bloc d'affichage correspondant est affiché ou déplacé aux coordonnées spécifiées (cf FIGURE 37) ;

- Si OVERLAP est fausse, et que le positionnement n'interfère pas avec les autres objets, alors les coordonnées spécifiées sont respectées. Si le bloc d'affichage chevauche tout ou partie d'un autre déjà présent, alors le positionnement se fait à la première position libre dans la fenêtre, en général, en dessous des autres parties affichées. Si `x`, respectivement `y`, vaut 0 (valeur par défaut), alors la partie de message est affichée à la première position libre en vertical, respectivement en horizontal. Dans le cas où `x` et `y` valent 0, le positionnement est réalisé à la première position libre en vertical (cf FIGURE 37). Ces positions par défaut ne sont effectives que si le chevauchement des parties de message n'est pas autorisé.




```
(OVERLAP true)
(show A W 10 10)
(show B W 40 50)
```




```
(show A W 10 10)
(show B W 20 0)
(show C W 0 0)
```

(la variable OVERLAP est fausse par défaut)

FIGURE 37 : *exemples de positionnement*

1.3. Fonctions de synchronisation

Ces fonctions permettent la synchronisation temporelle du message.

`-(par` exp_1 exp_2 exp_n`)`: définit une séquence d'exécution simultanée de fonctions script. Les fonctions spécifiées en paramètres sont exécutées en parallèle.

`-(seq` exp_1 exp_2 exp_n`)`: définit une séquence d'exécution de tâches séquentielles.

`-(sleep nb)`: suspend l'exécution du script pendant `nb` secondes. .

`-(evt type action)`: définit une action (expression GLisp) déclenchée sur un événement. Celui-ci peut être de deux types, soit un événement clavier (appui sur une touche) soit un événement généré par la souris.

`-(pause)`: suspend la tâche en cours d'exécution jusqu'à `resume`.

`-(resume)`: reprend la tâche suspendue à l'endroit où elle s'est arrêtée.

Plus de détails sont donnés sur les tâches en cours d'exécution au paragraphe 2.1. Notion de tâches page 121.

1.4. Fonctions externes

Il est possible de définir des programmes GLisp utilisant des fonctions existantes telles que des fonctions C ou des fonctions de base Xlib. Celles-ci sont rendues accessibles par l'utilitaire TGen qui permet la génération automatique d'interface entre un programme C et le système InTalk. Les objets que l'on souhaite pouvoir utiliser depuis GLisp (fonctions, variables, tableaux et structures) sont déclarés sous une forme très simple, voisine de C. TGen est utilisé pour compiler ces déclarations qu'il transforme en un programme C qui rend accessibles aux programmes GLisp les objets ainsi définis.

1.4.1. Fonctions d'environnement

Ces fonctions permettent de modifier des paramètres de production du message ; l'utilisateur peut changer ainsi le volume sonore ou la police de caractères, ou encore la variable de positionnement. Ces paramètres correspondent à des variables C qui sont compilées par TGen et interprétées par GLisp comme des fonctions.

`-(variable)` : rend la valeur contenue dans la variable.

`-(variable valeur)`: affecte une valeur à une variable d'environnement.

`-(reset)`: remet à jour toutes les variables d'environnement avec leur valeur par défaut.

exemples :

`(fontname "helvetica")` : modifie la police de caractères en fonction de la police spécifiée

`(volume)` : rend la valeur du volume par défaut.

1.4.2. Fonctions X11

Bien que l'on puisse utiliser directement des fonctions X standards, pour simplifier l'écriture et en particulier le nombre de paramètres, certaines des fonctions graphiques les plus utilisées ont été redéfinies. Ce sont par exemple :

`-(draw_string string idf_window x y)`: affiche la ligne de texte dans la fenêtre référencée aux positions `x` et `y`.

`-(draw_point idf_window x y)`: dessine un point dans la fenêtre référencée, aux coordonnées `x` et `y`.

`-(draw_circle idf_window x y r)`: dessine un cercle de rayon *r* dans la fenêtre référencée, aux positions `x` et `y`.

`-(draw_rect idf_window x y lg ht)`: dessine un rectangle de hauteur `ht` et de largeur `lg` dans la fenêtre référencée, le coin supérieur gauche correspondant aux valeurs `x` et `y`.

1.5. Fonctions de comportement

Ces fonctions agissent sur le comportement du message, et plus précisément sur le devenir du message. Ces fonctions sont pour l'instant limitées au renvoi du message à l'expéditeur ou à d'autres destinataires, le message pouvant être modifié ou complété au préalable.

`-(sender)` : cette variable fonctionne de la même façon que les variables d'environnement. Par défaut l'adresse de l'expéditeur du message lui est affectée mais elle peut être modifiée.

`-(reply)`: renvoie le message au(x) destinataire(s) dont les adresses sont fournies par la variable (sender).

1.6. Configuration

En utilisant les diverses fonctions que nous venons de voir, l'utilisateur a la possibilité de concevoir des scripts complexes. Ces scripts accompagnent les messages multimédias mais peuvent aussi être utilisés pour configurer l'environnement de l'utilisateur et même l'interface en elle-même. En effet, comme nous l'avons introduit au chapitre précédent, il est possible de fournir en paramètre au lancement de l'application un nom de fichier contenant un script. Celui-ci est aussitôt exécuté par l'interpréteur GLisp. Ce script peut donc créer des fenêtres, des boutons, des actions, etc, comme le ferait n'importe quel programme Xlib. Il peut aussi définir de nouvelles fonctions utilisables par la suite. Enfin, il ne faut pas oublier que GLisp est un langage Lisp et donc que l'utilisateur peut employer les fonctions définies ou en créer d'autres à tout moment grâce à l'interface avec GLisp.

2. ÉXECUTION DES SCRIPTS

2.1. Notion de tâches

Le script correspond à une certaine activité, en l'occurrence la restitution du message. Cette activité est effectué par plusieurs tâches qui coopèrent les unes avec les autres. Elles sont réalisées selon les cas, soit par des acteurs différents crées spécifiquement et uniquement pour cela, soit par un même acteur spécialisé dans une certaine mission. On peut avoir par exemple, un acteur spécialisé dans l'affichage d'une image, qui est sélectionné par un message, pour afficher une image particulière. Chaque événement correspondant au déclenchement d'une action entraîne la création d'une tâche et l'envoi d'un message à l'acteur adéquat si celui-ci existe. Si ce n'est pas le cas, un acteur est crée pour cette tâche.

exemple :

tâche A : *ouvrir une fenêtre* ce qui se traduit par l'envoi d'un message à l'acteur responsable de l'ouverture d'une fenêtre

tâche B : *afficher l'image* : envoi d'un message à l'acteur d'affichage d'images et celui de production du son (cf ligne suivante)

tâche C : *produire le bruit du train en même temps que l'image*

tâche D : *afficher le texte une fois le son terminé* : envoi d'un message à l'acteur d'affichage de textes.

Une tâche peut regrouper et agir sur différents objets (multimédias, médias ou relations). Une tâche confiée à un acteur peut consister en une action ou un ensemble d'actions. L'ensemble des tâches est géré par un gestionnaire de tâches dont les fonctions sont la création, la suppression et la synchronisation de celles-ci. Il permet leur exécution séquentielle ou simultanée

2.2. Synchronisation des tâches

Les tâches sont synchronisées par le gestionnaire de tâches en concordance avec le script du message (cf FIGURE 38). Elles peuvent être exécutées en séquentiel ou en parallèle. Pour cela, on définit une file de tâches ayant chacune une priorité. Deux tâches ayant la même priorité sont exécutées concurremment. Le gestionnaire utilise un principe d'ordonnancement et sélectionne les tâches selon un ordre de priorité décroissant dans la file des tâches éligibles. Pour cela, la valeur courante de la priorité est diminuée à chaque tâche terminée. Le gestionnaire est aussi responsable de la gestion des périphériques d'entrée/sortie et en particulier celui du son (en effet, deux sons ne peuvent être produits en même temps même si l'équipement est stéréo).

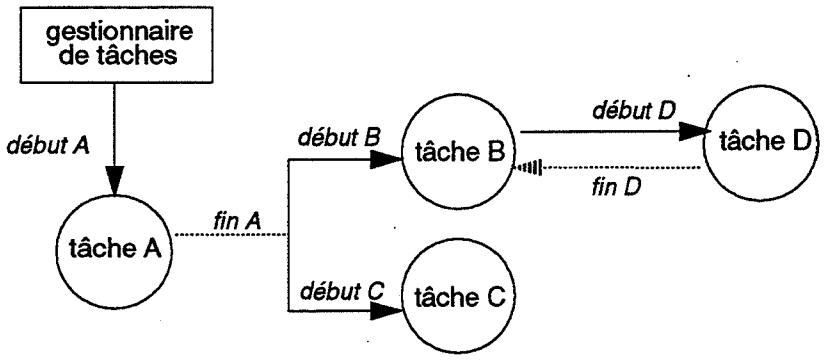


FIGURE 38 : *gestion des tâches*

Reprenons l'exemple précédent. Au début, le gestionnaire lance la tâche A qui s'exécute entièrement puis il lance B et C en parallèle. D est exécutée à la suite de B sans tenir compte du déroulement de C. Le script est terminé quand les tâches D et C le sont.

Pour tenir compte de l'aspect asynchrone du fonctionnement de X11, le gestionnaire de tâches est explicitement basculé en mode synchrone de façon à respecter le synchronisme des tâches. On revient au comportement normal de X dès que le script est terminé.

2.3. Implantation des tâches

Nous associons un processus léger dit LWP ("Light Weight Process") à chaque acteur réalisant une tâche. Les processus légers sont appelés ainsi à cause de leur faible coût en temps d'exécution et en espace nécessaire. Ils peuvent communiquer entre eux facilement de façon explicite ou par envoi de messages puisqu'ils partagent le même espace d'adressage. Il est possible que toutes les tâches soient exécutées dans un simple espace de mémoire partagée, par exemple, l'espace d'adressage d'un processus Unix [BUHR 92]. Ces processus sont particulièrement adaptés à notre cas car ils permettent d'exécuter les tâches indépendamment et facilitent leur synchronisation et leurs communications. Ils sont créés et gérés par le gestionnaire de tâches (cf FIGURE 39).

- Exemple

Prenons l'exemple d'un script de message simple, sans s'occuper pour l'instant de sa signification :

```
1 (setq W (window 200 200))                priorité : 99
2 (setq I (picture "image.rs" ))                    98
3 (setq T (text "texte.txt"))                       97
4 (xmap W)                                          96
5 (par
6 (show I W 3 3)                                    95
7 (show T 4 5))                                     95
8 (play_sound "son.au" 5)                           93
```

Une première lecture du script permet d'associer à chaque tâche (dans ce cas, une expression Lisp) un LWP avec une certaine priorité. L'affectation des priorités est séquentielle au fur et à mesure de la lecture des expressions et par ordre décroissant. Une fois les processus créés[1], on lance le scheduler chargé de leur synchronisation. La fonction (`par f1, f2`) implique que l'ordre des priorités reste le même pour les fonctions incluses. A partir de la liste des processus, le scheduler sélectionne la plus forte priorité et ainsi de suite, soit les lignes 1 puis 2, 3 et 4. Les lignes 6 et 7 ayant la même priorité, engendrent une exécution en parallèle. Ensuite est exécutée la ligne 8. On peut remarquer que l'ordre des priorités ne se suit pas puisque chaque processus décrémente la priorité courante.

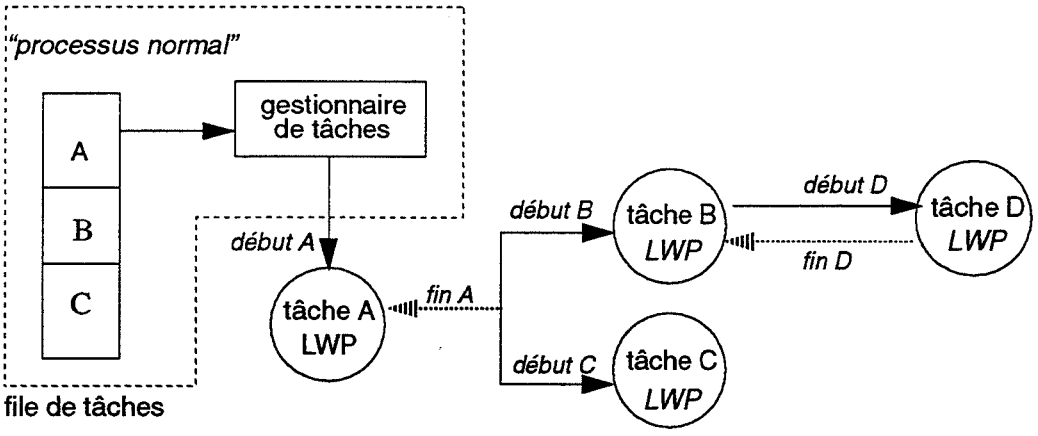


FIGURE 39 : *implantation des tâches*

1. La création d'un processus n'entraîne pas obligatoirement son exécution.

3. EXEMPLES DE MESSAGES

3.1. Structure générale

Les scripts des messages utilisent les fonctions de scripts du langage de commande que nous avons décrites précédemment. La plupart des scripts des messages ont une structure commune, c'est à dire une première partie de création d'objets référencés puis une seconde partie de fonctions appliquées sur ces objets. Il n'existe aucun ordre à respecter, une déclaration d'objets peut donc se trouver n'importe où dans le script. Cependant si le script est conçu par l'outil de création de script, elles sont regroupées au début du script lors de sa génération du fait du fonctionnement interne de l'outil. Il est évident qu'une fonction appliquée à un objet non défini entraînera une erreur de l'interprète GLisp. En cas d'erreur le scénario continue de se dérouler si cela est possible et les messages d'erreur s'affichent dans des fenêtres successives.

Les exemples qui suivent montrent quelques unes des possibilités de messages multimédias. Ils ont été choisis volontairement assez simples pour illustrer au mieux certaines spécificités des scripts.

3.2. Considérations spatiales

Le premier script est un exemple des spécifications spatiales des objets d'un message. Deux types de médias sont prévus : un texte et deux images qui s'affichent dans deux fenêtres distinctes. Le déroulement du message est séquentiel. L'expéditeur a prévu aussi la possibilité de répondre automatiquement au message par un bouton prédéfini. Nous allons expliciter le script du message correspondant.

```
(setq W1 (window 200 200))                                    1
(setq T1 (text "FF/quest.txt"))                               2
(setq P1  picture "FF/bf.rs"))                                3
(setq P2  picture "FF/three.mpt"))                            4
                                                              5
( W1 5 10)                                                    6
( P1 W1 0 10)                                                 7
(setq W2 (window 200 200))                                    8
(xmap W2)                                                     9
(show P2 W2 0 0)                                             10
(reply)                                                      11
```

La première ligne du script crée une fenêtre de taille 200 sur 200 pixels (relativement à la taille de l'écran) et lui affecte un identificateur W1. Les trois lignes suivantes définissent des objets du message en spécifiant leur type et le nom du fichier contenant chacun d'eux. De la même façon, on leur associe un identificateur. Le déroulement du message commence à la ligne 5, il est séquentiel par défaut. La fenêtre W1 s'ouvre et le texte, puis la figure P1 s'affichent l'un après l'autre. On remarquera que l'image est

prévue pour s'afficher à côté du texte car sa position en x, égale à 0, spécifie un positionnement horizontal par défaut. La ligne 8 définit une deuxième fenêtre de taille identique à la première qui est ouverte à la ligne suivante. Dans cette fenêtre, s'affiche la seconde figure (ligne 10). La dernière ligne du script définit un bouton qui s'affiche dans l'en-tête du message et auquel est associée une fonction de renvoi du message. Une fois le script exécuté, on obtient le message de la figure suivante (cf FIGURE 40).

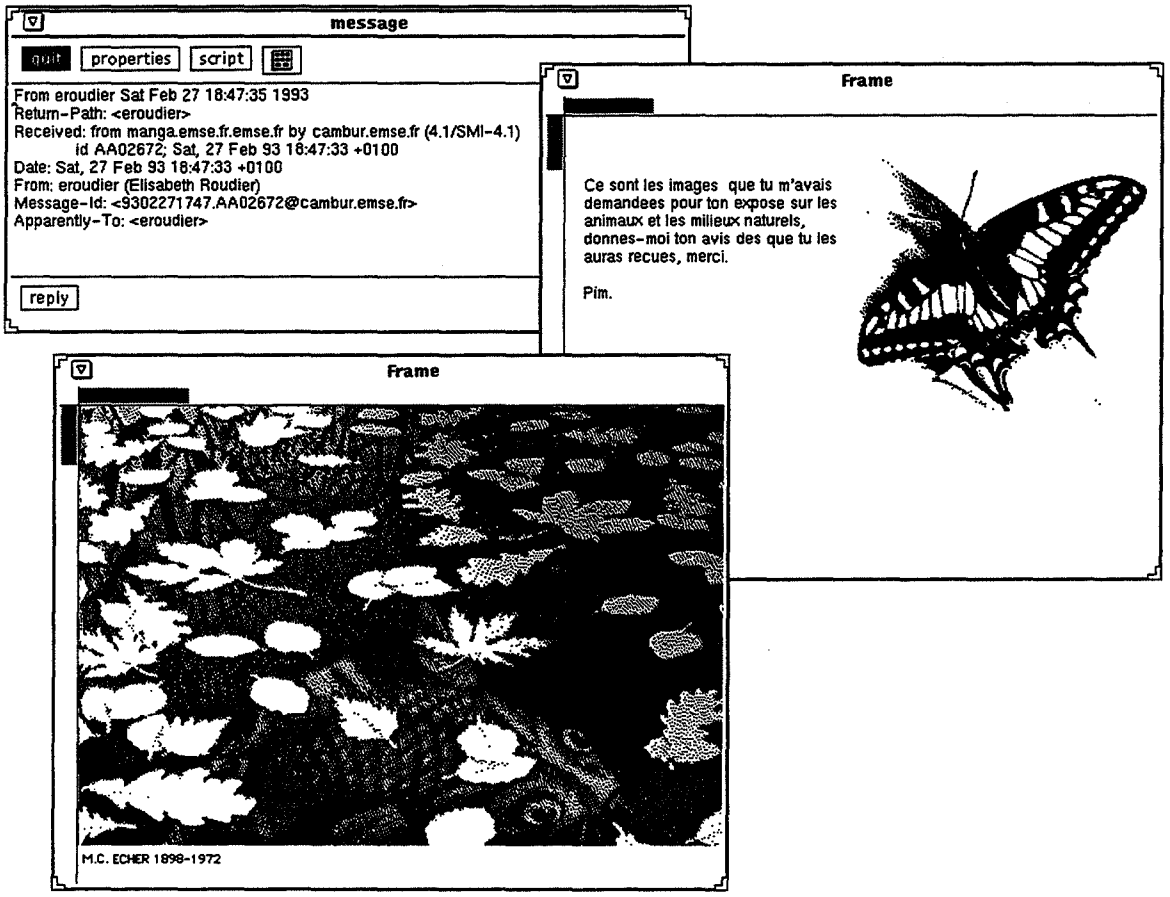


FIGURE 40 : *relations spatiales*

Voici maintenant le script du message, modifié par le destinataire, avant d'être renvoyé comme prévu à son expéditeur :

```
(setq W1 (window 150 150))                    1
(setq T (text "FF/quest.txt"))                2
(xmap W1)                                     3
(show T W1 5 10)                              4
(draw_rect W1 65 200)                         5
(fontname "times-bold")                       6
```

```
(draw_string "Bien recu - Images parfaites. Pam." W1 6 75) 7
```

L'utilisateur a supprimé dans le script les deux images et leur affichage mais a rajouté une réponse avec les lignes 5 à 7. La première dessine un rectangle, la suivante change la police de caractères pour l'affichage à la dernière ligne d'un message situé à l'intérieur du rectangle. On obtient le message résultant (cf FIGURE 41).

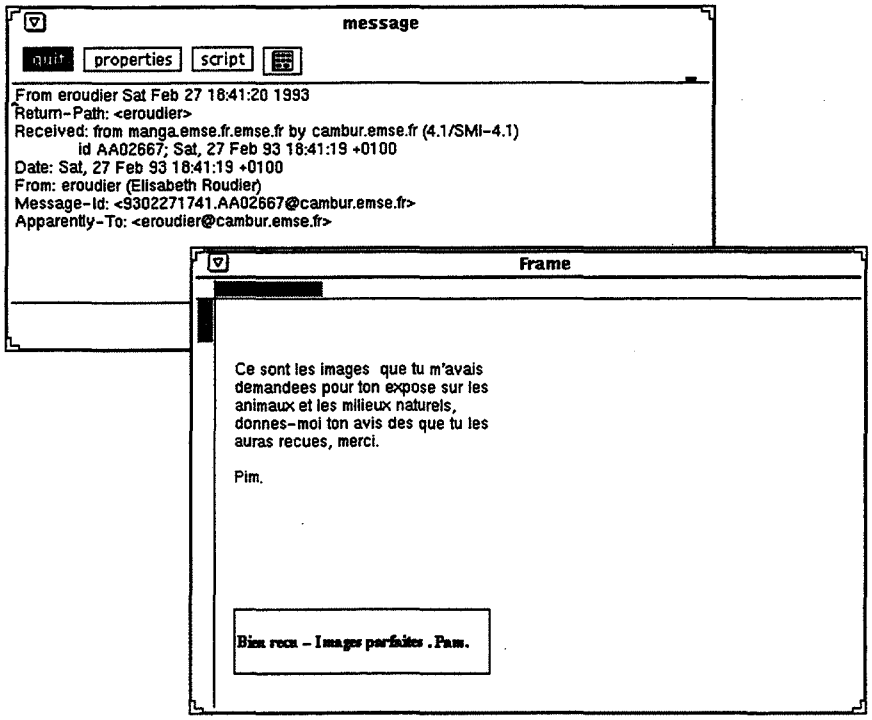


FIGURE 41 *: réponse*

3.3. Coordination temporelle

Cet exemple montre un script de coordination temporelle. Trois types de contrôles temporels sont présents : séquentiel, simultané et indépendant (asynchrone). Le message comporte un texte, une image et un son qui sont produits dans l'ordre suivant : le texte, puis l'image couplée avec le son. Un avertisseur, affiché à la suite, permet de rejouer le son quand le destinataire le souhaite. Le script de ce message a été conçu graphiquement par l'outil de création de script ; le voici :

```
(setq Obj1 (window 350 200))                          1
(setq Obj2 (text "FF/parg.txt"))                      2
(setq Obj3 (picture "FF/train.mpt"))                  3
(setq Obj4 (sound "FF/train.au"))                     4
(setq Obj5 (warning "Ecoute ..."))                    5
(action Obj5 "(play Obj4 30)")                        6
(xmap Obj1)                                           7
```

```
(show Obj2 Obj1 10 10)                                    8
(par                                                      9
(show Obj4 Obj1 0 10)                                    10
(play Obj5 30)                                           11
)                                                        12
(show Obj3 Obj1 10 50)                                   13
```

Les premières lignes du script (lignes de 1 à 5) définissent les objets du message en leur attribuant automatiquement, lors de la génération du script, un identificateur qui sera utilisé dans les fonctions suivantes. Ces identificateurs sont numérotés et affectés directement lors de la génération du script. Pour le bouton "avertisseur" (ligne 3), on ne spécifie pas un nom de fichier mais le libellé que l'on veut afficher en avertissement. La ligne suivante lui associe une action, en l'occurrence jouer le son avec un volume de 30. Après les définitions, vient la présentation du message. De manière séquentielle (mode par défaut) la fenêtre s'ouvre (ligne 7) et le texte s'affiche aux positions spécifiées (ligne 8) à l'intérieur de la fenêtre. Puis le message bascule en mode parallèle grâce à la fonction (par). Les deux expressions suivantes sont donc exécutées simultanément ; elles correspondent à l'affichage de l'image accompagnée du son. On revient au mode séquentiel pour afficher le bouton avertisseur. La présentation du message se termine à cette ligne. Le message obtenu est représenté par la figure suivante (cf FIGURE 42). L'utilisateur peut ensuite écouter de nouveau le son en cliquant sur l'avertisseur.

3.4. Exemple X11

Nous avons vu que n'importe quelle fonction de base Xlib pouvait être incorporée dans un message grâce à l'utilitaire TGen. Voici un exemple de script qui ouvre une fenêtre pour afficher dans un rectangle une ligne de texte puis pour produire un son. La fenêtre s'autodétruit au bout de 3 secondes.

```
(setq D (OpenDisplay))
(setq RW (Root-Window))
(setq CF (Current-Foreground))
(setq CB (Current-Background))
(setq W (XCreateSimpleWindow D RW 100 50 350 150 1 CF CB))
(setq GC (Graphic-Context D))
(XMapRaised D W)
(XFlush D)
(sleep 2)
(XDrawRectangle D W GC 10 10 100 50)
(setq text "Hello, World!")
(XDrawString D W GC 15 25 text (length text))
(XFlush D)
(play_sound "/FF/laugh.au" 10)
(sleep 3)
```

```
(XDestroyWindow D W)
(XFlush D)
```

Toutes les fonctions commençant par X sont des fonctions Xlib appelées par GLisp. On obtient ainsi le résultat suivant (cf FIGURE 43).

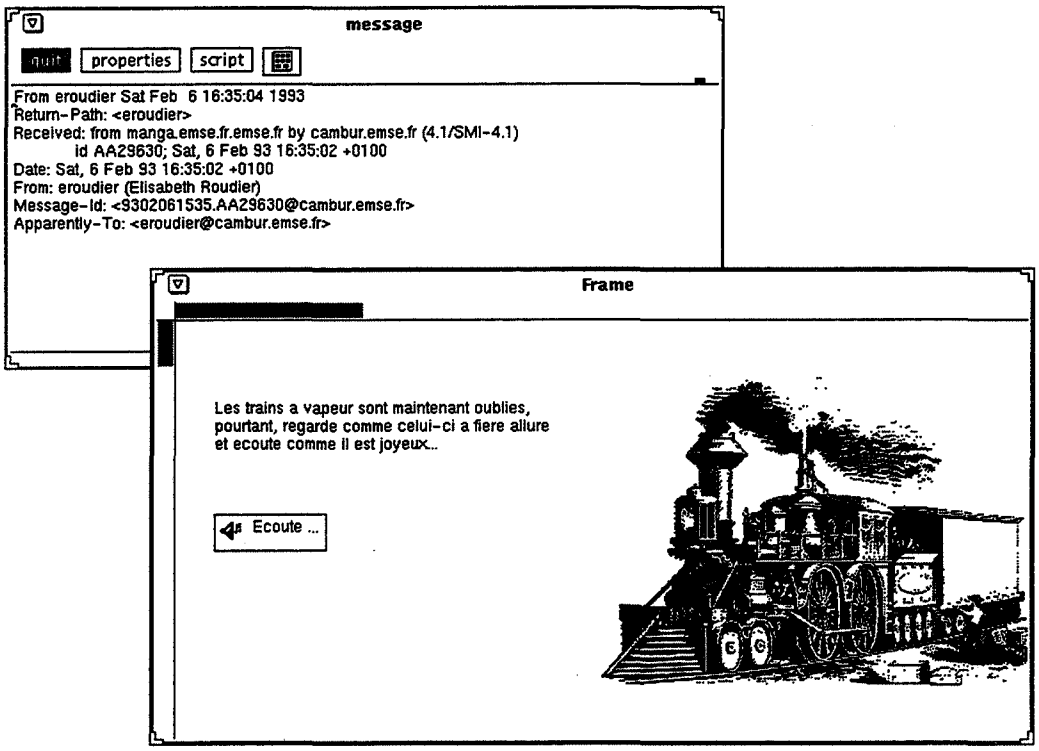


FIGURE 42 *: coordination temporelle*

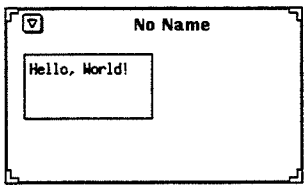


FIGURE 43 *: message Xlib*

3.5. Exemple récapitulatif

Ce dernier exemple est un script plus complexe pour illustrer la plupart des fonctionnalités possibles. Ce message multimédia explique quelques caractéristiques de fonctionnement de l'outil 4M en guidant l'utilisateur dans sa démarche par des interactions et des séquences sonores explicatives et en permettant de visualiser un

message déjà composé. Ce script manipule plusieurs fenêtres, tous les types d'objets possibles et traite des interactions utilisateur à travers des boutons et événements. Pour ne pas surcharger le script, nous nous sommes limités à la composition et la restitution d'un message très simple.

```
(setq Feng (window 50 50))                                    1
(setq Fenc (window 200 200))                                  2
(setq Fenr (window 200 200))                                  3
(setq Fens (window 200 200))                                  4
(setq Tg (text "DEMO/4m.txt"))                                5
(setq Tcp (text "DEMO/mcp.txt"))                              6
(setq Tr (text "DEMO/mrm.txt"))                               7
(setq Ts (text "DEMO/sct.txt"))                               8
(setq Son (sound "DEMO/pp.au"))                               9
(setq Scp (sound "DEMO/sp.au"))                              10
(setq Ssc (sound "DEMO/sc.au"))                              11
(setq Imcp (picture "DEMO/icp.rs"))                          12
(setq Imrs (picture "DEMO/ires.rs"))                         13
(setq Imsc (picture "DEMO/isc.rs"))                          14
(setq Imtp (picture "DEMO/itp.rs"))                          15
(setq Btncp (button "composition"))                          16
(setq Btnrs (button "restitution"))                          17
(setq Btnsi (button "suite"))                                18
(setq Wrn (warning "explications"))                          19
(action Btncp "(compo)")                                     20
(action Btnrs "(rest)")                                      21
(action Btnsi "(suite)")                                     21
(action Wrn "(play Sound 30)")                               23
(de affich (Widget Wind x y)                                 24
 (fontname "9x15bold")                                       25
 (overlap 1)                                                 26
 (show Widget Wind x y)                                      27
 (reset)                                                     28
)                                                            29
(de rest ()                                                  30
 (xmap Fenr)                                                 31
 (show Tr Fenr 0 10)                                         32
 (show Imrs Fenr 0 0)                                        33
 (inactive Btnrs)                                            34
)                                                            35
```

```
(de vue (Bool)                                                    36
 (overlap 1)                                                      37
 (if (Bool)                                                       38
 (progn (erase Imsc Fens) (show Imtp Fens 1 55))                  39
 (progn (erase Imtp Fens) (show Imsc Fens 1 55))                  40
 )                                                                41
 (setq Bool (not Bool))                                           42
 (overlap 0)                                                      43
)                                                                 44
(de suite ()                                                      45
 (xmap Fens)                                                      46
 (setq Sound Ssc)                                                 47
 (par                                                             48
 (show Ts Fens 0 10) (show Imsc Fens 1 55) (play Sound 20)49
 )                                                                50
 (inactive Btnsi)                                                 51
 (affich Wrn Fens 120 10)                                         52
 (evt Fens 2 "(vue t)")                                           53
)                                                                 54
(de compo ()                                                      55
 (xmap Fenc)                                                      56
 (setq Sound Scp)                                                 57
 (par                                                             58
 (show Tcp Fenc 0 10) (show Imcp Fenc 0 0) (play Sound 20)59
                                                                  60
 (inactive Btncp)                                                 61
 (affich Btnrs Fenc 120 10)                                       62
 (affich Wrn Fenc 120 20)                                         63
)                                                                 64
(xmap Feng                                                        65
(show Tg Feng 5 10)                                               66
(play Son 20)                                                     67
(affich Btncp Feng 150 10)                                        68
(affich Btnrs Feng 150 20)                                        69
```

Les lignes de 1 à 19 sont des définitions d'objets que l'on a regroupées au début du script pour en faciliter la lecture. On définit ainsi quatre fenêtres qui vont correspondre à diverses étapes du fonctionnement expliqué (présentation générale du message, composition et restitution d'un message). Chacune comportera un paragraphe de texte et, selon les cas une explication sonore et une image de l'outil ou du résultat correspondant à

l'étape. On crée divers boutons nécessaires à l'enchaînement du message ; deux d'entre eux sont chargés respectivement de lancer les démonstrations de la composition et de la restitution d'un message, l'un est utilisé pour ouvrir la composition du script, le dernier permet d'écouter de nouveau les diverses séquences sonores. Toutes ces actions sont associées dans les lignes 20 à 23. Viennent ensuite la définition des fonctions déclenchées par les interactions utilisateurs ou comprises dans d'autres fonctions. La première affiche un widget (bouton ou avertisseur) dans une fenêtre avec une certaine police. La fonction suivante visualise dans une fenêtre un message reçu avec un texte explicatif et inactive le bouton responsable de son lancement (étape de restitution). Les trois autres fonctions correspondent à l'explication de la composition d'un message. La fonction `vue` permet de basculer l'affichage de l'étape de composition de script dans l'espace et dans le temps, en appuyant sur une touche quelconque (définition d'un événement associé à la fenêtre). La fonction suite correspond aux explications de la composition de script avec des explications sonores et textuelles et l'affichage de l'étape de composition dans l'espace par défaut. La fonction suivante explique la composition générale d'un message avec image, texte et son à l'appui. Les dernières lignes correspondent à l'affichage de la première fenêtre du message avec un texte, une séquence sonore et les deux boutons permettant de continuer. Le résultat de ce message est donné par la figure suivante (cf FIGURE 44).

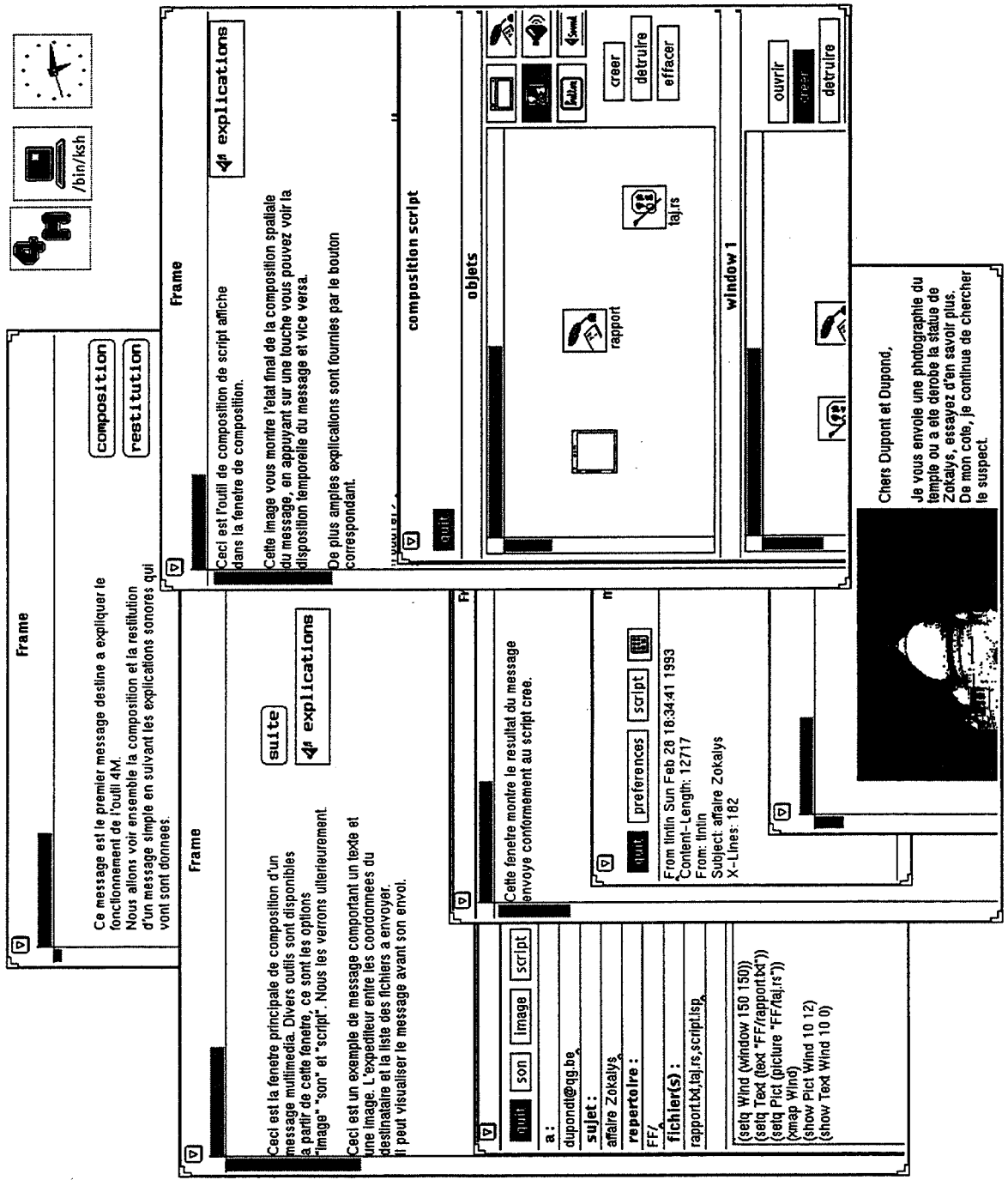


FIGURE 44 : *explications de l'outil*

4. EXTENSIONS POSSIBLES

Nous venons de présenter, dans cette dernière partie, une réalisation de messagerie multimédia basée sur le modèle développé précédemment. L'originalité de l'outil est de formaliser la présentation et le comportement des messages dans un script transmis avec le message. Divers modules graphiques sont fournis à l'utilisateur pour réaliser ces messages. Nous avons aussi montré au travers d'exemples simples quelques possibilités apportées par l'ajout de scripts dans des messages multimédias.

De nombreuses évolutions sont envisageables pour étendre les possibilités de cet outil. On peut les classer en deux catégories : les extensions sur les scripts des messages et celles liées à l'intégration de l'outil dans l'environnement actuel de la messagerie.

Les premières extensions portent sur les scripts eux-mêmes. Nous avons défini un certain nombre de fonctions de base pour la présentation des messages mais il est possible de concevoir très facilement de nouvelles fonctions pour prendre en compte d'autres aspects de présentation spatiale ou temporelle. Par exemple, des transitions par des effets cinématographiques (zoom, rétrécissement, baisse de volume, fondu enchaîné, etc.) entre les différentes séquences du message. Un autre aspect plus intéressant à développer est le comportement intrinsèque des messages. Pour l'instant, seules des possibilités de renvoi automatique de message sont prises en compte. Il serait intéressant de pouvoir définir des fonctions réalisant des questions/réponses ou gardant l'historique du message pour consultations et mises à jour.

Une deuxième perspective à envisager est l'intégration du système dans un environnement plus général et lui permettre d'échanger des messages avec d'autres systèmes de messagerie. Une solution est d'incorporer des caractéristiques du protocole MIME dans l'outil 4M. Deux approches sont possibles : rendre automatiquement accessible l'outil 4M pour des messages 4M arrivants ou, mieux, représenter les messages 4M au format MIME. Les deux approches sont facilement réalisables.

Pour la première, il est possible d'encapsuler un message 4M entièrement au format MIME en définissant un type MIME pour cela (par exemple, `application-4M`). En effet, n'importe quelle application qui connaît ce qui doit être transmis et comment le représenter par une séquence d'octets peut avoir un nouveau body-part enregistré par l'IANA qui lui assure un identificateur unique global pour ce contenu. Il suffit ensuite de spécifier dans un fichier de configuration (répondant aux critères RFC 1343) quel est l'outil à appeler (en l'occurrence 4M) pour traiter les messages 4M.

La seconde approche nécessite de formatter les messages 4M au format MIME. Ceci permettrait à un utilisateur n'ayant pas l'outil 4M mais reconnaissant le format MIME, de pouvoir récupérer les informations véhiculées même s'il ne pourra pas exécuter le script transmis. On peut ainsi envisager une architecture hybride, capable de reconnaître un certain nombre de types de contenus et de les restituer par l'outil 4M. Quand un type inconnu est rencontré, l'application consulte un fichier externe de configuration l'informant sur l'outil correspondant à lancer pour récupérer ce body-part. Pour garder la notion de document du message multimédia, c'est à dire les relations complexes entre les types de médias, une solution est de sauvegarder les diverses parties du message sous une

forme formattée plutôt que de les restituer directement et de déclencher ensuite cette restitution par le script. Ce type d'architecture permet d'utiliser les possibilités de l'interface pour la plupart des types communs et utiliser la souplesse d'un outil de configuration pour les types non reconnus. Pour reconnaître un nouveau type de données, il suffit de rajouter une ligne dans le fichier de configuration (exemple : foo; display foo). Dans tous les cas, il serait sans doute intéressant d'accepter à la fois le format propre à 4M et le format MIME et de générer celui que l'utilisateur préfère.

CONCLUSION

Dans cette thèse, a été présenté un agent utilisateur de messagerie multimédia. Au commencement de l'étude (1990), le domaine de la messagerie multimédia était encore neuf et restreint à quelques systèmes expérimentaux peu utilisés. En effet, la plupart se heurtaient au problème de compatibilité entre applications. L'approche préconisée était d'essayer de réaliser des systèmes pratiques en attendant que des modèles d'application et des standards de représentation apparaissent. Notre vision s'est donc orientée dans cette direction en proposant un outil simple prenant en compte les besoins utilisateurs sans s'attacher aux aspects de formats de données ou de transmission. A l'heure actuelle, des progrès ont été faits pour formaliser et représenter des données multimédias et de plus en plus, de nouveaux systèmes se développent pour faire façe aux besoins croissants des utilisateurs. Néanmoins, il n'existe toujours pas de formalisme définissant ce que doit contenir une application multimédia, comment doit être représentée l'information à l'utilisateur ou quelles sont les propriétés que doivent remplir les interfaces multimédias. Aussi, il semble correct de penser que l'approche du départ, élargir le champ des applications au multimédia sans pénaliser les utilisateurs, est toujours actuelle ; le développement des agents utilisateurs autour du protocole MIME en est d'ailleurs la preuve.

L'originalité de notre réalisation est de s'appuyer sur des systèmes de messagers. L'objectif principal est de représenter la disposition spatio-temporelle des composantes et le comportement dynamique de messages multimédias actifs. Le contrôle du déroulement et de l'organisation d'un message ainsi que les interactions avec l'utilisateur sont assurés par un système d'objets actifs coopérants s'exécutant sur la machine réceptrice. Le comportement de ces acteurs est programmé dans un langage d'extension de type Lisp et transmis avec le message sous forme de messagers. Cette approche par acteurs apporte la souplesse et la puissance nécessaires aux spécificités des messages multimédias.

Au vu de cette réalisation, plusieurs remarques s'imposent.

Tout d'abord, l'outil présenté apparait comme un UA, d'utilisation et de conception simples. L'utilisateur crée ses messages et les met en forme au moyen des divers outils graphiques qui lui sont proposés. La composition ou la restitution des messages est rapide et ne demande pas un investissement important. Seule, la création de scripts complexes nécessite quelques notions de Lisp, bien que des utilitaires et de nombreuses fonctions simplifiées soient fournies.

Outre sa simplicité, ce système est ouvert à d'autres spécifications et peut être facilement intégré dans d'autres applications. L'utilisation des scripts dans les messages permet facilement de leur incorporer de nouvelles spécificités.

Un autre point très important est la légèreté du système. En effet, l'ensemble des exécutables représentent moins de 1 Mega octets et peut être très rapidement installé. Ceci est à comparer avec des systèmes comme Andrew qui certes, plus complexe, représente néanmoins plus de 150 Mega au total, et des heures de compilation pour les générer.

Cette réalisation a montré que le modèle choisi est particulièrement bien adapté aux besoins des systèmes de messagerie. Elle remplit sans aucun doute les objectifs que nous nous étions fixés, à savoir : d'une part, la représentation de l'architecture spatio-

temporelle, le comportement et les aspects interactifs des messages multimédias et d'autre part, la compatibilité, la souplesse et la facilité d'utilisation du système.

Enfin, cette réalisation nous a permis de préciser les concepts d'un modèle de communication général visant à répondre aux besoins divers de communications entre applications quelconques dans des environnements hétérogènes. Ce modèle a pour principe la circulation d'agents actifs sur les réseaux et sera amené à être utilisé dans des contextes autres que la messagerie mutlimédia.

Les travaux futurs vont s'orienter ainsi dans plusieurs directions.

L'outil présenté a été conçu comme un prototype simple servant d'illustration. Un certain nombre d'extensions sont maintenant prévues. Ces extensions devraient porter en particulier, sur l'amélioration des scripts pour manipuler des messages complexes et leur circulation entre des applications. Des efforts sont aussi envisagés au niveau de l'interface utilisateur et au niveau des formats de données et de transmission ; les premiers dans le but de gérer des données plus variées, les seconds pour se conformer à d'autres systèmes de messagerie.

Une première réalisation a servi à la fois de base et d'illustration du modèle proposé. De nouvelles applications de communications spécifiques s'appuyant sur ce modèle sont envisagées. A l'heure actuelle, une réalisation concrète reprenant les concepts théoriques du modèle est à l'étude. Il s'agit d'implanter un système complet de messagers avec des acteurs dialoguant par des messages. Ces acteurs doivent réaliser des fonctions demandées par un utilisateur au travers d'un langage de commande. Dans le domaine de la messagerie, une autre réalisation plus appliquée est de formaliser par des acteurs les fonctions d'un agent de transfert de message.

ANNEXE A

SWAMP : Acteurs en Scheme

Avant-propos

Cette annexe présente SWAMP une *extension à Acteurs* du langage Scheme. Elle est extraite d'un document interne intitulé *SWAMP : acteurs en Scheme,* E. Ahronovitz, J.J. Girardot, E. Roudier[1].

1. INTRODUCTION

La réflexion autour du modèle de communication utilisé dans 4M nous a fait prendre conscience de la nécessité de doter le langage d'extension de processus autonomes, capables de communiquer entre eux et de s'exécuter en parallèle, éventuellement sur des machines distinctes et distantes.

Ce document décrit SWAMP (*Scheme With Actors and Multi Processing*), l'extension du langage GLISP, ainsi que son implantation.

2. MOTIVATION ET CONCEPTS

Toute extension d'un langage (et en particulier, d'un langage bien conçu et bien spécifié tel que Scheme) doit répondre à un besoin bien défini, et respecter des contraintes qu'il est important de spécifier au départ. C'est ce que nous ferons dans cette première partie.

2.1. Description

L'analyse du modèle d'un message multimédia, et du modèle sous-jacent de communication, a fait apparaître des besoins bien précis auxquels ne répondait pas entièrement le langage d'implantation choisi, GLISP.

Le plus immédiat de ces besoins était la nécessité de disposer, au sein même du langage, d'un mécanisme permettant la gestion de processus légers. Ce besoin était net, aussi bien dans le cadre de la restitution d'un message multimédia (puisque des actions de natures différentes étaient à réaliser en parallèle, et de manière asynchrone), que pour la mise en place de la couche de transport, afin de permettre à plusieurs processus coopérant de s'exécuter de manière asynchrone. Dans ce dernier cas, il était également nécessaire de permettre à des processus s'exécutant dans des environnements distincts, ou des machines distinctes (et distantes) de communiquer entre eux.

Ces activités coopérantes nécessitaient d'autre part de permettre à ces tâches de s'exécuter chacune dans son environnement propre. Enfin, la nature même de ces tâches ne pouvait être déterminée entièrement à la conception du système. Il était clair que certaines spécifications (par exemple les opérations réalisant l'interface avec des applications non encore connues) ne pouvaient être déterminées qu'au coup par coup,

1. Ecole des Mines de Saint-Etienne.

suivant les besoins spécifiques à chaque site particulier. Il était ainsi nécessaire d'offrir la possibilité de créer dynamiquement de telles tâches, non seulement selon des modèles spécifiés lors de la conception du système, mais aussi selon des modèles dont la description ne serait connue qu'après la mise en oeuvre effective sur ces sites particuliers.

Ces différentes caractéristiques suggéraient donc fortement la mise en place d'une infrastructure largement inspirée des caractéristiques des langages à acteurs, infrastructure dont la description est proposée ici. Cependant, cette extension doit naturellement tenir compte des caractéristiques du langage de base, ici Scheme, et les respecter autant que faire se pouvait. Il était nécessaire par exemple, de respecter la syntaxe et la sémantique du langage, mais peut être plus encore, de conserver les concepts et la philosophie minimaliste de celui-ci. Il était désirable également d'obtenir un outil efficace sur le plan des performances, de l'encombrement de la mémoire, et de la facilité de la programmation. Les différentes décisions qui ont été prises reflètent donc ces diverses contraintes.

2.2. Entités et Concepts

Comme tout langage de programmation, Scheme fait appel à des *entités* et des *concepts*. Les concepts interviennent dans la description de la sémantique du langage : types de données, nature d'une liaison ou d'une application fonctionnelle, mécanisme général de l'exécution, etc. Les entités sont les éléments qui interviennent effectivement lors d'une mise en oeuvre d'un programme écrit dans ce langage particulier : nombres, caractères, variables, fonctions, etc.

En Scheme, et contrairement à ce qui se passe dans la majorité des langages de programmation, il y a *réification* de la plupart des concepts du langage sous la forme d'entités effectivement manipulables par des programmes. C'est le cas bien sûr des données (nombres, caractères, chaînes, tableaux, tables, listes, etc), mais aussi des *fonctions*, des *ports d'entrées-sorties*, des *erreurs*, des *environnements*, des *continuations*, etc. Ces entités en outre, sont de *première classe* : elles peuvent être conservées dans des structures, passées en paramètres à des fonctions, fournies comme résultats.

2.2.1. Interprétation d'une expression Scheme

Le mécanisme d'évaluation d'une expression Scheme est bien connu. Certains éléments du langage représentent des *constantes*, et l'interprétation d'un tel élément va fournir comme résultat la valeur de cette constante. Un *symbole* représente une variable, et l'interprétation de ce symbole va fournir la valeur associée à cette variable. Une *liste*, enfin, est composée d'éléments qui vont être évalués séparément selon l'algorithme ici décrit, le premier élément de la liste étant ensuite appliqué, en tant que fonction, sur les autres éléments, fournissant la valeur finale de l'expression.

Un aspect de ce mécanisme nous intéresse particulièrement : c'est la *résolution* des variables, qui va constituer l'un des points clefs de notre réflexion.

2.2.2. Liaison

Toute utilisation d'un *identificateur* nécessite donc de la part du système un travail *d'association* de cet identificateur à l'une des liaisons visibles de l'environnement courant.

Un environnement est assimilable, en première approximation, à un ensemble de *liaisons*, une liaison étant un couple *(symbole, référence)* (nous dirons que le *symbole* correspondant *désigne* cette liaison particulière). Dans un tel ensemble, tous les *symboles* des liaisons sont distincts les uns des autres, et les *références* sont indéterminées ou désignent une valeur quelconque.

Trois *formes du langage* manipulent ces liaisons :

- la définition :

(`define` <*nom*> <*valeur*>) qui permet la *création* d'une nouvelle liaison désignée par le symbole <*nom*>, en lui associant éventuellement une valeur initiale.

- la référence :

<*nom*> qui fournit la valeur associée à <nom> dans l'environnement courant, si elle existe (et provoque une erreur si la liaison n'existe pas).

- la modification :

(`set!` <*nom*> <*valeur*>) qui *modifie* la liaison associée à <*nom*> dans l'environnement courant, en associant <*valeur*> au champ *référence* de la liaison (là également, la liaison doit exister).

2.2.3. Environnement

La distinction entre les formes `set!` et `define` ne serait pas si fondamentale, si un environnement était un objet aussi simple que décrit ci-dessus. GLISP, tout comme nombre d'autres systèmes Scheme, propose une vision étendue de la notion d'environnement, en gérant des *environnements* emboîtés, constitués d'une succession d'*espaces* de noms, qui eux-mêmes sont des ensembles de liaisons (cf FIGURE 45).

L'environnement `E` est ici constitué d'une liste de trois espaces de noms, `A`, `B` et `C`. Dans un espace de noms, les symboles associés aux liaisons présentes sont tous distincts, mais des liaisons situées dans des espaces de noms différents peuvent être associées au même symbole.

La présence de plusieurs espaces de noms nous impose de mieux spécifier la sémantique des diverses formes. Le fait qu'un environnement soit constitué d'une *liste* d'espaces de noms impose un ordre total dans la recherche d'une liaison : dans notre exemple, `A` est exploré avant `B`, qui est lui-même exploré avant `C`. La *recherche* d'une liaison (cas de la forme `set!` ou de la *référence*) s'exécute donc par exploration des éléments successifs de la liste constituant l'environnement. La première liaison correspondant au symbole recherché est sélectionnée. La forme `define` au contraire va créer une liaison dans le premier espace de noms rencontré, celui qui est en tête de la liste.

Notons enfin qu'une même liaison peut être connue de plusieurs espaces de noms. Ce mécanisme, nous le verrons, nous permettra de résoudre partiellement certains conflits de noms.

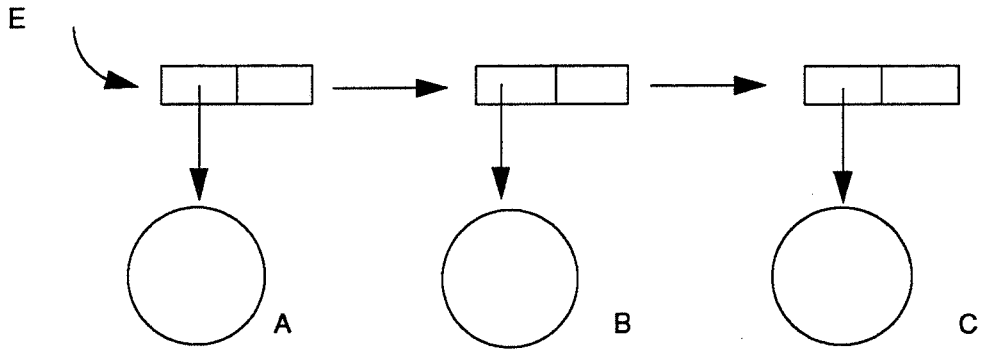


FIGURE 45 : *environnement*

2.2.4. Résolution des liaisons

Le dernier point important à préciser est celui de la *résolution des liaisons* : quand, et comment, les liaisons sont-elles résolues ?

Scheme utilise un modèle de *liaison statique*, c'est à dire que les liaisons sont résolues au moment de la définition d'une fonction, dans l'environnement de cette définition, et non lors de l'exécution ultérieure de cette fonction. Notons que le cas d'une expression <*exp*> introduite en mode terminal, se ramène au cas `((lambda () <exp>))`, c'est à dire à celui d'une fonction anonyme définie et immédiatement exécutée.

3. CONCEPTION D'UN MODELE ACTEUR

Un acteur est un objet de première classe du langage. Il est le représentant d'un concept nouveau, qui s'ajoute à ceux qui existent déjà dans le langage, mais ne remplace ou ne modifie pas des concepts existants.

Nous définissons un *acteur* en SWAMP comme l'association d'un *environnement* et d'une *fonction*. L'environnement décrit l'état et les propriétés de l'acteur, la fonction correspond à la notion de *script* des modèles de Hewitt ou Agha. Cette fonction ne fait pas nécessairement partie de l'environnement de l'acteur, mais s'exécute dans celui-ci.

3.1. Communication avec un acteur

La communication avec un acteur est réalisée par une fonction, dont la syntaxe est la suivante :

`(send` <*acteur*> <*message*> <*promesse*>`)`

L'envoi d'un message à un acteur est une opération *synchrone*, dont le résultat est une *promesse* (équivalente à une variable *future* de ABCL/1). Le troisième paramètre est une promesse optionnelle. Si ce paramètre est précisé, le receveur est chargé de tenir cette promesse qui avait été initialement confiée à l'envoyeur du message. Si le troisième paramètre n'est pas spécifié, le système construira une nouvelle promesse, en la transmettant à *<acteur>*. Dans tous les cas, le résultat de `send` sera la promesse elle-même.

L'exécution du processus envoyeur du message se poursuit immédiatement. Le message destiné à l'acteur est mise en attente dans une file. Il sera délivré à l'acteur dès que celui-ci sera activable. Un acteur ne reçoit effectivement un message que s'il est disponible, c'est à dire s'il n'est pas *actif* ou en *attente* d'un événement.

En résumé, la primitive `send` :

- crée une *promesse* si aucune n'est spécifiée dans la forme d'appel,
- ajoute le triplet {*<acteur>* *<message>* *<promesse>*} dans la liste des activations en attente,
- rend comme résultat la promesse.

3.1.1. Promesse

Une *promesse* est un objet de première classe. Elle contient une *valeur*, un *drapeau*, indiquant si la promesse a été tenue ou non, et enfin une liste de *processus en* attente de la disponibilité de la valeur. Les opérations de base sur les promesses sont les suivantes :

- Création :

`(make-promise)` qui fournit une promesse,

- Renseignement :

(`ready?` *<promesse>*) qui indique si la promesse est disponible ou non,

- Obligation :

(`force` *<promesse>*) qui attend la disponibilité de la valeur associée à la promesse, en forçant éventuellement le calcul de celle-ci,

- Délai :

(`delay` *<expression>*) qui crée une promesse en lui associant une expression ; cette expression n'est pas immédiatement exécutée, et ne le sera que par une opération `force`.

- Attente multiple :

(`wait` *<p1>* *<p2>*... *<pn>*) qui permet de forcer l'attente sur un ensemble de promesses. Le résultat est la première (ou l'une parmi les premières) disponibles.

- Affectation :

(`set!` (`force` *<promesse>*) *<valeur>*) permet d'affecter une valeur à une promesse : la promesse devient disponible, et les processus en attente sur cette promesses sont réactivés.

Remarque : peut-être une vision plus fonctionnelle serait-elle préférable, par exemple : (*<promesse>* *<valeur>*) ?

On notera que les formes `delay` et `force`, qui correspondent à des opérations standard en Scheme, permettent un fonctionnement synchrone et paresseux.

3.2. Exécution d'un acteur

L'exécution d'un acteur ne peut être déclenchée que lorsque cet acteur est *disponible*, c'est à dire après sa création, ou lorsqu'il a terminé le traitement d'un message antérieur. Un acteur est incarné par une fonction à deux paramètres : le message transmis, et une promesse à tenir. Typiquement, l'acteur affectera à cette promesse le résultat de sa tâche.

L'exécution d'un acteur se termine lorsque la fonction qui a reçu le message se termine, ou encore lorsque cette fonction, ou une fonction appelée par cette dernière, exécute un appel à `exit`. L'acteur redevient ainsi disponible pour le traitement d'un nouveau message.

Notons qu'un acteur peut achever son travail sans affecter de valeur à la promesse qui lui a été confiée. Une attente sur cette promesse risque bien de se poursuivre *ad vitam eternam...*

3.3. Formes de communications entre acteurs

Montrons que ces primitives suffisent à réaliser les opérations traditionnelles de communication que l'on attend des acteurs.

3.3.1. Communication Asynchrone

Il s'agit de l'envoi, par un acteur initiateur *I*, d'un message *M* à un acteur *A*. L'acteur *I* n'attend pas de réponse. Cette opération s'écrit simplement :

```
(send A M)
```

L'acteur *I* ignore la promesse rendue par `send` et poursuit son exécution.

3.3.2. Communication Synchrone

L'acteur initiateur *I* envoie un message au destinataire *A*, et attend que la réponse soit disponible pour poursuivre son exécution. L'opération s'écrit :

```
(force (send A M))
```

L'utilisation de `force` assure ici que *I* restera bloqué jusqu'à l'arrivée de la réponse de *A*. Celui-ci a reçu le message *M* et la promesse *P*, et transmettra son résultat *V* par :

```
(set! (force P) V)
```

ce qui débloquera l'acteur *I* en attente.

3.3.3. Communication Anticipée

L'acteur initiateur *I* envoie un message au destinataire *A*, et conserve la promesse :

```
(set! R (send A M))
```

Il peut ultérieurement tester la fin d'exécution du travail de l'acteur *A* par :

```
(ready? R)
```

dont le résultat n'est pas bloquant, exécuter une autre partie de sa tâche, tester à nouveau la promesse, etc. Comme dans les autres cas, l'acteur *A* signalera l'achèvement de son travail en affectant une valeur à la promesse qu'il a reçue en paramètre.

3.3.4. Sous-traitance

La sous-traitance consiste, pour un acteur, à confier tout ou partie de la tâche décrite par un message à un autre acteur. Cette forme de travail coopérative est possible en SWAMP, et peut être combinée avec les communications synchrones, asynchrones ou anticipées. L'acteur *I* va par exemple envoyer un message à un premier acteur *A* en conservant la promesse liée à cet envoi :

```
(set! R (send A M))
```

L'acteur *A*, après analyse du message *M*, va décider de confier le travail à un acteur *B*. Il va lui transmettre la requête *et* la promesse qu'il a reçue :

```
(send B M R)
```

Son travail est alors terminé, il peut effectuer une autre tâche, ou s'interrompre. L'acteur *B* achèvera quant à lui son travail en fournissant le résultat par affectation à la promesse, résultat qui sera aussitôt accessible à l'acteur initial *I* :

```
(set! (force P) V)
```

Notons ici la différence entre sous-traitance et *délégation* dans les langages acteurs traditionnels. Dans la délégation, l'acteur B réalise des opérations pour le compte de l'acteur A en travaillant *dans l'espace privé de l'acteur A*. Ce n'est pas le cas ici, et les acteurs A et B sont totalement indépendants.

3.4. Vie et mort d'un acteur

Comme toute entité du langage, un acteur est doté d'une visibilité lexicale et d'une persistance illimitée (ce qui veut dire qu'il ne sera effectivement détruit que lorsqu'il n'existera plus aucune référence à lui). Une première fonction permet la création d'un acteur :

`(make-actor` *<fonction>* *<environnement>*`)`

Elle associe deux entités existantes, une fonction et un environnement, pour créer un nouvel acteur, qui est fourni comme résultat de la fonction. Cette fonction sera, la plupart du temps, une fonction définie en Scheme, mais certaines fonctions primitives permettront la création d'acteurs élémentaires (assimilables aux *rock-bottom actors* de Hewitt) assurant des services standards, comme des gestions de *timers*, l'accès à des opérations spéciales du système, etc.

Un acteur peut aussi être créé comme *copie* d'un acteur existant, par :

`(clone` *<acteur>*`)`

Cette opération crée un nouvel acteur, physiquement différent du paramètre, mais qui partage la même fonction et le même environnement.

Enfin, un acteur peut être construit par *composition* de diverses *facettes* élémentaires, point sur lequel nous reviendrons par la suite.

3.4.1. Un exemple

Voici un exemple montrant un acteur sachant calculer un élément de la suite de Fibonacci :

```
(define fib (make-actor
     (lambda (n P)
          (if (< n 2)
               (set! (force P) 1)
               (send (clone somme)
                    (list (send (clone fib) (- n 1))
                         (send (clone fib) (- n 2)))
                    P)))
     standard-environment))
(define somme (make-actor
     lambda ( (x y) P)
          (set! (force P) (+ (force x) (force y)))
     standard-environment))
```

Cet acteur `fib` fait appel à une accointance, `somme`, pour le calcul effectif de la somme des résultats des sous-appels. Les deux acteurs sont créés dans l'environnement standard.

Notons l'utilisation de la primitive de duplication d'un acteur pour créer des *clones* des acteurs initiaux (l'envoi des messages à l'acteur `somme` lui-même, et non à une copie nouvelle, provoquerait un blocage du système). L'utilisation d'un clone de `fib` permet de rendre ce dernier immédiatement disponible pour le traitement d'un nouveau message.

Le calcul de la valeur du dixième élément de la suite de Fibonacci s'écrit alors :

```
(force (send fib 10))
```

Cet exemple montre la mise en oeuvre de la plupart des notions du modèle : création statique et dynamique d'acteurs, messages synchrones et asynchrones, sous-traitance.

4. EXTENSIONS DU MODELE A DES ACTEURS REPARTIS

Le modèle présenté ci-dessus sous-entend une exécution fortement localisée : tous les objets manipulés par le système sont dans le même espace d'adressage, et peuvent être directement consultés et modifiés par n'importe quel processus léger. Il est cependant possible d'implanter une opération de communication entre sites distants, qui permet l'envoi d'un message (une entité Scheme quelconque, représentée sous une forme transférable) à un acteur situé sur un site distant.

Il est clair que certains mécanismes ne sont plus utilisables lors de la coopération entre deux acteurs distants, par exemple l'utilisation de *promesses* comme outil de synchronisation, ou encore le partage de comportements, opérations qui font implicitement utilisation d'un partage de la mémoire.

Le partage de comportements nécessite la présence, sur un site donné, de l'ensemble des facettes standards. Un comportement spécifique peut également être transmis à un site distant sous la forme du programme source qui permet, par exécution, de le reconstituer.

La synchronisation entre acteurs distants peut, quant à elle, se réaliser par l'utilisation, sur chaque site, d'un acteur relai. L'envoi (virtuel) d'un message avec promesse d'un acteur *A* sur le site *1* à un acteur *B* sur le site *2* peut se réaliser ainsi :

- l'acteur *A* crée un acteur local, *A'*, de type *correspondant vers B*, auquel il envoie le message *M*, et qui lui fournit une promesse *P*. *A* peut utiliser cette promesse à sa guise, et, de manière générale, s'adresser à *A'* comme s'il communiquait directement avec *B*.

- A' demande la création sur le site distant d'un acteur *B'* de type *correspondant vers A*.

- Une fois créé, *B'* envoie un message à *A'* pour l'informer de sa création.

- *A'* envoie alors à *B'* le message *M*.

- *B'* transmet à *B* le message *M*, et obtient en échange une promesse *P'*, sur laquelle il se met en attente.

- Lorsque *B* a achevé le traitement du message M, il satisfait la promesse *P'*, ce qui débloque *B'*.

- *B'* transmet à *A'* le résultat obtenu.

- *A'* satisfait *P* en lui affectant la valeur obtenue, ce qui permet éventuellement à *A* de poursuivre son travail.

On peut imaginer diverses variations sur ce thème, comme l'emploi de techniques permettant de fiabiliser les dialogues entre *A'* et *B'*, l'intervention de *time-out*, etc.

La référence entre entités distantes peut s'obtenir par l'utilisation de *références universelles*, permettant la désignation fiable d'entités existant (ou ayant existé) sur un site distant. Ce mécanisme peut être considéré comme une extension d'un trait du système Scheme du MIT (*object-hashing facility*, [HANS 91]).

5. CONCLUSION

Ce document présente SWAMP, une *extension acteur* du langage Scheme. L'extension est *minimale* et *conservative*. Elle introduit un seul nouveau type d'entité dans le langage, l'*acteur*, et affine certaines autres notions, comme celles d'*environnement* et de *promesse*.

L'extension permet la création d'acteurs, dont le comportement peut être calqué sur celui des modèles classiques ([AGHA 86], par exemple). La collaboration entre des systèmes d'acteurs résidents sur des sites distincts est rendue possible par un mécanisme de transfert d'entités d'un site à un autre.

Le modèle proposé est en cours d'implantation, et une première utilisation devrait permettre la validation d'un modèle de transfert d'informations entre applications situées sur des sites distants.

ANNEXE B

X Window System

1. PRÉSENTATION DE X11

1.1 Introduction

Les systèmes de fenêtrage sont devenus une caractéristique commune de la plupart des systèmes informatiques. Ils fournissent à l'utilisateur une interface conviviale dans laquelle il peut ouvrir des fenêtres, se déplacer a l'intérieur ou appeler des fonctions à l'aide d'objets graphiques. Les fenêtres, icones, menus et autres gadgets font maintenant partie intégrante du dialogue entre l'homme et la machine. Parmi les systèmes de fenêtrage les plus connus, on peut citer l'interface du Macintosh et Microsoft Window. Ces types d'interface ont la particularité d'être des systèmes mono-utilisateur, et sont de plus dépendants de la machine sur laquelle ils sont implantés.

Un système de fenêtrage doit répondre a un certain nombre de besoins. Parmi les plus importants on peut citer la transparence vis a vis du réseau et l'indépendance par rapport aux périphériques. Il doit aussi être implémentable sur n'importe quel type de display et fournir des primitives d'affichage de graphiques, images et textes de haut-niveau.

Dans le monde des stations de travail, les interfaces sont devenues une norme et l'adoption quasi-universelle du système d'exploitation Unix combinée avec le besoin de partager des informations travers un réseau à conduit à une exigence de standardisation.Le système de X Window a été développé dans cette optique. Le but était de permettre à un programme tournant sur un machine de créer des fenêtres sur une machine de constructeur différent et cela de manière transparente vis a vis du réseau.

1.2. Historique

X Window System ou X a été développé par le Massachusetts Institute of Technology (MIT) en collaboration avec Digital Equipement Corporation (DEC). Son nom ainsi que certains concepts dérivent d'un système précédent nommé W originaire de Stanford. X a vu le jour en 1984 dans le cadre du projet Athena. Le but de ce projet était que chaque étudiant sur sa station de travail puisse utiliser des outils locaux et en même temps avoir la possibilité de récupérer et afficher des documents et des graphiques provenant d'autres stations. Les stations utilisées étant de constructeurs différents, la première étape fut de concevoir un protocole indépendant du matériel pour envoyer des graphiques sur le réseau. Ce fut le point de départ du développement de X Window System.

En 1986, la version X10.4 est distribuée et conquiert le monde des stations Unix. MIT forme, en 1988, un consortium avec la plupart des principaux constructeurs pour le développement de X et son adoption comme standard ANSI. Parmi les membres du consortium, on trouve *HP, Apple, AT&T, Sun, IBM* et *DEC*. Un grand nombre d'utilisateurs contribuent au développement et à l'extension de X à travers le monde. La cinquième version de X11 est disponible depuis automne 1991.

2. CARACTERISTIQUES

2.1. modèle client-serveur

X est un système graphique de fenêtrage, multi-tâches, transparent vis a vis du réseau et indépendant des périphériques. Il permet d'afficher plusieurs applications sur le même écran, ou d'avoir plusieurs fenêtres de la même application sur différents écrans. Il fournit des possibilités de tracés graphiques, d'affichage de textes multi-fonts, de gestion de fenêtres imbriquées ou indépendantes.

X est basé sur un modèle client-serveur. Le client est en général une application X c'est à dire un programme qui fait des requêtes d'affichage ou de demande d'informations au serveur. Le serveur est le programme qui gère le "display". On entend par "display" le dispositif d'entrées/sorties constitué du ou des écrans, du clavier et de la souris. Le rôle du serveur est de multiplexer les requêtes du client vers les fenêtres et de démultiplexer les entrées clavier et souris vers les clients appropriés. Il fournit en plus les ressources et mécanismes fondamentaux. Le serveur tourne sur chaque station, le client et le serveur pouvant s'exécuter sur des machines différentes. Le serveur et le client dialoguent au moyen de messages qui sont conformes au protocole X. Le client envoie des requêtes au serveur, le serveur répond par des événements ou des messages d'erreurs. Une requête est une primitive d'affichage ou une demande d'informations sur une ressource. Un événement correspond à un paquet d'informations généré par le serveur quand une certaine action arrive et mis en file d'attente pour emploi ultérieur. Ces actions correspondent généralement à une entrée au clavier ou un mouvement de souris de la part de l'utilisateur. Les communications sont donc bidirectionnelles; elles transitent également à chaque extrémité par des tampons pour réduire le trafic, et sont asynchrones, sauf comportement synchrone explicitement demandé (cf FIGURE 46).

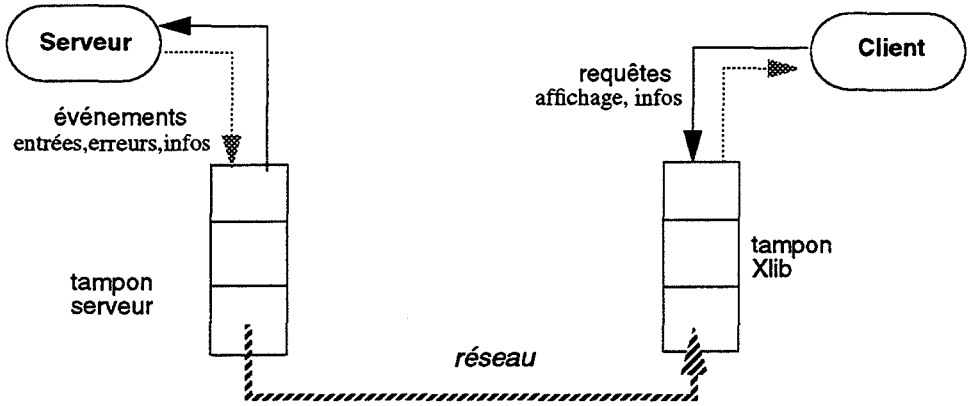


FIGURE 46 : *communications client-serveur*

Un point essentiel est le fait que X est construit sur un protocole conçu pour être indépendant des périphériques et donc de tout système d'exploitation. Bien que conçu sur

Unix, X peut être implanté sur n'importe quel système d'exploitation comme cela a été fait sur VAX/VMS de Digital.

2.2. Architecture

X11 est un modèle en couches (cf FIGURE 47).

Au niveau le plus bas se trouve Xlib, la bibliothèque d'interface avec le système de fenêtrage par laquelle passent toutes les communications entre le serveur et le client. Cette bibliothèque regroupe plus de 200 primitives de plus ou moins haut niveau qui sont ensuite traduites en requêtes au protocole X. Xlib fournit un grand nombre de fonctionnalités: primitives graphiques, définition et utilisation de ressources, gestion des événements... L'ensemble de ces fonctions est suffisant pour concevoir une interface mais dans la plupart des cas il est préférable d'utiliser la couche supérieure que sont les Toolkits.

Un "toolkit" est une boite à outils qui permet d'employer un plus haut niveau d'interface que Xlib. Le nom "toolkit" recouvre en fait deux parties, la première partie comporte ce que l'on appelle les Intrinsics, qui représente la base pour permettre aux programmeurs de créer des interfaces en combinant un ensemble de composants pré-existants. Ces composants, appelés widgets, forment la deuxième partie. Les "widgets" sont des entités graphiques de base telles que les menus, boutons ou ascenseurs. La partie Intrinsics fournit le moyen de les relier ensemble et de les utiliser. Elle est construite sur un mécanisme d'événements tout comme Xlib, mais prend en compte toute la gestion des événements en les distribuant aux widgets appropriés. La programmation avec un toolkit est donc beaucoup plus simple. Par analogie, on peut associer les widgets à des objets et la partie Intrinsics à des routines sur ces objets.

Les Intrinsics peut supporter un grand nombre d'ensemble de widget comme les *Athena widgets*, *Motif* ou *Open Look* widget set.

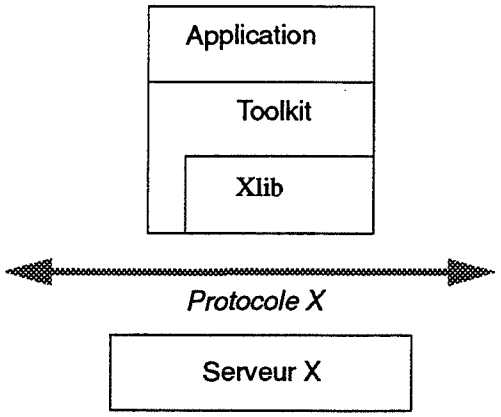


FIGURE 47 *: architecture de X11*

Bibliographie

[ADIE 93] C. Adie, A Survey of Distributed Research, Standards and Products, 1ère édition, Jan 1993 (liste mmm-people@isi.edu).

[AGHA 86] G. Agha, Actors : A Model of Concurrent Computation in Distributed Systems, MIT Press, Cambridge, Mass., 1986.

[AGHA 87] G. Agha, C. Hewitt, Actors : A conceptual Foundation for Concurrent Objects Oriented Programming, Editions B.Shriver et P.Wegner, Research Direction in Object-Oriented Programming, MIT Press, 1987.

[AHRO 93a] E. Ahronovitz, B. Amade, J.J. Girardot, E. Roudier, *Un modèle de communication par des Messagers Acteurs*, EDI & Multimédia, SITEF 1993, Toulouse.

[AHRO 93b] E. Ahronovitz, J.J. Girardot, E. Roudier, *Modèle de Messagers Acteurs*, RPO 93, La Grande-Motte, Juin 1993.

[ARTO 87] X. Artozqui, C. Dusfour, MIDI à l'heure de la norme, Micro Systèmes, Oct 1987.

[BAUD 91a] F. Baude, utilisation du paradigme acteur pour le calcul parallèle, Thèse de doctorat, Orsay, Paris-Sud, 1991.

[BAUD 91b] F. Baude, G. Vidal-Naquet, *Actors as a parallel programing model*, Proc of the VIII Symposium of theorical aspects of Computer Sciences, Lecture Notes in Computer Science, vol 480, p 184-195, 1991.

[BECK 91] B. Beckman, *A Scheme fot little Languages in Interactive Graphics*, Software-practice and experience, vol 21(2), Fev 1991.

[BEGO 90] J.A. Begory, *An introduction to hypermedia issues, systems and applications areas*, International Journal of Man-Machine Studies, vol 33, n° 2, Août 1990.

[BETZ 88] D. Betz, *Embedded Languages*, BYTE. Nov 1988

[BORE 88] N.S. Borenstein, C.F. Everhart, J. Rosenberg, A. Stoller, *Architectural issues in the Andrew Message System*, Message Handling Systems & Distributed Applications, Costa Mesa, Oct 1988.

[BORE 91a] N.S. Borenstein, *Multimedia Electronic Mail: will the Dream Become a Reality ?*, Communications of the ACM, vol 34, n° 4, Avril 1991.

[BORE 91b] N.S. Borenstein, C.A.Tyberg, *Power, ease of use and cooperative work in a pratical multimedia message system*, International Journal of Man-Machine Studies, vol 34, n° 2, Fev 1991.

[BOUD 92] J.Y. Boudec, T*he Asynchronous Transfer Mode : A Tutorial*, Computer & ISDN Systems, vol 24 n° 4, Mai 1992.

[BOUN 92] G. Boundin, *Amiga 4000 : Hight Tech and Multimedia*, SVM, Nov 1992.

[BUHR 92] P.A. Buhr, R.A. Stroobosscher, *The usystem : Providing Light-weight Concurrency on shared-memory Multiprocessor computers running UNIX*, Software-Practice ans Experience, vol 22(9), Sept 1992.

[CFI 91a]] CFI Extension Language Selection Document, CFI Extension Language working Group of The Arhitecture TSC, document n° 87.

[CFI 91b]] CFI Extension Langage : Core Language Selection Version 0.7, CFI Extension Language working Group of The Arhitecture TSC, document n° ARCH-91-G-1.

[CONK 87] J. Conklin, *Hypertext : an introduction and survey*, Computer, vol 20, n° 9, Sept 1987.

[COUT 86] J. Coutaz, La construction d'Interfaces Homme-Machine, Rapport IMAG RR 635-I, Nov 1986

[COUT 90] J. Coutaz, Interface Homme-Ordinateur : conception et réalisation, dunod 1990.

[COUT 91] J. Coutaz, J. Caelen, *A Taxonomy for Multimedia and Multimodal User Interfaces*, Proceedings of the ERCIM Wokshpos, INESC, Lisbonne, Nov 1991.

[CROW 85] T. Crowley, H. Fordsdick, M. Landau, V. Travers, *The Diamond multimedia Editor*, BBN Laboratories, Computer, Dec 1985.

[CROW 90] T. Crowley, H. Lison, *Sight and Sound*, Unix Review, vol 7, n° 10, Oct 1989.

[de MEY 92] V. de Mey, C. Breiteneder, L. Dami, S. Gibbs, D. Tsichritzis, *Visual composition and Multimedia*, Eurographics'92, vol 11, n° 3.

[DAVE 91] G. Davenport, T.A. Smith, N. Pincever, *Cinematic Primitives for Multimedia*, IEEE Computer Graphics & Applications, Juil 1991.

[FOX 91] E.A. Fox, *Advances in Interactive Digital Multimedia Systems*, Computer, Oct 1991.

[GALL 91] D. Le Gall, *MPEG: A Video Compression Standard for multimedia applications*, Communications of ACM, vol 34, n° 4, Avril 1991.

[GARR 90] Y. Garret, *NEXT Nouvelles dimensions*, SVM, Nov 1990.

[GATE 87] B. Gates, *Beyond Macro Processing*, Applications Software Today, BYTE Bonus Edition, Summer 1987.

[GIBB 91] S. Gibbs, *Composite Multimedia and Active Objects*, OOPSLA'91.

[GIRA 91] J.J. Girardot, InTalk, Un langage de Contrôle d'Applications, Juin 1991.

[GIRA 92] J.J. Girardot, GLisp 1.04b, Manuel de Référence, Juin 1992.

[GOLD 91] C.F. Goldfarb, *HyTime: A standard for structured hypermedia interchange*, Computer, Août 1991.

[HANS 91] C. Hanson, MIT Scheme Reference Manual. Technical report, Massachusetts Institute of technology, boston, Mass., 1991.

[HARN 91] K. Harney, M. Keith, G. Lavelle, L.D. Ryan, D.J. Stark, *The i750 Processor: A ToTal Multimedia Solution, Communications of ACM*, vol 34, n° 4, Avril 1991.

[HAYE 90] F. Hayes, *EDI : The Business Connection*, UnixWorld, 1990.

[HAYE 92] F. Hayes, *Rivals Compete for E-mail Standards*, UnixWorld, Juil 1992.

[HERN 93] J. Hernández, P de Miguel, M. Barrena, J.M. Martínez, A. Polo, M. Nieto, ALBA : A Parallel Langage Based on Actors, ACM Sigplan Notices, vol 28, n° 4, Avril 1993.

[HEWI 77] C. Hewitt, *Viewing control as patterns of pasing messages*, Artificial Intelligence, vol 8, n° 3, p 323-364, 1977.

[HOEP 91] P. Hoepner, *Synchronizing the Presentation of Multimedia Objects - ODA Extensions*, SIGOIS, Juillet 1991.

[HORR 86] J.H. Horris, M. Satynarayanan, M. Conner, J. Howard, D. Rosenthal, D. Smith, *Andrew: A distributed personal computing environment*, Communications of the ACM, vol 29, n° 3, Mars 1986.

[HUIT 89] C. Huitema, *The Challenge of Multimedia Mail*, Computer Networks & ISDN Systems, vol 17, 1989.

[JONE 74] P. Jones, The technique of the Television Cameraman, Focal Press, 1974.

[KAY 79] A.C. Kay, *Programming your own computer*, Science Year, 1979.

[KIM 91] Y. Kim, *Chips deliver multimedia*, BYTE, Dec 1991

[KREU 91] C. M.A. Kreuwels, *EDI Implementation : Practical suggestions based on Experience*. 3rd International Congress of EDI Users, pages 310-321, Sept 1991.

[KRIE 92] Ph. Krief, Utilisation des langages objets pour le prototypage, Etudes et Recherches en Informatique, editions Masson, 1992.

[KUO 83] F.F. Kuo et panélistes, Panel on Multimedia Computer Mail - Technical issues and future standards, ACM, 1983.

[LAUR 90] B. Laurel, T. Oren, A. Don, *Issues in multimedia interface design: media integration and interface agents*, CHI'90 Proceedings, Avril 1990.

[LESU 90] L. Lesur, *Multimédia : vers de nouvelles applications*, Soft & Micro, Sept 90.

[LEE 91] K. Lee, *Behind Curtain X*, Unix Review, vol 9, n° 6, 1991.

[LIEB 86] H. Lieberman, *Using Prototypal Objects to Implement Shared Behavior in Object Oriented Systems*, OOPSLA'86 proceedings, vol 21, p 241-223, ACM Press, Nov 1986.

[LIEB 91] M. Liebhold, E.M. Hoffert, *Toward an open Environment for Digital video*, Communications of the ACM, vol 34, n° 4, Avril 1991.

[LIPP 90] R. Lippincott, *Beyond Hype*, Byte, Fev 1990.

[LITT 90] T. D.C. Little, A. Ghafoor, *Network Considerations for Distributed Multimedia Object Composition and Communication*, IEEE Network, vol4, n° 6, Nov 90.

[LITT 91] T. D.C. Little, *A. Ghafoor, Spatio-Temporal composition of Distributed Multimedia Objects for Value-Added Networks*, Computer Oct 1991.

[MacA 90] MacApp 2.0. Tutorial, MacApp 2.0 Cookbook, MacApp 2.0.1, Apple Computer, Inc, 1990.

[MAGR 90] C. Magrin, Les cahiers d'Hypercard, MacWorld, 1990

[MASI 90] G. Masini, A. Napoli, D. Colnet, D. Leonard, K. Tombre, Les langages à objets, InterEditions, Fev 1990.

[MOOR 90] D. J. Moore, *Multimedia Presentation Development Using The Audio Visual Connection*, IBM Systems Journal, vol 29, n° 4, 1990.

[MORE 90] R. del Moretto, MULTIWORKS MULTimedia Interface WORKStation, E.T.W., Nov 90.

[Mult 93] Multimédia solutions, n° 4, Janv 93.

[NANA 90] J. Nanard, *La manipulation Directe en Interface Homme-Machine,* thèse d'état, Université de Montpellier, Dec 1990.

[NEWC 91] S.R. Newcomb, N.A. Kipp, V.T. Newcomb, *the HyTime Hypermedia/ Time-based document Structuring Language,* Communications of the ACM, Nov 1991.

[NIER 87] O.M. Nierstrasz, *Active Objects in Hybrid,* OOPSLA'87 proceedings, vol 22, p 243-253, ACM Press, Oct 1987

[NOEL 90] J.P. Noel, A. Luciani, *Multimedia : vers de nouvelles applications,* Soft & Micro, Sept 1990.

[ODA 88] Information processing - Text and office systems- Office Document Architecture(ODA) and interchange format, ISO 8613-1 à ISO 8613-8, 1989.

[POGG 85] A.Poggio et al, CCWS: *A Computer-Based Multimedia Information System,* Computer, Oct 1985.

[POPE 90] R. Popescu-Zeletin, *From Broadband ISDN to Multimedia Computer Networks,* Computer Networks and ISDN, n° 18, 1989/1990.

[Post] Postscript Language Reference Manuel, Adobe Systems incoporation, Addison Wesley Publishing company, Inc., 1987.

[POST 86] J.B. Postel, G.G. Finn, A.R. Katz, J.K. Reynolds, An Experimental Multimedia Mail System, ACM, T.O.I.S. , vol 6, n° 1, Jan 86.

[POUN 89] D. Pountain, *The X Window System,* Byte, Jan 1989.

[REIS 68] K. Reisz, G. Millar, The technique of film editing Focal Press, 1968.

[REYN 85] J.K. Reynolds, J.B.Postel, A.R.Katz, G.G. Finn, A.L. DeSchon, *The DARPA Experimental Multimedia Mail System,* Computer 1985.

[RFC 1341] N. Borenstein, MIME (Multipurpose Internet Mail Extensions), Mechanisms for Specifying and Describing the format of Internet Message Bodies.

[RFC 1342] K. Moore, Representation of Non-ASCII Text in Internet Message Headers

[RFC 1343] N. Borenstein, A User Agent Configuration Mechanism for Multimedia Mail Format Information.

[ROBI 90] P. Robinson, The Four Multimedia Gospels, BYTE, Fev 1990.

[ROSE 88] J. Rosenberg, C.F. Everhart, N.S. Borenstein, *An overview of the Andrew Message System*, Proceedings of the August 87 sigcomm workshop, ACM 1988.

[ROUD 90] E. Roudier, Présentation et problèmes de la messagerie multimédia, JAM 90, Journées Administrateurs de Messagerie, Dec 90.

[ROUD 91] E. Roudier, J.J. Girardot, *A MultiMedia Mail Manager*, Kamoun & Pouzin éditeurs, Computer Communications, p 69-84, IFIP, ICCC, North Holland, Mai 1991.

[SCHU 90] G. Schürmann, U. Holzmann-Kaiser, *Distributed Multimedia Information Handling and Processing*, IEEE Network Magazine, Nov 1990.]

[SENT 90] A. Senteni, S. Giroux, *From Rock-Bottom to Metalevel Actors : Contribution to an Actor Programming Methodology*, 1990.

[SHEI 86] R.W. Sheifler, J. Gettys, *The X Window System*, ACM Transactions on Graphics, vol 5, n° 2, Avril 1986.

[SPRI 92] G. Springer, D.T. Friedman,, Scheme and the Art of Programming. MIT Press, Cambridge, MA, 1992.

[STEF 90] S. Stefenac, L.Weiman, *Multimedia is it real ?*, MacWorld, Avril 1990.

[THOM 85] R. Thomas, T. Crowley, H. Fordsdick, G. Robertson, R. Saaaf, R. Tomlinson, V. Travers, *Diamond, A Multimedia message system built on a distributed archi tecture*, BBN Laboratories, Computer, Dec 85.

[UNGA 87] D. Ungar, R.B. Smith, *Self : The Power of Simplicity*, OOPSAI'87 Proceedings, vol 22 p 227-242, ACM Press, Dec 1987.

[VEZI 91] G. Vezian, *Multimedia : the future is here*, Tech Images, n° 15, 1991.

[VIN 91] H.M. Vin, P.T. Zellweger, D.C. Swinehart, P.V. Rangan, *Multimedia Conferencing in the Etherphone Environnement*, Computer, vol 24, n° 10, Oct 91.

[WALL 91] G.K. Wallace, *The JPEG Still Picture Compression Standard*, Communications of the ACM, vol 34, n° 4, Avril 1991.

[WAYN 91] P. Wayner, *Inside QuickTime*, BYTE, Dec 1991.

[WILS 92] D. Wilson, *Wrestling with multimedia standards*, Computer Design, Jan 1992.

[WOOD 91] L.Wood, *Script Languages: The Basic of the 1990s?*, BYTE, Avril 1991.

[WYAT 92] B.B. Wyattl, K. Kavi, S. Hufnagel, *Parallelism in Object-Oriented Langages: A Survey,* IEEE Software, Nov 1992.

[X11 90] The X Window System Series, vol 1, 2, 4 et 5, Editions O'Reilly & Associates, Inc.

[X400 88] Recommendations X400-88 du CCITT.

[YAVA 92] R.Yavatkar, *Issues of coordination and Temporal Synchronization in multimedia Communications,* Computer Communication Review, SIGCOMM , Juil 1992.

[YONE 86] A. Yonezawa, J.P. Briot, E. Shibayama, *Object Oriented Concurrent Programming in ABCL/1*, OOPSLA'86 proceedings, vol 21, p 258-286, ACM Press, Nov 86.

[YONE 90] A. Yonezawa, ABCL : An Object-Oriented Concurrent System, MIT Press Series in Computer Systems, 1990.

Liste des figures

FIGURE 1 : exemple de message multimédia....17
FIGURE 2 : station multimédia type....19
FIGURE 3 : impact du multimédia sur le volume des informations....20
FIGURE 4 : récapitulatif des principaux formats en cours....25
FIGURE 5 : système de gestion de messages X400....28
FIGURE 6 : composantes du modèle de communication....45
FIGURE 7 : principe de communication par messagers....52
FIGURE 8 : système de messagers....53
FIGURE 9 : message multimédia interactif....58
FIGURE 10 : organisation spatiale et temporelle du corps du message....59
FIGURE 11 : objets superposables ou disjoints....60
FIGURE 12 : relations temporelles entre deux intervalles de temps....62
FIGURE 13 : exemple de scénario....66
FIGURE 14 : structure arborescente de représentation de scénario....68
FIGURE 15 : fonctionnement général....72
FIGURE 16 scenario exprimé par un script....73
FIGURE 17 architecture hybride....76
FIGURE 18 fonctionnement général....84
FIGURE 19 architecture générale....87
FIGURE 20 composition de messages - éditeur de textes....91
FIGURE 21 outil de visualisation d'images....91
FIGURE 22 : outil de production de sons....92
FIGURE 23 : sous-module de composition de script....93
FIGURE 24 : message texte....94
FIGURE 25 : message 4M....95
FIGURE 26 : outil de configuration....96
FIGURE 27 : outil de télécommande....96
FIGURE 28 : visualisation du script....97
FIGURE 29 : exemple de message 4M....98
FIGURE 30 : format X400 d'un message 4M....99

FIGURE 31 : fenêtre principale 100
FIGURE 32 : interface GLisp 101
FIGURE 33 : hiérarchie des fenêtres iconifiables 106
FIGURE 34 : architecture de l'interface 108
FIGURE 35 : enchaînement des fenêtres 111
FIGURE 36 : bouton et avertisseur 117
FIGURE 37 : exemples de positionnement 119
FIGURE 38 : gestion des tâches 122
FIGURE 39 : implantation des tâches 123
FIGURE 40 : relations spatiales 125
FIGURE 41 : réponse 126
FIGURE 42 : coordination temporelle 128
FIGURE 43 : message Xlib 128
FIGURE 44 : explications de l'outil 132
FIGURE 45 : environnement 144
FIGURE 46 : communications client-serveur 152
FIGURE 47 : architecture de X11 153

Glossaire

4M	MultiMedia Mail Manager
ADPCM	ADaptative Pulse Code Modulation
ANSI	American National Standards Institute
ATM	Asynchronous Transfer Mode
AU	Access Unit
B-ISDN	Broadband Integrated Data Network
CCITT	Comité Consultatif International Télégraphique et Téléphonique
CDI	Compact Disk Interactive
CD-ROM	Compact Disk Read Only Memory
CDROM-XA	Compact Disk ROM eXtended Architecture
CSCW	Computer-Supported Cooperative Work
DVI	Digital Video Interactive
EDI	Échange de Données Informatisées
G3-G4	format d'images fac-similés N/B
GA	Gestionnaire d'Acteurs
CGM	Computer Graphics Metafile
GIF	Graphical Interchange Format
GKS	Graphical Kernel System
HyTime	Hypermedia/Time-based document structuring language
IANA	Internet Assigned Number Authority
ISO	International Standard Organisation
JBIG	standard de codage d'images binaires
JPEG	Joint Photographic Experts Group
IIF	Image Interchange Facility
LWP	Leight Weight Process
MCM	Mail Composition Module
MHEG	Multimedia and Hypermedia Information coding Expert Group
MHS	Message Handling System
MIDI	Musical Instrument Digital Interface
MIME	Multipurpose Internet Mail Extensions
MMUA	MultiMedia User Agent

MPC	Multimedia Personal Computer
MPEG	Moving Pictures Experts Group
MPM	Mail Processing Module
MTP	Mail Transfer Program
MRM	Mail Rendering Module
MTA	Message Transfert Agent
MTS	Message Transfert System
MVC	Modèle Vue Controleur
NTSC	National Television Standards Comitee
ODA	Office (Open) Document Architecture
ODIF	Open Document Interchange Format
OSF	Open Software Fundation
PBM	format de fichiers bitmap
PCT	Part Composition Tool
RFC	Requests For Comment
RNIS	Réseau Numérique à Intégration de Services
RTF	Rich Text Format
SCT	Script Composition Tool
SGML	Standard Generalized Markup Language
SMML	Standard Music Markup Langage
SMTP	Simple Mail Transport Protocol
SPM	Script Processing Module
TIFF	format propriétaire d'images raster
UA	User Agent
UIM	User Interface Module
UIMS	User Interface Managing System
X11	X Window System version 11.x
X400	Serie de Recommendations du CCITT sur un système de messagerie
XPM	X PixMap
WAN	Wide Area Network

4M, une messagerie multimédia opérant par des messages actifs

La messagerie électronique est l'une des nombreuses applications touchées par le concept du multimédia. La messagerie multimédia associe à une messagerie électronique des outils de conception et de transmission de messages composés de différents types de médias (textes, images, séquences audio ou vidéo, etc). L'analyse des besoins rencontrés dans ce domaine nous a amené à étudier le problème plus général de la communication de données quelconques dans des environnements hétérogènes. Notre approche propose un modèle de communication qui prend en compte les diverses contraintes liées à l'échange de données complexes entre applications. Elle repose sur la circulation sur des réseaux, d'agents actifs, représentant les communications par des messagers acteurs. L'objectif est de résoudre le problème de la communication d'informations spécifiques et difficilement structurables comme un message multimédia.

Illustrant cette approche, a été développé un outil de messagerie électronique pour la composition et la restitution de messages multimédias. Ce prototype est un agent utilisateur dont l'un des objectifs principaux est de prendre en compte l'organisation spatio-temporelle de messages multimédias actifs. Ceci est réalisé au moyen d'un scénario qui coordonne la présentation et le comportement du message et qui est représenté, conformément au modèle proposé, par un script. Ce script est exprimé en Lisp et exécuté sur la machine réceptrice pour restituer le message. L'agent utilisateur proposé se présente comme un ensemble d'outils graphiques simples dont la réalisation montre la validation du modèle dans le contexte de la messagerie multimédia.

Mot clés : messagerie multimédia, communication, acteurs, messagers, script.