THESE DE DOCTORAT DE L'UNIVERSITÉ PIERRE ET MARIE CURIE

(PARIS VI)

Spécialité: Informatique

présentée par

Jean-Marc BERNARD et Jean-Luc MOUNIER

pour obtenir le titre de

DOCTEUR DE L'UNIVERSITÉ PARIS VI

Supjet de la thèse

Conception et Mise en Oeuvre d'un environnement système pour la modélisation, l'analyse et la réalisation de systèmes informatiques

Soutenance le 21 Décembre 1990, devant le Jury:

Claude GIRAULT	Président
Paul FEAUTRIER	Rapporteur
Jean-Claude DERNIAME	Rapporteur
Pascal ESTRAILLIER	
Jacques FERBER	
Jean-Marie LABORDE	
Michel LEMOINE	

Résumé

Résumé: Cette thèse décrit l'architecture fonctionnelle de l'atelier défini dans le projet MARS. L'objectif est de concevoir un atelier logiciel ouvert enchaînant les phases de spécification, de validation et de génération de systèmes. Notre proposition d'architecture résulte de l'analyse des besoins et de l'étude des principales composantes des ateliers de génie logiciel. Nous détaillons plus particulièrement les composantes systèmes. Notre architecture repose sur une gestion multi-utilisateurs et multi-sessions dans un environnement distribué hétérogène facilitant l'intégration de nouvelles applications. Nous présentons la réalisation d'un prototype: l'atelier AMI regroupant des services allant de la modélisation graphique à la génération de code à partir de spécifications orientées réseaux de Petri. L'atelier possède un programme d'interface utilisateur Macao basé sur un méta-modèle de graphes assurant l'introduction des données et la visualisation graphique des résultats.

Mots clefs: Atelier de Génie Logiciel, Interface utilisateur, Système distribué, Architecture logicielle, multi-formalismes, Modélisation, Ouverture.

Abstract: This article describes the functional architecture of the software environment defined in the MARS project. The aim is to design an open software environment allowing the user successively to specify, validate and finally generate code. The architecture we suggest is the result of the study of the main components of an software engineering environment and of the careful analysis of the user's needs. This article will focus on the system architecture. Our architecture is based on a multi-session multi-user management in a distributed environment. We will describe the first prototype : the AMI software environment composed of a set of tools from graphical modeling to code generation from Petri net specifications. The Macao user interface is based on a graph meta-model. It allows the graphical data input and the results display

Keywords: Computer Aid Software Engineering, User interface, Distributed systems, Software architecture, Multi-formalism, Modeling techniques, Open software.

Nous tenons à remercier Monsieur le Professeur Claude Girault, de l'université Pierre et Marie Curie, qui a dirigé nos travaux depuis notre entrée à l'Université avec un grand intérêt et de nombreuses idées fructueuses. Nous remercions naturellement Monsieur Pascal Estraillier, maître de conférences à l'Université Pierre et Marie Curie, qui nous a aidé de façon décisive à structurer et rédiger cette thèse. Nous n'oublierons pas son accueil gastronomique du mercredi et des "vacances" studieuses. Ils ont contribué tous deux à la qualité scientifique de cette thèse par leurs relectures pointilleuses et rigoureuses.

Nous remercions vivement Monsieur le Professeur Jean-Claude Derniame, de l'Université de Nancy pour ces remarques pertinentes sur notre travail. Il a bien voulu accepter d'être rapporteur.

Nous remercions, Monsieur le Professeur Paul Feautrier de l'Université Pierre et Marie Curie, pour s'être intéressé à notre travail et d'avoir été rapporteur.

Nous témoignons à Monsieur Michel Lemoine, ingénieur au Centre d'Etude et de Recherche de Toulouse, l'expression de notre gratitude pour avoir examiné avec grande précision notre travail et de faire partie du jury.

Nous remercions Monsieur le Professeur Jacques Ferber, de l'Université Pierre et Marie Curie pour avoir accepté de participer à ce jury et nous apprécions son intérêt pour nos travaux.

Nous remercions Monsieur le Professeur Jean-Marie Laborde, de l'IMAG, Université de Grenoble qui nous a fait l'honneur de participer à ce jury.

Nous remercions l'ensemble du personnel des laboratoires de L'IBP, en particulier ceux de Modélisation Analyse et Réalisation de Systèmes distribués du laboratoire MASI, qui font de ce lieu un cadre agréable où l'ambiance est chaleureuse et animée.

Jean-Marc Bernard remercie Jean-Luc Mounier pour l'honneur qu'il lui a fait d'être co-auteur de cette thèse et pour la mise en page de celle-ci. Il remercie sa famille qui l'a toujours soutenu et qui lui a permis de poursuivre ses études jusqu'à l'aboutissement de ce travail.

Jean-Luc Mounier remercie Jean-Marc Bernard pour l'honneur qu'il lui a fait d'être co-auteur de cette thèse et pour l'avoir incité à rédiger. Il remercie tous ceux qui ne savent pas que recevoir.

Introduction

La conception, la spécification et la réalisation de systèmes informatiques parallèles ou distribués présentent des difficultés importantes d'ordre méthodologique, théorique et pratique. C'est pourquoi l'élaboration d'outils utilisables dans un cadre industriel s'avère être de plus en plus fondamentale dans le cadre du génie logiciel, motivant de nombreux développements et recherches.

Toutefois, les phases successives de la conception de systèmes à la génération de leurs prototypes s'effectuent au moyen de formalismes et d'outils différents. Aussi, pour valider le système étudié et éviter ainsi des erreurs au niveau de l'implémentation, le processus de conception nécessite l'utilisation d'une méthodologie bien définie reposant sur des outils étayés par de solides bases formelles et des conversions automatiques de formalismes. C'est pourquoi notre objectif est de fournir une plate-forme couvrant à la fois la spécification de systèmes, leur vérification et le développement des logiciels.

Alors qu'il existe un grand nombre de travaux sur les méthodologies et la conception d'outils de spécification, d'analyse et de prototypage , beaucoup reste à faire pour concevoir des environnements systèmes sous-jacents afin de résoudre les problèmes de communication, concurrence et coopération entre les outils qui leur assurerait ainsi une plus grande puissance associative.

Notre travail est donc concentré sur l'étude d'un tel environnement système et a essentiellement porté sur l'infrastructure d'un atelier de spécification (MARS), c'est à dire sur la plate-forme accueillant les outils de spécification et l'interface utilisateur. Cette orientation système de notre thèse peut sembler paradoxale bien qu'initialement motivée par une problèmatique concernant un formalisme particulier des Réseaux de Petri. L'historique de cette "dérive" témoigne, dans une certaine mesure, de la grande mutation observée dans de domaine de la conception de logiciels et de l'essor des techniques issues du génie logiciel.

Lorsque nous avons débuté nos recherches, les travaux de l'équipe étaient principalement basés sur la théorie et les applications des réseaux de Petri [Feldbrugge, 1989, Jensen, 1987]. Il était alors naturel d'envisager la programmation d'un environnement logiciel permettant d'une part l'élaboration des modèles et, d'autre part, leurs analyses.

Trés vite, une évaluation des outils existants [Ogive (Azema, Diaz-LAAS)] Petripote [Beaudouin-Lafon, 1983], Papetri [Berthelot, 1990], Serpe [Feldbrugge, 1986], Rafael [Behm, 1985] (Memmi-ENST, BULL), RDPS (Florin-CNAM), GreatSPN [Chiola, 1985] nous a montré qu'aucun logiciel n'intégrait l'introduction de multiples catégories de réseaux de Petri (ordinaires, colorés, à prédicats, temporisés, stochastiques, ...), leurs outils d'analyse et la visualisation graphique des résultats. Nous avons regretté cette multiplicité, l'absence d'interfaces entre ces logiciels, mais chaque produit visait un domaine théorique particulier correspondant trés souvent à l'expérimentation de thèses. Il était alors impossible (ou très difficile) de partager des résultats entre logiciels car la plupart n'utilisaient pas de représentation commune ou de gestion globale des résultats d'analyse.

En dépit du travail considérable fourni, ces outils logiciels restent généralement figés sur l'état de la théorie au moment de leur programmation et ne sont étendus que trop lentement. Répercuter systématiquement les avances théoriques demande un travail fastidieux compte tenu de la variété et de la complexité des modèles proposés. L'apparition de nouveaux types de formalismes pose même des problèmes de réutilisation et de compatibilité des outils déjà réalisés.

Nos objectifs principaux ont donc été d'intégrer aisément de nouveaux modèles, d'assurer le partage et la communication entre logiciels, d'alléger l'insertion d'applications et d'accélérer la conception de nouvelles applications. Une première étude a consisté à élaborer un cahier des charges puis à concevoir et à réaliser une plate-forme de développement et d'utilisation d'outils interactifs de création, manipulation et analyse de réseaux de Petri.

Nous avons adopté une architecture doublement distribuée pour l'atelier et les difficultés à résoudre ont essentiellement porté sur la gestion de la communication et le partage d'informations entre des systèmes et des applications hétérogènes. D'une part nous avons localisé l'exécution de l'interface utilisateur sur des postes de travail pour des raisons ergonomiques (interface utilisateur interactive et conviviale disponible sur ce type d'ordinateur, facilité de transport et d'installation), techniques (primitives graphiques spécialisées intégrées, spécialisation du poste de travail dédié) et économiques (large diffusion, faible coût d'achat et de maintenance). D'autre part, dans un souci de performance, nous avons localisé les exécutions des outils applicatifs d'analyse des modèles, sur un parc de machines possédant un système d'exploitation multi-tâches, multi-utilisateurs et multi-sites (système Unix) muni d'un système de fichiers répartis. En effet ces outils nécessitent en général, une forte puissance de calcul et manipulent de grands volumes d'informations.

L'architecture logicielle ainsi définie nous a permis de réaliser un Atelier de Modélisation Interactif (AMI) qui supporte un ensemble d'outils permettant la spécification et l'analyse de réseaux de Petri. L'atelier a ainsi été utilisé pour valider plusieurs systèmes [Cousin, 1988] [Merchant]. Il a été éprouvé avec des intégrations de différents outils en démonstrations publiques lors des congrès annuels sur les réseaux de Petri (Bonn 1989, Paris 1990).

Nous n'avons cependant pas limité l'atelier au domaine des réseaux de Petri. Dès les premiers temps, nous avons en effet pris conscience de la nécessité de généraliser les notions de modèles et d'application. Cette approche a été étendue dans une seconde phase de notre travail.

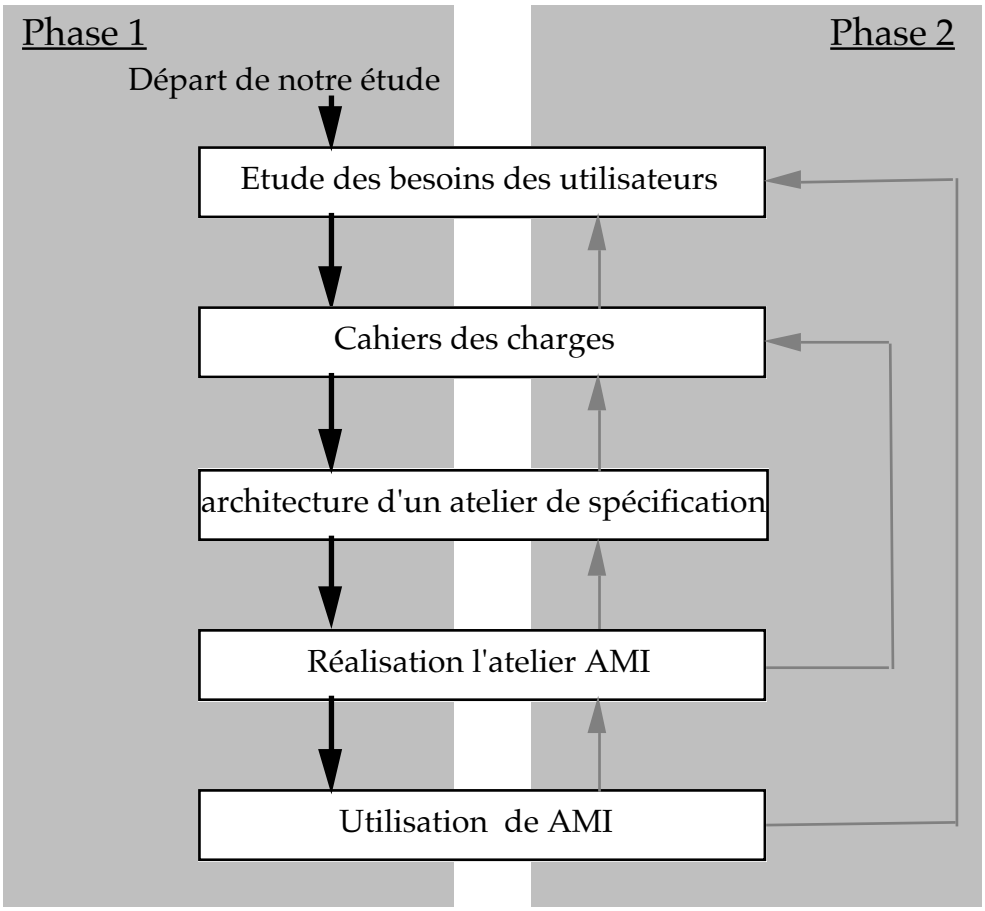
Cette première version d'AMI s'est rapidement trouvée confrontée à quatre phénomènes :

- Des travaux de recherche dans le cadre du projet MARS, se sont orientés vers des outils de génération semi-automatique de code et des outils de traduction de spécifications qui ont demandé des communications et des interfaces plus sophistiquées.
- L'ergonomie et l'ouverture de l'atelier ont suscité dans l'équipe ainsi que dans d'autres équipes de recherche (Hambourg, Orsay, Turin, Toulouse,) et même chez un industriel (Sligos)), l'intégration de nombreux outils existants.
- Certains travaux de recherche du Projets MARS, associés à des projets européens , ont nécessité de nouveaux formalismes de modèlisations différents des réseaux de Petri.
- L'utilisation de AMI, a permis de se rendre compte que les besoins avaient évolué et a fait découvrir aux utilisateurs des possibilités telles que la coopération entre les utilisateurs et la hiérarchies de modèles.

Pour que l'atelier AMI supporte tous ces changements, principalement induits par la problèmatique du projet Mars, nous avons étendu les ambitions du cahier des charges. L'objectif du projet est maintenant d'assiter l'ingénieur durant l'ensemble des étapes de spécification, de validation et de génération de prototypes. Les domaines d'application du projets sont les systèmes distribué, des protocoles de communications, de la productique et du temps réel. Ces domaines sont difficilement appréhendables sans un environnement logiciel muni des formalismes adaptés aux domaines et susceptibles d'être validés automatiquement.

Notre seconde version de l'atelier AMI prolonge alors la première version par la construction graphique de systèmes d'objets et la visualisation de leurs propriétés. Les interfaces, extensibles à de nouvelles catégories d'objets, sont téléchargeables sur les postes de travail et simplifient le développement homogène des divers outils. L'ajout de ces nouvelles fonctionnalités nous a amenés à accroître l'extensibilité et l'ouverture de notre architecture logicielle.

Les deux phases de notre travail sont résumées sous forme du schéma ci-dessous. Les problèmes d'extensibilité de gestion d'objets et de transparence du système support ont nécessité des modifications dans l'architecture même de l'atelier. Mais, la majorité de ces modifications, liées aux outils, se sont avérées relever principalement de leur environnement système.



Délibérément, nous avons basé notre atelier sur un ensemble de stations de travail hétérogènes connectées en réseau. Il s'agit d'élaborer un environnement système pour la réalisation d'un atelier intégré supportant l'exécution d'outils applicatifs. Une des contraintes est de mettre en oeuvre, de manière transparente aux utilisateurs, l'exécution des différents outils dans cet environnement distribué. Ceci apporte une nouvelle vision des stations de travail.

Le rôle de l'environnement système est alors de gérer les problèmes de communication, concurrence et coopération entre les exécutions d'outils applicatifs et entre les outils eux mêmes. A cet effet, nous avons créé un langage commun de représentation de données facilite la communication des résultats entre les outils coopératifs.

Pour la réalisation de l'environnement système, l'importance des problèmes à surmonter tient surtout aux difficultés conceptuelles. Un des problèmes fondamentaux des systèmes distribués est l'impossibilité de capter de façon instantanée l'état global d'un application ou d'un système réparti. Cette absence d'état global rend nécessaire la gestion rigoureuse des informations associées aux utilisateurs, aux services et aux sessions de travail. Enfin, la volonté d'exploiter le parallélisme pour augmenter l'efficacité et la robustesse nécessite la prise en compte des critères de distribution entre les sites.

La séparation fonctionnelle et physique de l'interface utilisateur offre la possibilité de développer les outils applicatifs en s'affranchissant des problèmes graphiques et d'interactivité. La réalisation d'interfaces utilisateurs est une tâche fastidieuse, nous avons proposé une forme graphique homogène pour la saisie des données, la gestion des menus et des diagnostics, ainsi que pour la visualisation des résultats. Cette normalisation permet de réaliser, de manière efficace, des interfaces paramétrées que le développeur particularise pour chaque traitement par des descriptifs. L'interface devient ainsi une donnée et non plus une partie du code de l'application. Le concepteur d'applications n'a alors plus à se préoccuper de l'introduction et de la manipulation des données graphiques.

Nous avons doté la station de travail supportant l'interface utilisateur d'un mode d'utilisation autonome. L'utilisateur peut ainsi procéder à la manipulations graphiques des différents modèles et les stocker localement. Il est alors nécessaire, lors de la reconnexion, d'assurer la cohérence des données. Des objectifs de performance et de respect des contraintes d'interface utilisateur nous ont amené à concevoir un atelier multi-sessions dans lequel chaque utilisateur peut simultannément effectuer plusieurs traitements. L'atelier prend alors en charge la gestion des contextes d'exécution qui commutent en fonction des demandes de traitement.

La prise en compte par l'atelier de plusieurs formalismes constitue une originalité de notre travail, surtout dans le cadre des spécifications formelles pour lesquelles une évolution rapide des modèles est observée. La gestion externe des formalismes, au moyen d'un langage d'objets, constitue un point clé de l'atelier. Sa mise en oeuvre assure une ouverture indéniable mais contraint à une rigueur importante dans la conception de l'architecture fonctionnelle de l'atelier. En effet, il est nécessaire d'isoler, dans chacune de ses composantes, les opérations associées aux différents formalismes. Naturellement, la nature distribuée de l'environnement rend une telle gestion plus complexe.

Notre thèse ne reprend naturellement pas les résultats intermédiaires obtenus dans la première phase de notre travail mais décrit, dans leur majorité, sous une forme agrégée, les résultats finals. Cette présentation vise essentiellement à définir le contexte pour lequel l'atelier a été conçu (partie I), à préciser son architecture fonctionnelle (partie II) et à présenter le produit logiciel réalisé (partie III).

La **partie I** constitue le cahier des charges d'un atelier de spécification supportant le projet MARS. L'ensemble des composantes nécessaires sont identifiées et définies. Les principales caractéristiques sont dégagées.

Le chapitre 1 introduit les outils de modélisation, d'analyse et de réalisation de systèmes. L'ossature générale d'un atelier de spécification est présentée. L'objectif est de positionner clairement notre travail dans le cadre des recherches sur les ateliers de génie logiciel.

Le chapitre 2 énonce les besoins auxquels doit répondre un atelier de spécification vis à vis des modélisateurs, des théoriciens, des concepteurs d'outils, mais aussi des administrateurs .

Le chapitre 3 introduit la problèmatique des aspects purement système des ateliers de spécification. En plus de critères d'ouverture d'un atelier, la prise en compte d'un environnement distribué induit, en effet, des contraintes de tolérance aux pannes et de gestion d'hétérogénéïté.

Le chapitre 4 présente une synthèse des précédents chapitres dans un tableau, tous les besoins recensés et les problèmes sous jacents examinés.

Le chapitre 5 définit précisément les composantes que nous avons retenues dans cette étude. Il s'agit des composantes, orientées système, pour la description d'un environnement de spécifications. Certaines sont directement issues de la nature répartie de l'atelier, d'autres sont la conséquence des types de services offerts. Les composantes systèmes sont divisées en deux classes: les composantes liées à l'infrastructure système et celles imposées par le domaine applicatif. Le côté applicatif du projet MARS apparaît ainsi comme la partie supérieure de l'iceberg de sorte que l'infrastruture système constitue la partie la plus importante de notre travail.

La **partie II** détaille l'analyse fonctionnelle de l'environnement système nécessaire au projet MARS.

L'atelier a été conçu de manière hiérarchique. Il peut ainsi être vu comme une encapsulation de fonctionnalités qui contribuent à la prise en charge du travail demandé par l'utilisateur. Dans cette partie, nous définissons tous les objets manipulés dans l'architecture, ainsi que les liens entre ces objets. Cette architecture prend en compte la séparation de l'interface utilisateur et des outils de modélisation, de validation et de génération de prototypes.

Le chapitre 1 présente l'architecture générale de l'environnement système d'un atelier de spécification supportant le projet Mars. Le chapitre 2 rappelle les fonctionnalités de base attendues du système d'exploitation support. Le chapitre 3 définit les extensions nécessaires au système de base pour supporter d'une part l'interface utilisateur et, d'autre part, la gestion de la répartition et du contexte multi-utilisateurs.

Le chapitre 4 décrit précisément les différents éléments permettant la gestion interne de l'interface utilisateur. Le chapitre 5 détaille les modules de gestion des utilisateurs et des différents services qui leur sont offerts.

Le chapitre 6 introduit le concept de station de travail virtuelle assurant l'indépendance entre l'interface utilisateur et les services. Cette caractéristique, observée par l'utilisateur comme une gestion de multi-fenêtrages dissimule, au niveau de l'environnement système, des mécanismes de gestion de contextes, soumis à de multiples commutations. L'atelier supporte aussi les contraintes d'un environnement réparti, en particulier, celles liées à la communication d'informations et au stockage distant de données.

Le chapitre 7 présente les fonctionnalités permettant d'intégrer dans l'atelier les différents outils applicatifs. Le chapitre 8 donne les caractéristiques des différents outils possibles. Une description détaillée des fonctionnalités offertes par l'application d'interface utilisateur est proposée.

Le chapitre 9 conclut l'établissement de l'architecture fonctionnelle en présentant une synthèse des différents modules décrits dans les chapitres précédents. En particulier, nous montrons que la solution proposée répond à l'ensemble des besoins exprimés dans le cahier des charges de la partie I.

Nous présentons dans la **partie III** une première réalisation de l'atelier AMI (Atelier de Modélisation Interactif). Après avoir précisé les hypothèses et les choix des matériels et des systèmes, nous détaillons l'implémentation de AMI dans l'environnement Unix et Macintosh. La version actuelle de l'atelier AMI (V 1.2) inclut l'essentiel des fonctionnalités introduites dans la partie II. Dans la description détaillée qui en est faite, nous mentionnons les fonctionnalités qui seront l'objet de AMI v2.0.

Le chapitre 1 donne une présentation générale de l'emploi de l'atelier dans le domaine des réseaux de Petri.

Le chapitre 2 décrit les environnements système et réseau choisis pour l'implémentation de l'atelier AMI.

Le chapitre 3 présente l'état d'implémentation de la version 1.2.

Le chapitre 4 offre une description de synthèse des différents langages de communication décrits dans la partie II. Ce chapitre montre l'uniformisation de la syntaxe choisie pour les différents langages.

Le chapitre 5 et 6 décrivent la réalisation de l'application d'interaction utilisateur et de la structure d'accueil.

Le chapitre 7 synthétise l'utilisation de l'atelier par un survol des fonctionnalités offertes aux utilisateurs.

Notre contribution a été de concevoir et de réaliser un système complet et cohérent. Ainsi, compte tenu du grand niveau d'interaction entre les différentes composantes, il a souvent été nécessaire de définir communément de nombreux mécanismes, les différents points de vue étant complémentaires.

Dans ce travail de recherche, Jean-Marc Bernard s'est plus spécifiquement interessé à l'i**nfrastructure système** et aux communications tandis que Jean-Luc Mounier a focalisé son étude sur les problèmes liés à l'**interface utilisateur**. Pour la partie de réalisation du prototype, la figure 2 de la partie III montre le découpage effectué.

Le document présenté est donc le résultat d'un travail commun. La contrainte d'un document unique synthétisant les différents aspects a nécessité un processus itératif d'écriture et de relecture de la part des deux auteurs. Si bien qu'il est assez difficile, a posteriori, d'attribuer de manière péremptoire la paternité de chacun des chapitres. Ainsi, l'expression du cahier des charges (partie I) reflète la synthèse des besoins des deux aspects.

Cependant, au niveau de la rédaction initiale des parties II et III, une séparation peut être mise en évidence. Selon l'orientation des recherches de chacun, Jean-Marc Bernard a pris en charge la rédaction des chapitres 3.2, 5, 6, 8.1 de la partie II et 6 de la partie III et Jean-Luc Mounier a assuré la rédaction des chapitres 3.1, 4, 7.4, 8.2 de la partie II et 5,7 de la partie III.

Le projet MARS

Table des matières

Introduction

Cette partie constitue le cahier des charges d'un atelier de spécification. Le cadre de cette étude repose sur le projet MARS, développé dans l'équipe "Modélisation, Analyse et Réalisation des Systèmes" du laboratoire de Méthodologie et Architecture des Systèmes Informatiques (MASI).

En positionnant clairement notre travail dans le cadre des recherches sur les ateliers de génie logiciel, nous énonçons les besoins auxquels doit répondre un atelier de spécification. Ces besoins tant industriels qu'universitaires concernent les modélisateurs, les théoriciens, les concepteurs d'outils, mais aussi les administrateurs .

Nous introduisons la problèmatique des aspects purement système des ateliers de spécification. En plus de critères d'ouverture d'un atelier, la prise en compte d'un environnement distribué induit, en effet, des contraintes de tolérance aux pannes et de gestion d'hétérogénéïté.

Nous présentons dans un tableau une synthèse de tous les besoins recensés et les problèmes sous jacents examinés.

Nous définisons précisément les composantes que nous avons retenues dans cette étude. Il s'agit des composantes, orientées système, pour la description d'un environnement de spécifications. Certaines sont directement issues de la nature répartie de l'atelier, d'autres sont la conséquence des types de services offerts. Les composantes systèmes sont divisées en deux classes: les composantes liées à l'infrastructure système et celles imposées par le domaine applicatif.

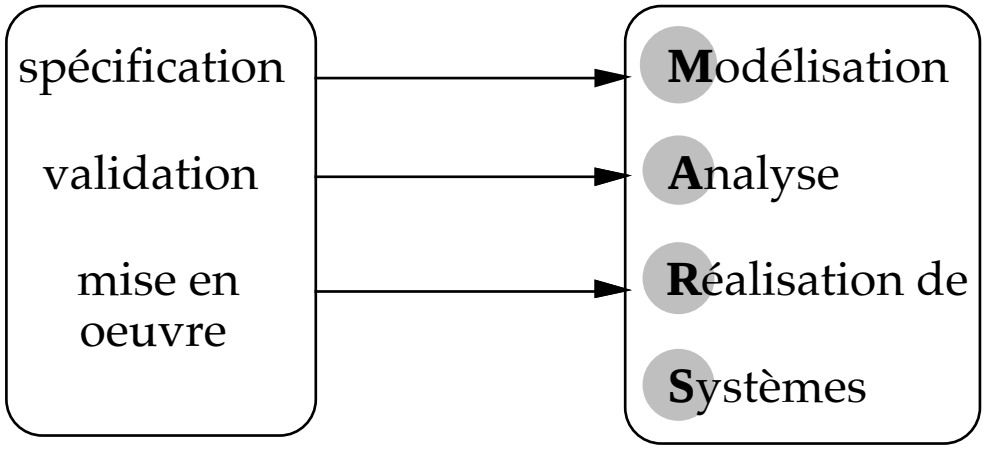
1. Présentation du projet MARS

La difficulté de mise en oeuvre des systèmes complexes tient aux problèmes de spécification, d'organisation, de développement, d'intégration et de vérification d'ensembles d'actions nombreuses et compliquées. Ces problèmes sont aggravés pour les systèmes distribués par la difficulté de maîtriser dynamiquement les interactions de multitudes de programmes, de mécanismes et de personnes œuvrant en parallèle, communiquant, coopérant, partageant des ressources communes [Ramamoorthy, 1987]. Par exemple, le passage de l'électromécanique à l'informatique dans un secteur industriel fait que des systèmes à concevoir, tout en étant plus puissants et plus performants sont d'une complexité telle que les méthodes de conception et de vérification traditionnelles ne sont plus applicables. La question n'est pas d'offrir des outils de synchronisation et communication mais de garantir qu'ils sont utilisés correctement.

Ces problèmes sous-jacents de synchronisation sont d'autant plus difficiles à contrôler que les comportements des entités peuvent être indéterministes et qu'un fonctionnement distribué empêche toute vision globale.

L'analyse se complique considérablement si l'on exige que des erreurs, défaillances et pannes n'entraînent pas de conséquences désastreuses. Des vérifications sont nécessaires pour garantir que des situations indésirables ne peuvent se produire même dans des circonstances exceptionnelles. Pour diminuer les coûts et le temps de développement il est donc nécessaire d'effectuer ces vérifications le plus tôt possible.

Le projet MARS concerne la définition et la mise en oeuvre des outils logiciels nécessaires à un atelier de spécification de systèmes informatiques. Le but est de fournir aux ingénieurs, des outils de conception de systèmes distribués afin de leur permettre, non seulement de spécifier et de vérifier la correction de leur spécification, mais aussi de produire un prototype de leur système réaliste et conforme à leurs spécifications.



Un tel atelier correspond à un réel besoin car la multiplication des techniques de spécification formelle et l'essor des techniques d'analyse rend très difficile le partage de résultats entre outils. La plupart des applications actuelles n'utilisent pas de représentation intermédiaire commune et il n'y a pas de gestion globale des résultats sur une spécification analysée avec différents outils.

Les domaines d'application visés par le projet sont les systèmes distribués, les protocoles de communication, la productique et le temps réel. Ces domaines d'application ont la particularité d'exprimer le parallélisme et l'indéterminisme des événements, ils sont donc difficilement appréhendables sans environnement logiciel.

1.1. Présentation générale

Le métier d'ingénieur évolue plutôt vers l'expression de la solution d'un problème, suivant un **formalisme**, que vers la réalisation effective du produit. Alors, dans le but de garantir la qualité du système réalisé et d'augmenter l'efficacité des ingénieurs, le projet MARS a pour objectif d'assister les ingénieurs tout au long des étapes de spécification, de validation et de génération de prototypes.

1.1.1. Méthode générale

Le premier aspect à prendre en compte lors de la réalisation d'un système (quel qu'il soit) est de définir les besoins de ses futurs utilisateurs.

La figure 1.1.a présente les différentes étapes assurant le passage d'un spécification informelle à la génération de code.

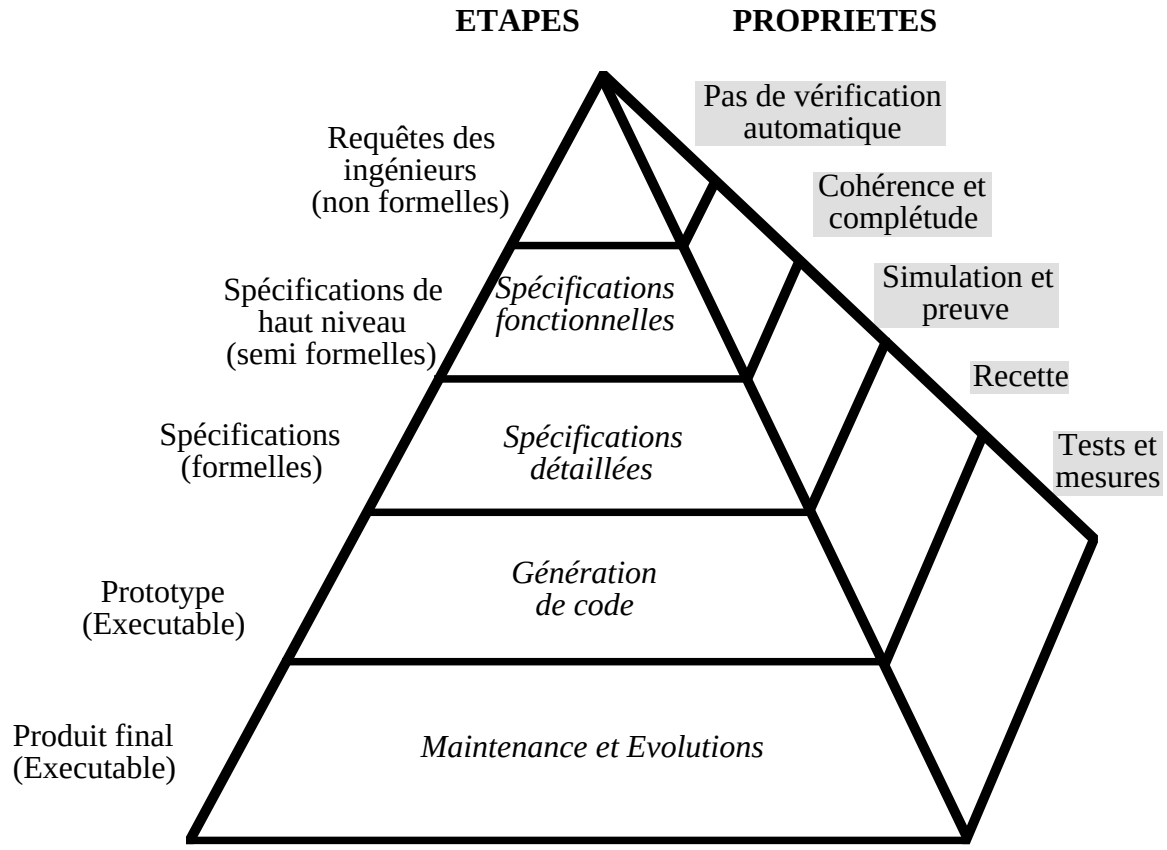


Figure 1.1.a: Méthodologie de MARS

A chaque étape, on associe un formalisme de spécification (ou éventuellement plusieurs). En fonction de l'étape considérée, plusieurs catégories de formalismes sont utilisés avec des méthodes de plus en plus ardues et coûteuses.

Une **plate-forme commune d'outils logiciels** est nécessaire pour assurer l'interfaçage entre les formalismes, pour faciliter la manipulation des modèles appropriés à chaque étape et pour garantir la cohérence des analyses lors des modifications de modèles.

De tels outils de conception doivent en particulier:

- Permettre la conception de systèmes complexes.
  Le formalisme de spécification doit être structuré, modulaire et reposer sur une méthodologie bien définie. De plus, le formalisme de spécification doit être facilement compréhensible et éventuellement dédié à un type d'application spécifique (systèmes de téléphonie, de télécommunications, de production.....).
- Permettre l'analyse (éventuellement par étapes successives) du système spécifié jusqu'à l'obtention de la garantie de son bon fonctionnement.
  Un ensemble d'outils d'analyse doit fournir une validation formelle et interactive du système spécifié.
- Fournir les résultats d'analyse permettant la correction d'une spécification donnée.
  Les résultats d'analyse fournis à l'utilisateur doivent être suffisamment clairs et retranscris dans le langage de spécification originel afin d'être utilisables, dés réception.
- Permettre éventuellement des vérifications spécifiques (évaluations de performance, résolution de problèmes d'ordonnancement...).
  Le formalisme doit permettre la description et l'analyse de ces points spécifiques.
- Assurer le prototypage de la spécification validée par génération semi-automatique de code.
- Permettre l' animation automatique (ou quasi-automatique) de la spécification validée. C'est à dire, l'exécution du prototype avec visualisation graphique de son évolution.

Il est préférable de détecter le plus tôt possible qu'une spécification comporte des erreurs ou des inadéquations, que de s'en rendre compte au stade du produit réalisé. Il faut, pour cela, utiliser des formalismes adaptés aux domaines d'application et validables automatiquement. Lors des différentes étapes de développement d'un système plusieurs formalismes sont souvent utilisés et doivent permettre des vérifications avant de passer à l'étape suivante.

1.1.2. Mise en oeuvre

Il est clair que les étapes de la méthodologie proposée doivent pouvoir être supportées par des outils logiciels.

Afin d'être largement (et efficacement !) utilisés, ces outils de conception de systèmes doivent répondre aux besoins industriels (formalisme de spécification puissant et facilement utilisable, validation formelle, interprétation simple des résultats d'analyse ...).

L'accroissement du nombre des logiciels amène à souhaiter des bases de connaissances intégrant de nombreuses descriptions, propriétés, versions, pour fournir un environnement adapté à la conception collective de systèmes.

La figure 1.1.b ci-après présente succinctement les interactions entre les différents outils existants. La spécification initiale est d'abord traduite en un modèle qui joue un rôle de pivot pour toute la méthodologie. En effet, on peut en tirer des estimations par simulation, des propriétés par analyse, ou un prototype par génération de code. Pour répondre aux questions de l'ingénieur sur les propriétés de son modèle, il est essentiel de disposer de tables de correspondance entre la spécification et le modèle.

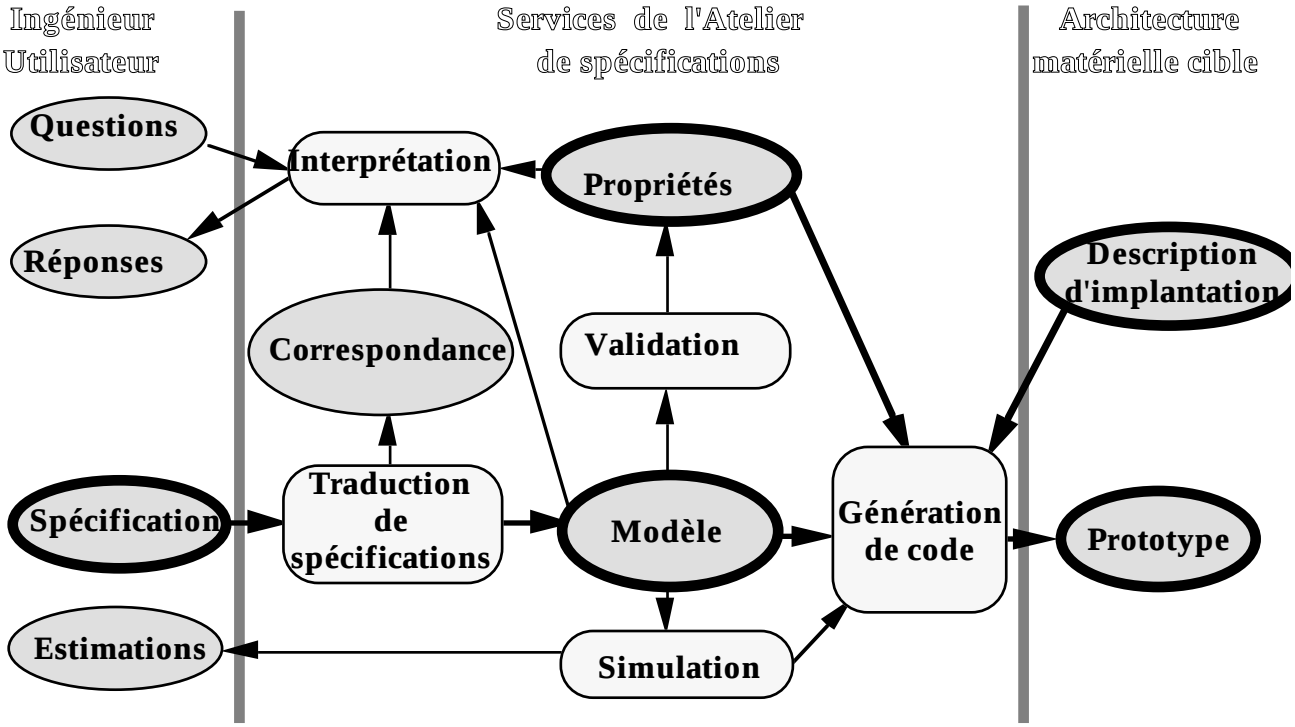


Figure 1.1.b: Utilisation du projet MARS

1.2. Atelier de spécification

La définition de l'architecture d'un atelier de spécification repose sur une étude préalable des besoins spécifiques liés à l'activité de spécification et des composantes traditionnelles des ateliers de génie logiciel. Nous analysons et décrivons les composantes essentielles d'un atelier supportant la spécification, l'analyse et la réalisation de systèmes.

Après un survol des caractéristiques principales des Ateliers de Génie Logiciels (AGL), nous positionnerons le projet MARS par rapport aux concepts des AGL. Compte tenu de notre domaine de recherche, notre approche sera davantage orientée vers les **aspects systèmes** des ateliers de génie logiciel.

1.2.1. Atelier de spécification et atelier de génie logiciel

Cette étude des AGL va nous permettre de préciser les objectifs et les caractéristiques de l'architecture d'un atelier de spécification. Le génie logiciel répond à un besoin industriel pour **construire à faible coût** des logiciels. La production de logiciels est passée à l'étape industrielle, car elle coûte cher en fabrication et surtout en maintenance (suivant les sources de 70 à 80% du coût total). Historiquement, diverses raisons ont amené les industriels à s'intéresser au génie logiciel: diversité des langages de programmation, manque de méthodes de développement, outils non portables et autonomes. Des solutions existent en utilisant des méthodes comme Merise [Collongues, 1987], STP [Wasserman, 1987], SADT [Marca, 1987] ou HOOD [Jaulent, 1990]..., un formalisme basé sur les réseaux de Petri [Endres, 1988], des langages de haut niveau comme Ada [Booch, 1988], Eiffel [Broussard, 1990], Spoke [Caseau, 1987], et des environnements de support de projets (AGL) couvrant le cycle de vie du logiciel [Arthaud, 1989][Brettnacher, 1988][CXP, 1989].

Un AGL permet donc **l'assistance** pendant la vie du logiciel, le suivi du logiciel (planification, contrôle des délais...), l'assurance et le contrôle de la qualité. Il peut être perçu comme un "**système organisé**" avec ses structures, ses personnes et les tâches réparties entre ces personnes. C'est pourquoi, les AGL proposent des ensembles d'outils cohérents coopérants et l'utilisation d'une **interface utilisateur** ergonomique. On peut décrire un AGL par son architecture matérielle, par le dialogue homme-machine, et par les méthodes utilisées tout au long du processus de développement [Drouas, 1982]. Des approches récentes intègrent des méthodes d'intelligence artificielle pour suivre et aider le développement des applications [Hughes, 1988].

Nous rappelons une définition d'un atelier de génie logiciel et ses objectifs:

> **Définition:** Atelier de génie logiciel
>
> Un **atelier de génie logiciel** (AGL) permet de spécifier du logiciel [Bernas, 1989], de le réaliser tout en gérant, pilotant et intégrant des outils.
>
> Un AGL doit permettre d'accroître la productivité de la réalisation de logiciels tout en maintenant une meilleure qualité.

Tous les ateliers n'ont pas la même vocation et il est **difficile de proposer une classification des différents AGL** [Drouas and Nerson, 1982] car cela dépend de l'approche de leurs concepteurs ainsi que des formalismes choisis. Par exemple le terme "environnement de programmation" semble s'appliquer à des AGL issus d'institutions de recherche et proposant un ensemble d'outils très évolués destinés à réaliser du logiciel très complexe [Bernas, 1989]. Mais un environnement peut être aussi vu comme un cadre de production ou de gestion...

La spécification doit constituer une référence tout au long du cycle de vie [Boehm, 1981]. Un AGL doit assurer deux types de cohérence : la **cohérence logique** [Roubine, 1989] qui correspond à une coordination judicieuse des méthodes employées durant les différentes phases du cycle de vie et la **cohérence physique** qui représente la capacité des outils à opérer sur un ensemble cohérent de données (**interopérabilité**). On aboutit donc à la notion d'**atelier intégré** [Roubine, 1989] qui permet d'offrir dans un même environnement un ensemble cohérent et complet d'outils de génie logiciel supportant l'ensemble des activités du cycle de vie:

- **outils "verticaux"** : spécification, conception, codage, test, vérification et validation, prototypage…
- **outils "horizontaux"** : documentation, gestion de configuration, gestion de projet, communication…

Le cycle de vie du logiciel est passé dans le domaine courant et a été normalisé. La figure 1.2.a montre le cycle de vie du logiciel proposé par l'Association Française pour le Contrôle Industriel de Qualité [Jaulent, 1990].

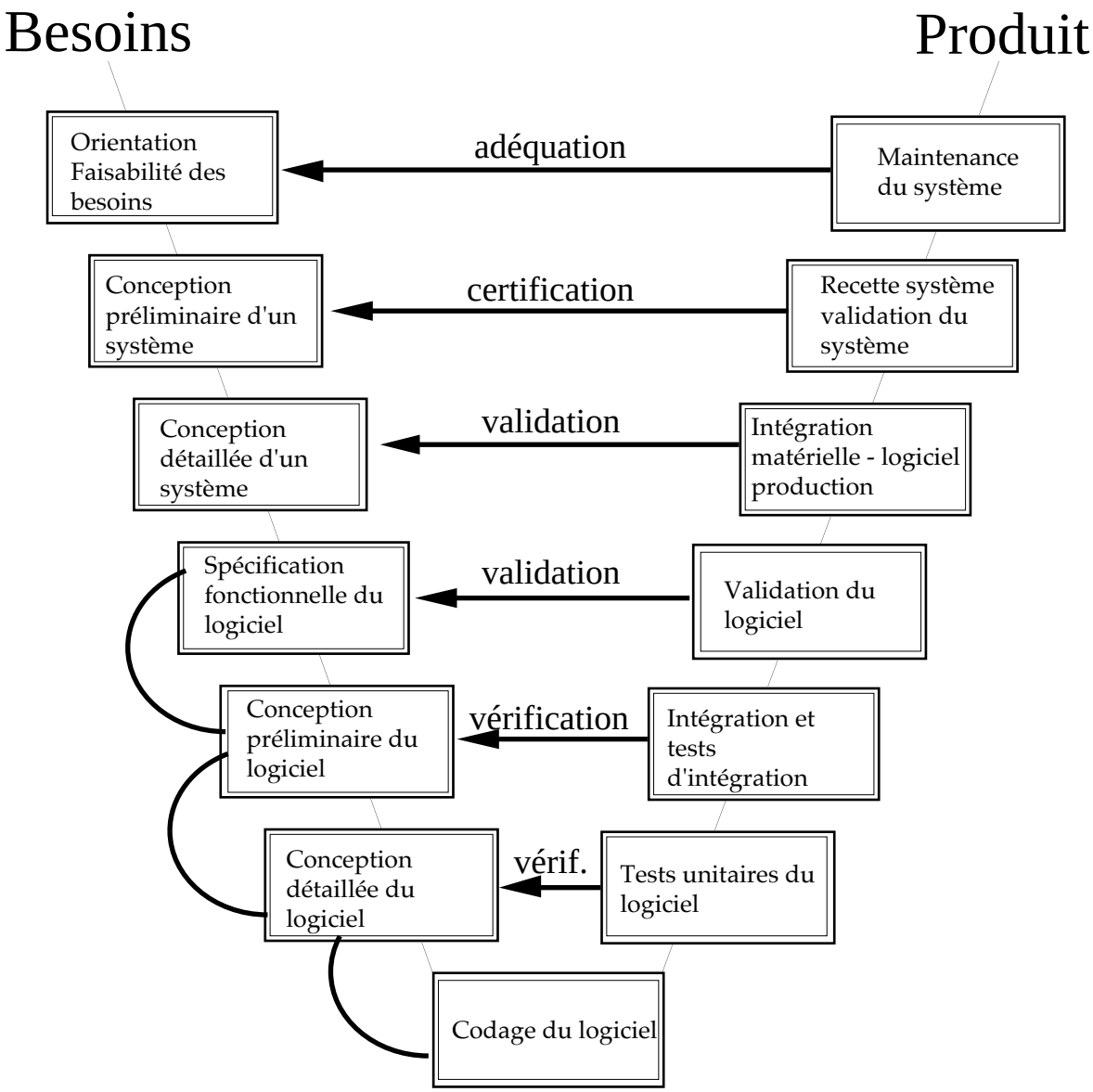


Figure 1.2.a: Cycle de vie du logiciel (AFCIQ)

Ce modèle montre bien que la démarche de spécification-conception est globalement descendante tandis que la phase de réalisation test est globalement ascendante, car il s'agit d'assembler les composantes pour obtenir les fonctionnalités requises.

L'axe horizontal représente les phases du projet et la durée de chacune. L'axe vertical représente le niveau d'abstraction pour l'application. Les phases de spécification et de conception conduisent à définir des niveaux de description de plus en plus détaillés. Pour une application logicielle, le codage est la forme la plus détaillée.

Les phases d'intégration, de test et de vérification permettent d'évaluer la conformité de la réalisation pour chaque niveau de la conception en commençant par les parties les plus élémentaires puis en remontant progressivement jusqu'au produit complet. Pour cela, en correspondance de chaque phase de la conception est associée une phase de test et d'évaluation de la conformité. Ainsi, les deux démarches descendante et ascendante sont complémentaires.

Parmi les environnements qui se réclament d'un tel cycle de vie, il faut cependant faire une différence, entre ceux associés à la programmation et ceux prenant en charge l'exécution.

- Un **environnement de programmation** doit permettre de suivre le cycle de vie d'un logiciel et aider à la programmation et la coopération entre programmeur et concepteur (OASIS [De Postel, 1989], NSE [NSE, 1990]). Un tel environnement permet l'aide à la conception de logiciel et à sa maintenance [Arora, 1985]. Dans un environnement de programmation, l'utilisateur est responsable de l'emploi successif des différents outils (comme dans Unix).
- Un **environnement d'exécution** (IPSE [Lehman, 1987]: Integrated Programming Support Environment ou SEE: Software Engineering Environment) [PCTE, 1989] [Taylor, 1988] doit permettre une coopération des outils entre eux, faciliter la portabilité des outils (machine, système d'exploitation), définir les interfaces normalisées entre outils, permettre une exécution automatique dans un environnement distribué et donner une interface utilisateur. Un environnement d'exécution permet de supporter des ateliers de génie logiciel.

Notre travail s'est orienté vers les environnements d'exécution.

L'environnement d'un atelier de spécification n'est pas lié à un type particulier de système d'exploitation ou même de machine. Le but pour un utilisateur dans un environnement distribué hétérogène, est que la présentation et l'utilisation des outils soient identiques, en regard du système d'exploitation et des architectures matérielles des machines.

Un système d'exploitation cache les détails sous-jacents de la machine et donne des facilités de communication et d'exécution des programme. L'environnement système de MARS peut être vu comme un système qui cache non pas une machine mais l'ensemble des machines du réseau et les systèmes d'exploitations: il gère le lancement, l'exécution, la synchronisation des outils et il rend le réseau transparent aux utilisateurs.

Un des dilemmes a résoudre consiste à définir la barrière entre la construction d'un AGL général et d'un AGL spécialisé, c'est à dire le degré de paramètrisation d'un AGL.

Alors que les AGL pour la programmation de grandes applications sont étudiés depuis longtemps, beaucoup reste à faire pour la programmation de systèmes afin de tenir compte des problèmes de communication, concurrence et coopération.

Nous avons donc choisi de spécialiser MARS qui n'est pas un AGL général mais un **atelier de spécification de systèmes**. Ce type d'atelier répond à des besoins spécifiques. La construction des systèmes relève de différents domaines qui vont de la définition des besoins (spécification) à la gestion projets.... définit un environnement de conception comme l'ensemble méthodologique, technique et progiciel qui est conçu et développé dans le but de supporter une approche de conception.

Le métier d'ingénieur évolue vers la **formulation de l'énoncé** du problème suivant un certain formalisme jusqu'à l'obtention d'une solution que plutôt vers la réalisation effective [Arthaud and Roan, 1989]. Il est préférable de détecter le plus tôt possible qu'un énoncé paraît absurde (est faux), que de s'en rendre compte à la fin de la chaîne au niveau du produit réalisé.

Nous pouvons donner notre acception d'un atelier de spécification et de ses objectifs:

> **Définition:** Atelier de spécification
>
> Un **atelier de spécification** assiste l'ingénieur pendant les étapes de spécification jusqu'à la génération de code, tout en maintenant une cohérence globale des données. L'objectif est de garantir la qualité du système réalisé. Un point fondamental communément reconnu est la séparation entre la **spécification** et la **conception**. La spécification doit être une référence tout au long de la chaîne.

On peut distinguer **deux générations** d'atelier de génie logiciel [Muenier, 1989].

La première génération permettait l'**enchaînement manuel** des outils simplement par échanges de fichiers et par accès à l'environnement système (ce qui oblige à le connaître). Dans la génération actuelle d'atelier, les outils sont étroitement **intégrés** et dialoguent à travers des **structures de données** adaptées permettent d'exprimer la sémantique des outils.

Nous pensons que partager des structures de données lie trop les applications à une architecture fixée et entraîne une centralisation excessive. Le code des applications devient ainsi fortement dépendant de ces structures de données, même si on introduit des primitives d'accès à ces structures. Le dialogue entre les outils devrait donc s'effectuer par **envoi de messages** entre les différents acteurs constituant les outils.

Nous avons remarqué que les environnements sont fermés et souvent **liés à une méthodologie**, qu'ils permettent mal les évolutions et la personnalisation, car **l'intégration** de nouveaux outils est souvent difficile, voire impossible. La communication est pauvre et ne supporte pas toujours le travail en équipe. L'utilisation d'une structure d'accueil commune aux différents outils et méthodes est nécessaire pour combler les faiblesses actuelles des AGL [Giavitto, 1989].

1.2.2. Structure d'accueil

Un atelier de spécifications intègre une **structure d'accueil** qui fournit des moyens de communication entre les outils (éventuellement dans un environnement distribué). Cette structure d'accueil (Figure 1.2.b) supporte une **interface utilisateur** et des **outils logiciels**.

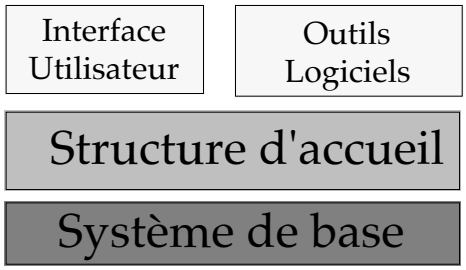


Figure 1.2.b: Atelier de spécification

Notre travail a porté essentiellement sur l'infrastructure d'un atelier de spécification, c'est à dire la structure d'accueil et l'interface utilisateur. L'exposé de notre thèse est concentré sur cet aspect essentiel et nous ne détaillerons donc pas les outils de spécification, d'analyse et de prototypage relevant du domaine applicatif.

> **Définition:** Structure d'accueil
>
> Une **structure d'accueil** est un environnement logiciel qui permet **l'intégration** et **l'enchaînement** d'outils logiciels et la gestion globale des données des outils.

Les projets ou produits actuels (AD/Cycle (IBM), Entreprise 2 (Syséca), AIMS, Foundation (Andersen Consulting) PACLan/X (CGI) [CXP, 1987], CAIS(-A) (US) [CAIS, 1986], PCTE/PCTE+ (Esprit), EAST (Eureka-SFGL:Société Française du Génie Logiciel)) ne développent pas d'atelier logiciel unique apportant une réponse universelle; mais ces projets proposent des solutions à certains problèmes adaptés à un domaine particulier. Ces ateliers proposent des **structures d'accueil** pour **intégrer** de nouveaux outils et s'étendre à d'autres domaines.

L'intégration d'outils dans PCTE passe par l'utilisation d'un modèle de données commun (Object Management System) et des interfaces en cours de normalisation. Cette approche peut restreindre l'intégration d'outils existants puisque ces outils doivent utiliser les mêmes structures de données que la structure d'accueil [Hubert, 1989]. Pour obtenir une intégration plus large, la structure d'accueil doit offrir un niveau d'adaptation (interface) pour autoriser les outils à utiliser leurs propres structures de données. Elle assure la normalisation de la communication des données. Ce niveau rend la structure d'accueil ouverte.

Au demeurant, il n'existe **pas d'atelier de génie logiciel général** intégrant un ensemble d'outils couvrant l'ensemble du cycle de vie du logiciel [Muenier, 1989]. Les raisons en sont la diversité des applications, la multiplicité des méthodes et des langages et l'absence de standards sur lesquels les outils peuvent s'appuyer.

1.2.3. L'interface utilisateur

Avec l'évolution des matériels (disponibilité de postes de travail individuels puissants, banalisation des écran graphiques à haute résolution, utilisation de souris pour la désignation), la définition d'interface de communication adaptées aux utilisateurs devient indispensable.

> **Définition:** Interface utilisateur
>
> L'interface utilisateur est l'ensemble des communications entre l'ordinateur et l'utilisateur, elle permet à l'utilisateur d'entrer des informations et à l'ordinateur de transmettre des informations à l'utilisateur. Une interface utilisateur n'est pas simplement une interface graphique, en fait, il peut y avoir interface utilisateur sans interface visuelle du tout.

L'interface utilisateur est la vue externe qu'ont les utilisateurs de l'atelier. La définition d'interfaces utilisateurs des environnements d'exécution comme PCTE ont fait l'objet de nombreux projets ESPRIT [Hayselden, 1987, Tedd, 1987]. L'interface utilisateur regroupe tous les outils facilitant les communications entre l'usager et les applications qu'il faut exécuter. Pour permettre une utilisation simple d'un l'atelier de spécification, l'interface utilisateur s'exécute sur un poste de travail et il est important que celle-ci soit **intégrée** avec les autres logiciels de son environnement quotidien (courrier électronique, traitement de texte, tableur, téléphone dans un futur proche…). Nous avons délimité deux types d'intégration, la coexistence et la coopération.

La **coexistence** est l'utilisation de plusieurs outils indépendants, par exemple un utilisateur peut se servir d'un tableur puis d'un traitement de texte. Le confort d'utilisation se manifeste par la possibilité de contrôler visuellement plusieurs outils indépendants en même temps [Deguine, 1988].

La **coopération** permet à un utilisateur de faire coopérer des applications entre elles. Elle nécessite des transferts de données entre applications, par exemple, le transfert d'un dessin réalisé à l'aide d'une application graphique vers un traitement de texte. Cette opération peut être manuelle comme on le fait par des opérations de couper/coller ou automatique lorsque l'on maintient un "lien" entre la donnée transférée et sa provenance (pouvant même être d'un autre utilisateur). Cette dernière possibilité est une tendance actuelle dans les interfaces utilisateurs conviviales (MacOS 7.0).

Toutes ces interfaces possèdent les caractéristiques évidentes pour permettre une bonne interaction avec l'utilisateur. Nous citerons simplement ces caractéristiques qui se sont vulgarisées (utilisation d'écran bitmap, souris, icônes, multi-fenêtrages…) dont les origines sont dans le système Star de Xerox [Smith, 1983].

L'interface utilisateur a une très grande importance pour l'ergonomie et la convivialité d'un atelier. Nous développerons son cahier des charges dans le chapitre suivant consacré à l'étude des besoins des utilisateurs.

2. Analyse des besoins des utilisateurs

Notre expérience en système et logiciel nous a montré les besoins des différents utilisateurs vis à vis du logiciel et du système. Nous avons dialogué avec différentes catégories d'utilisateurs: directeurs de recherche, chercheurs, ingénieurs systèmes, étudiants, … Nous pouvons "classer" ces utilisateurs dans trois catégories: **les utilisateurs finals** (les modélisateurs et théoriciens), **les concepteurs d'outils**, **les administrateurs de systèmes et de réseaux**.

- Les utilisateurs de la première catégorie réclament d'abord de bons services et ensuite une interface homme-machine conviviale et homogène avec l'environnement de travail. Ils apprécient aussi que cette interface soit intégrée avec les autres outils qu'ils utilisent (traitement de texte, courrier électronique…).
- Les concepteurs d'outils ont besoin de langages et d'un environnement pour fabriquer et modifier plus rapidement du bon logiciel.
- La dernière catégorie d'utilisateurs, les administrateurs de systèmes et de réseaux, est souvent délaissée dans les outils logiciels. Cette catégorie d'utilisateurs est **indispensable** pour une utilisation efficiente des ressources d'un atelier informatique utilisé à grande échelle. La prise en compte des problèmes d'administration des outils logiciels permet de résoudre les problèmes d'installation, de maintenance des logiciels et de gestion d'environnement multi-utilisateurs (définition des droits d'accès aux données dans le réseau).

De la présentation du projet MARS nous avons déduit qu'un atelier de spécification était composé d'une structure d'accueil supportant des outils logiciels permettant aux utilisateurs de modéliser des problèmes dans différents formalismes. Le schéma ci-dessous (Figure 2.0) fait apparaître quatre points cardinaux qui sont la base d'un atelier de spécification: les utilisateurs, les formalismes, la structure d'accueil et les outils logiciels.

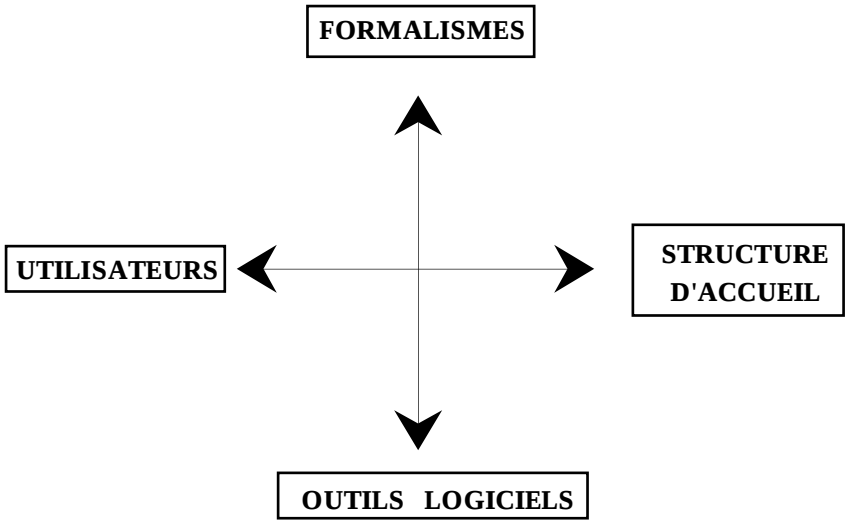


Figure 2.0: Les quatre axes d'un atelier de spécification

Dans cette partie, nous analysons les besoins des différents types d'utilisateurs depuis le théoricien jusqu'au concepteur de logiciels en tirant profit de notre expérience dans les domaines de conception de logiciel [Bernard, 1985, Bernard, 1989, Bernard, 1988] et d'environnement système (inter-connexion de réseau hétérogène).

L'analyse des besoins est le processus visant à établir quelles fonctionnalités le système doit fournir et les contraintes auxquelles il sera soumis [Sommerville, 1988]. Il est important de faire une distinction entre besoins et buts. Un besoin peut être testé tandis qu'un but est une caractéristique plus générale que devrait posséder le système. Par exemple un but pourrait être la facilité d'utilisation d'un système. Ceci ne peut être testé car la facilité d'utilisation est une notion trés subjective. Un besoin associé à ce but pourrait être que toutes les commandes utilisateurs soient activées par menu. Par l'analyse des besoins, nous procèdons à la définition des composants logiciels qui réaliseront les fonctionnalités souhaitées par les utilisateurs.

Nous allons dégager de cette analyse les objectifs jugés primordiaux pour un atelier de spécification. Nous les classerons suivant les quatre axes d'un atelier de spécification.

2.1. Besoins des modélisateurs et théoriciens

2.1.1. Formalismes

L'organisation actuelle des équipements, des personnes et des applications conduit à un fort besoin en réseaux d'ordinateurs [Diaz, 1989]. Il est maintenant clair que la complexité des interactions nécessaires entre les calculateurs impose l'utilisation de méthodologies de conception et que l'étape première de mise en œuvre d'une telle méthodologie consiste à **spécifier** de façon **formelle** le fonctionnement distribué.

Bien spécifier et analyser les problèmes de contrôle est essentiel, dès les premiers stades de la conception. La cohérence avec ces spécifications originales, puis avec leur mise à jour, doit être maintenue tout au long des analyses et réalisations. L'utilisation de modèles formels prenant en compte le parallélisme est indispensable pour assurer la rigueur et la précision des spécifications, puis pour analyser et prouver les propriétés de ces spécifications.

Un vaste champ de recherches consiste à étendre les propriétés théoriques des modèles, construire des outils d'analyse qualitative et quantitative, puis élaborer des méthodes de modélisation et d'analyse pour traiter des systèmes complexes. Il faut de plus, **garantir la cohérence** des phases de spécification, de **validation**, **d'évaluation** et de réalisation.

Objectif 1: Analyser le plus tôt possible

Pour diminuer les coûts et le temps de développement, il est nécessaire d'effectuer des **vérifications sur les modèles** le plus tôt possible.

Pour des systèmes importants, les vérifications, sont nécessaires pour garantir que des situations indésirables ne se produisent pas, ne peuvent se faire directement sur des spécifications complètes. Il est notoire que le coût global d'une erreur est d'autant plus élevé qu'elle est détectée tard dans le cycle de développement. Dans les grands organismes, les coûts de maintenance des logiciels varient entre 50 et 75% des dépenses engagées durant le cycle de vie [Boehm, 1981].

Objectif 2: Multi-formalismes

Un atelier de spécification doit permettre **d'ajouter** de nouveaux **formalismes** et de **gérer** simultanément plusieurs formalismes qui, éventuellement peuvent être mis en relation par des outils logiciels. Un tel atelier gère la notion de **multi-formalismes**.

La complexité croissante des systèmes réels étudiés et les exigences des modélisateurs ont amené à introduire de nombreuses espèces de formalismes et variantes. Ces formalismes engendrent à leur tour de nouveaux types d'objets intermédiaires ou de résultats. Ainsi plusieurs catégories de formalismes (SADT [Marca and Gowan, 1987], CSP, CCS [Milner, 1980], LOTOS [Van Eijk, 1989], ESTELLE [Budkowski, 1988], réseaux de Petri [Brams, 1983]) sont, en fonction des domaines d'application, couramment utilisées et suscitent des avances théoriques régulières. Les langages de spécification ESTELLE, LOTOS et LDS, issus de la normalisation des Techniques de Description Formelle (FDT), sont ceux utilisés pratiquement par les concepteurs. Il existe des environnements spécifiques utilisant LDS (Environnement GEODE [Aubry, 1990]), LOTOS et ESTELLE (projet esprit CEDOS [Courtiat, 1989, Diaz, 1987]).

Une vérification approfondie nécessite un modèle formel synthétique et des outils de preuve automatique de propriétés qui font porter les efforts sur cette difficulté majeure qu'est le contrôle du parallélisme.

La plupart de ces formalismes sont issus de modèles théoriques de haut niveau. Les méthodes d'analyse associées aux formalismes sont de plus en plus ardues et coûteuses. Il faut donc être capable d'étendre rapidement un atelier de spécification par l'ajout de nouveaux formalismes.

Objectif 3: Paramétrage des modèles

Un atelier de spécification doit permettre la description et l'analyse de modèles **paramétrés**.

Il est intéressant de pouvoir spécifier un système indépendamment de son dimensionnement. Ainsi, le même type de système peut être exploité dans des configurations et charges différentes. On considère alors un **modèle paramétré** de ce système. Ce type de modèle devra être supporté par des théories adaptées.

Objectif 4: Correspondance spécification-modèle

Un atelier de spécification doit **gérer** et **conserver** des fonctions (inversibles) de correspondance entre les spécifications et les modèles.

Comme nous l'avons déjà vu dans la figure 1.1.b, il est nécessaire de permettre aux modèlisateurs de **réinterpréter les propriétés du modèle en termes de propriétés des spécifications**. Par exemple par une fonction de correspondance entre le formalisme de haut niveau et le formalisme sous-jacent.

2.1.2. Structure d'accueil

Objectif 5: Multi-utilisateurs

Un atelier de spécification doit prendre en compte la **modèlisation** de système par **plusieurs utilisateurs**.

Des modélisateurs peuvent être amenés à **coopérer** pour modéliser tout un système. Par exemple, il est envisageable dans le cadre d'un projet que les tâches de modélisation soient réparties entre différents partenaires.

Avec une gestion des utilisateurs, on peut envisager un travail simultané sur un même énoncé. La modèlisation coopérative simultanée est un enjeu d'avenir. Les grands modèles nécessiteront la collaboration de groupes d'ingénieurs et seront modifiés durant leur analyse pour créer de nouvelles versions. La cohérence des versions de modèles et à fortiori des sous-modèles relève des notions classique en bases de données.

Objectif 6: Gestion globale des données sur un modèle

Le modélisateur a besoin d'une **gestion globale** des résultats sur une spécification analysée avec différents outils.

Un même modèle peut être analysé avec différents outils, par exemple de vérification et d'évaluation de performance. Il peut être transformé ou compilé en un autre formalisme.
Pour éviter les dérives et maintenir la cohérence globale des différentes phases de la modèlisation à la réalisation, une **gestion globale** sur les différentes données des outils est primordiale.

Objectif 7: Prototypage et animation

Un atelier de spécification doit offrir une infrastructure adaptée au prototypage et à l'animation de systèmes**.**

La réalisation d'un système est une opération coûteuse, longue et source d'erreurs. Des outils doivent permettre d'obtenir des prototypes que l'on peut expérimenter dans le contexte réel avant une réalisation définitive. Elle est aussi source de corrections de spécifications en fonction des contraintes d'architecture et de programmation apparaissant à ce stade.

Il n'y a pas qu'une seule solution à la génération automatique de prototypes et il est souvent impossible de déduire directement une conception de la spécification fonctionnelle [Arthaud and Roan, 1989]. Dans certain domaines, la spécification peut fournir un squelette de la conception. Les problèmes sont dus d'une part à ce qu'une fonction peut être réalisée par plusieurs composants et qu'un composant peut effectuer plusieurs fonctions et d'autres part à la réutilisabilité de composants.

L'expérience [Murphy, 1989] de réalisation en langage Ada de protocoles de communication à partir de diverses spécifications formelles normalisées (LOTOS, ESTELLE, LDS) montre que subsistent des erreurs dans les programmes réalisés. Il faut donc assurer la conformité de la réalisation vis-à-vis de la spécification. Pour qu'il n'y ait pas d'incohérence entre les spécifications initiales et le modèle il est donc préférable que celui ci soit engendré automatiquement.

La **génération automatique** de code est une technique intéressante pour la réalisation rapide de systèmes complexes. Elle est utilisée pour montrer le potentiel du système à l'utilisateur avant l'investissement souvent long et coûteux de l'écriture du système complet. Il est possible de réaliser un prototype afin de permettre à un utilisateur non spécialiste des techniques de conception de systèmes, et donc incapable d'interpréter les spécifications du système, de voir le comportement du système à travers son prototype. Enfin, la génération de code se révèle être souvent un moyen efficace et rapide d'expérimentation de certains aspects spécifiques d'un système. Il faut apporter des aides pour effectuer la génération en fonction de la grande variété des architectures nouvelles qui apparaissent rapidement. Il faut donc une **génération automatique configurable** suivant les types d'architectures et de systèmes. L'utilisation d'une architecture distribuée hétérogène peut faciliter la génération de code pour un système cible et l'intégration du système cible dans l'atelier. On peut donc obtenir des simulations réelles [Bütler, 1990]: les animations.

Un **système d'animation** permet de voir le comportement d'un système pendant son exécution [Bustard, 1989]. Le but d'une telle animation est d'aider l'utilisateur à comprendre les structures et l'activité de niveau logique de son système. Cette visualisation peut aller du simple suivi du code exécuté à une représentation graphique.

Objectif 8: Multi-linguismes

Un atelier de spécification doit prendre en compte la notion de **multi-linguismes** pour permettre une **utilisation internationale** de **l'atelier** et de ces **outils**.

De nombreuses sociétés sont multinationales ou ont des ingénieurs de plusieurs pays. De nombreux projets internationaux font échanger des spécifications et modèles entre organismes des différents pays. Les comités de normalisation sont internationaux.

Pour des raisons ergonomiques, il faut traiter le **multi-linguismes** au niveau des modèles comme des outils. La gestion du multi-linguismes doit être entièrement prise en charge par l'environnement logiciel de l'atelier. Ainsi, des outils de modèlisation réalisés en Allemagne pourront être exploités avec des modèles décrits en Français. Cette internationalisation de l'atelier est prévue pour se rapprocher des habitudes de travail des utilisateurs: c'est à dire que les noms des objets sont exprimés dans la langue de l'utilisateur.

Contrairement aux apparences, le multi-linguismes ne relève pas que de l'interface utilisateur mais il faut l'intégrer au niveau de la structure d'accueil.

2.1.3. Utilisateurs

Nous avons remarqué que dans les AGL actuels les interfaces utilisateurs sont souvent disparates [Giavitto, et al., 1989]. Il faudrait que tous les outils aient un **comportement similaire** en ce qui concerne leurs interfaces externes, styles d'interaction, niveau et facilité d'aide en ligne…. L'interface utilisateur d'un AGL doit être **conviviale et homogène** [Roubine, 1989].

Objectif 9: Interface utilisateur homogène et privilégiée

L'interface utilisateur d'un atelier de spécification doit offrir une **simplicité d'utilisation** des outils logiciels et rendre prioritaire les actions de l'utilisateur.

Les modélisateurs, comme les théoriciens, ont besoin d'un outil se rapprochant le plus possible de leur façon habituelle d'aborder les problèmes. Beaucoup d'outils existent mais leur utilisation est jugée rebutante, parce qu'elle est souvent trop complexe. Ces utilisateurs ont besoin d'une **interface utilisateur simple et homogène**, c'est à dire facile à apprendre et à utiliser, et permettant de passer facilement d'une application à une autre. Notre expérience en logiciel, nous a montré que l'interface utilisateur doit être **privilégiée vis à vis des traitements** de ces données afin que l'utilisateur soit immédiatement informé (feed back) des conséquences potentielles et des résultats de ses actions.

Les utilisateurs insistent sur **l'unicité de l'interface utilisateur**. Ils veulent pouvoir voir sur le même écran l'ensemble des éléments du système réalisé. La juxtaposition de logiciels avec des interfaces hétéroclites amène différentes sources d'oubli et d'erreur de la part de l'usager. La normalisation de ces interfaces [Coutaz, 1985, Shneiderman, 1987] améliore le confort de l'usager qui s'habitue à des modes d'emploi similaires (consistance) et se consacre à l'exploitation des fonctionnalités du logiciel plutôt qu'à son mode d'emploi [Karsenty, 1987]. Elle réduit donc la charge cognitive de l'utilisateur.

Objectif 10: Interface utilisateur graphique et textuelle

L'interface utilisateur d'un atelier de spécification doit être adaptée aux représentations utilisées par les modélisateurs. L'interface utilisateur doit permettre au modélisateur, **l'édition des modèles** et la **visualisation des résultats**, sous forme **graphique** et **textuelle**.

De nombreux modèles de haut niveau ont une représentation graphique (Réseaux de Petri, SADT) avec des attributs textuels (noms, valeurs, prédicats) où encore une représentation totalement textuelle (ESTELLE, LDS [Aubry and Bougro, 1990]). Le modélisateur demande donc des outils à la fois de construction graphique, d'analyse syntaxique et sémantique. Certains modèles peuvent être décrits suivant les deux représentations, il est donc important de pouvoir changer de représentation (ELVIS [Camacho, 1989]). Les utilisateurs d'un atelier de spécification ont besoin d'**outils d'édition** de leurs modèles, ainsi que de leurs résultats. Il faut adopter une **représentation graphique des résultats**, soit dans le modèle d'entrée, soit sous une autre forme (ex: histogrammes…). Ainsi, l'interprétation des résultats est facilitée.

Objectif 11: Environnement logiciel

Un atelier de spécification doit offrir aux utilisateurs un **environnement logiciel intégré** permettant **l'enchaînement** automatique des outils.

Les outils mis à la disposition du modélisateur seraient inefficaces sans un **environnement logiciel** complet. Un souhait des utilisateurs porte sur **l'intégration** des outils aussi bien dans leur environnement de travail (tableur, traitement de texte, …) que des services de spécification (validation, génération de code…). Mais dans notre cas, l'intégration des outils n'est pas suffisante car l'utilisateur se retrouve en face d'un ensemble d'outils plus ou moins hétéroclites. Pour une utilisation optimale de ces outils, l'utilisateur doit pouvoir les **enchaîner** les uns aux autres (compatibilité), de manière implicite ou explicite.

L'ensemble de ces outils et des phases de travail doit être contrôlé par un environnement qui permet au modélisateur de suivre toutes les phases de son travail, de la spécification à la réalisation d'un prototype. Un **environnement** commun d'interfaces et de traitements facilite la manipulation des modèles appropriés à chaque étape et garantit la cohérence des analyses lors des modifications de modèles.

Objectif 12: Multi-sessions, parallélisme des traitements

L'interface utilisateur doit faciliter le travail du modèlisateur en lui offrant la possibilité de lancer plusieurs **traitements en parallèle**. L'atelier doit aussi prendre en compte le lancement d'outils en "arrière plan".

La plupart des outils logiciels permettant la validation, la génération de code… sur des modèles de l'utilisateur demandent des ressources de mémoire et de calcul importantes. Le temps de calcul est souvent très long. Il faut donc permettre aux utilisateurs de ne pas être bloqués dans l'attente de fin d'exécution de l'outil lancé. Il faudra prendre en compte la possibilité de **lancer des outils en "arrière plan**" et de pouvoir récupérer ultérieurement les résultats. Une autre solution consiste à pouvoir lancer **différents traitements simultanément** sur différents modèles. Par exemple, le modélisateur doit pouvoir demander une génération de prototype sur un premier modèle et continuer à travailler sur un autre modèle sans être ralenti. La modification "esthétique" d'une description graphique ou son impression ne doit pas empêcher son analyse en parallèle. Ainsi le travail du modélisateur sera plus efficace.

2.1.4. Outils logiciels

Objectif 13: Fournir une puissance de calcul

Un atelier de spécification doit **fournir une puissance de calcul suffisante** pour permettre l'exécution des outils logiciels.

L'analyse de systèmes conduit rapidement à une explosion combinatoire entraînée par la prise en compte dans le modèle de la répartition des traitements et des pannes. Ces outils d'analyse demandent une **forte puissance de calcul** (calcul et mémoire) et de grandes capacités de stockage.

Objectif 14: Communication entre les outils

La **communication des outils** durant les différentes étapes, de la modèlisation à la génération de prototype est un élément essentiel pour garantir le produit ainsi réalisé**.**

L'objectif du projet étant de permettre le prototypage à partir d'une spécification validée, il faut interpréter les résultats d'analyse ou de test en termes de propriétés de spécification. L'analyse des spécifications initiales n'est pas suffisante, car on souhaite des garanties sur le produit final réalisé (fiabilité des résultats). Or, de multiples dérives par rapport aux spécifications initiales sont introduites par les phases intermédiaires des spécifications détaillées, de programmation, d'intégration, de tests et de mises à jour. Il est donc nécessaire de disposer, pour chaque phase, **d'outils communiquant** entre eux.
De plus, lorsqu'on dispose de différents logiciels travaillant sur un même formalisme, on est confronté aux problèmes successifs de représentation externe unifiée, d'introduction simultanée d'une même spécification pour différents logiciels et de transfert, à un niveau élémentaire, de résultats entre les applications.
La multiplication des techniques de spécification formelle et l'essor des techniques d'analyse rend très difficile le partage et la communication de résultats. Il faut des mécanismes et des **moyens** de **communication** entre les outils des différents modèles. Un **langage interne** de communication permet, non seulement, la description sous une représentation intermédiaire unique des spécifications des concepteurs, mais aussi la saisie simultanée et le maintien de la cohérence entre les spécifications et ses différentes représentations internes pour chaque logiciel.

Définition: Langage interne de communication

La communication par messages semble adaptée aux besoins d'un atelier de spécification. Un **langage interne** de représentation de données est le moyen essentiel de transfert d'informations entre les outils. Ce langage permet de traiter les données en entrée et en sortie des outils (**gestion globale**).

Le moyen de communication doit être externe aux outils et être fourni par la structure d'accueil.

2.2. Besoins des concepteurs d'outils

Dans ce paragraphe, notre discours ne sera pas orienté uniquement vers la programmation mais aussi vers l'efficacité des concepteurs d'outils et vers les facteurs de qualité du logiciel.

2.2.1. Formalismes

Objectif 15: Neutralité vis à vis des formalismes

Un atelier de spécification doit **gérer** de **manière externe** (explicite) **les formalismes**. L'ensemble des connaissances relatives à chaque formalisme doit être décrit de manière externe à l'atelier indépendamment des applications.

Selon les désirs de l'utilisateur final, une spécification peut être exprimée à l'aide de plusieurs formalismes. Il est donc important pour le concepteur d'outils de pouvoir gérer des **changements de formalismes** donc de représentations. La nécessité d'étendre et de modifier les formalismes coûte des modifications inextricables de structures impliquant une **réécriture de nombreux programmes**. La cause majeure de ce coût réside dans le fait que les formalismes sont presque toujours exploités implicitement. C'est à dire qu'ils sont connus uniquement au niveau des applications qui les manipulent. Pour remédier à ce problème, dans un atelier de spécification, les formalismes doivent être **explicites**. L'objectif est de rendre ainsi l'atelier **neutre** vis à vis des formalismes.

La **qualité** d'un logiciel (outils) est directement liée à celle de sa conception. La qualité constitue la "mise en œuvre d'un ensemble approprié de dispositions préalables et systématiques destinées à donner confiance en l'obtention de la qualité requise" (Norme AFNOR X50109). Malgré la difficulté d'apprécier la qualité d'un logiciel, on s'accorde aujourd'hui à dire qu'elle est appréciable selon deux points de vue différents introduits par Mc Call [Mc Call, 1977]: la qualité externe et la qualité interne.

Les facteurs de la qualité externe sont ceux que perçoivent l'acquéreur ou l'utilisateur du produit final et en particulier la portabilité, **l'efficacité**, la fiabilité, la maniabilité et **l'extensibilité**. Les facteurs internes sont les attributs du logiciel permettant à l'informaticien d'obtenir les facteurs externes désirés, par exemple lisibilité et **modularité** des programmes.

Les **méthodes** et techniques de génie logiciel ont pour but de fournir aux ingénieurs et analystes des outils leur permettant de développer des logiciels d'une **qualité** [Galinier, 1988] en constante amélioration et avec une productivité croissante. La réduction des coûts de fabrication du logiciel n'est pas l'unique objectif du génie logiciel. Un objectif majeur est de réaliser des logiciels de plus en plus importants et complexes tout en garantissant une qualité et une **fiabilité** accrues. Il faut donc utiliser des **méthodes de spécification**, de développement et de mise au point. Chaque phase du cycle de vie du logiciel [Boehm, 1981] possède des méthodes spécifiques et dépend surtout des domaines d'applications. Brian Henderson-Sellers [Henderson-Sellers, 1990] compare plusieurs méthodes basée sur les flots de données, les structures de données et la décomposition orientée objet. Les méthodes doivent être supportées par un environnement logiciel. On assure la qualité par l'emploi obligatoire d'une méthode qui doit être décrite de manière externe. La description des différentes méthodes utilisées tout au long du cycle de vie du logiciel doit être décrit par un **méta-modèle** de définition [Giavitto, et al., 1989].

Nous pouvons remarquer dès maintenant les conséquences de l'utilisation d'un **méta-modèle** sur l'ouverture de l'atelier de spécification.

2.2.2. Structure d'accueil

Objectif 16: Ouverture

Un atelier de spécification doit être un environnement logiciel **ouvert** facilitant **l'intégration** de nouveaux outils.

La conception et la réalisation d'outils logiciels est une tâche laborieuse et longue. En dépit de ce travail considérable, les outils logiciels restent figés sur l'état de la théorie au moment de leur programmation et ne sont étendus que trop lentement. Le décalage croissant entre les théories et leurs mises en œuvre peut aboutir à des ensembles hétérogènes d'outils dépassés. Répercuter systématiquement les avances théoriques demande un travail d'analyse et de programmation fastidieux compte tenu de la variété et de la complexité des modèles proposés. Un atelier de spécification doit être **ouvert** pour faciliter **l'ajout de nouveaux outils**.

2.2.3. Utilisateurs

Objectif 17: Interface utilisateur générique

Un atelier de spécification fournit aux concepteurs une **interface générique**. Les concepteurs ne se préoccupent plus des problèmes d'édition et de visualisation de leurs données.

La réalisation d'interfaces utilisateurs est une tâche fastidieuse qui se pose à chaque nouvelle application: saisie des données, fonctionnalités (menus, diagnostics, aides), récupération des résultats. La normalisation permet de réaliser des interfaces paramétrées que le développeur particularise pour chaque application par des descriptifs [Masaï, 1989]. Le concepteur d'outils peut donc utiliser de manière standard des interfaces déjà expérimentées par d'autres concepteurs. Ils peuvent ainsi consacrer plus de temps aux fonctionnalités et à l'efficacité de leurs outils.

Pour **normaliser la saisie graphique ou textuelle** et la visualisation des résultats, nous pensons qu'il faut fournir une application graphique générique intégrant un noyau gérant toutes les interactions avec l'utilisateur, quel que soient le formalisme utilisé et le traitement exécuté. Cette application sera éventuellement particularisée lors de l'introduction d'un nouveau formalisme ou à l'intégration d'un nouveau traitement. L'interface utilisateur devient ainsi une donnée et non plus une partie du code de l'application.

Objectif 18: Séparation physique de l'interface utilisateur et des outils

L'interface utilisateur d'un atelier de spécification **ne fait pas partie** du **code des outils logiciels**.

La notion d'interface utilisateur générique induit un principe important: l'interface utilisateur et l'outil seront deux composantes modulaires distinctes. Nous obtiendrons ainsi une **séparation du** code de l'**interface utilisateur** et des **outils**.

Objectif 19: Vérifications syntaxique et sémantique

Les **vérifications syntaxique** et **sémantique** des modèles de l'utilisateur doivent être effectuées le plus tôt possible. Un atelier de spécification doit permettre la vérification dès **l'introduction** des modèles de la syntaxe et de la sémantique.

Conformément à l'objectif 1, il est nécessaire de détecter interactivement les erreurs immédiates le plus tôt possible et ceci pour tout outil disponible dans l'atelier. Il s'agit donc, à nouveau, d'une tâche systématique et fastidieuse pour laquelle nous pensons qu'il faut aider les concepteurs. Il faut donc fournir aux concepteurs d'outils des programmes de **vérifications** syntaxiques et sémantiques associés à chaque modèle théorique. Cet objectif implique que l'interface utilisateur soit capable de saisir interactivement la syntaxe des modèles de l'utilisateur. Ceci tout en respectant le caractère générique de l'interface utilisateur.

2.2.4. Outils logiciels

Objectif 20: Pas de limitation dans les types de langages

Un atelier de spécification **ne doit pas être lié à un type de langage** de programmation.

Il ne faut en aucun cas restreindre le concepteur des outils à des langages de programmation, l'atelier doit être capable d'accepter des langages compilés ou interprétés.

Objectif 21: Utilisation d'un système expert

Pour diminuer les délais de conception des outils et pour y introduire les heuristiques de leurs algorithmes, il faut faire appel à la **programmation logique**. **L'intégration d'un système expert** dans l'atelier doit faciliter ce type de programmation.

Dans notre domaine de recherche, les concepteurs de programmes ont souvent besoin d'introduire des heuristiques et les programmeurs se heurtent souvent à des problèmes de combinatoire. L'utilisation d'un système expert faisant coopérer des algorithmes et des règles de déductions semble nécessaire pour la conception d'outils de modèlisation et d'analyse. L'approche système expert permet de décrire les méthodes d'analyses choisies sous forme de règles et d'autre part de décrire explicitement les représentations adaptées pour les données sous forme de faits. Cette approche permet aussi un enchaînement automatique de méthodes de preuves dirigées par les données et l'interprétation automatique de résultats. Elle facilite l'intégration de théorèmes complexes à l'aide de règles très proches de leurs définitions mathématiques. Ce type de programmation ne remet pas en cause l'utilisation de la programmation classique pour certaines applications. Cette technique permet un développement rapide d'applications et de maquettes d'outils d'analyse.

Objectif 22: Réutilisation des outils entre eux

Un atelier de spécification doit apporter une réponse au problème de **réutilisation** des outils.

Pour concevoir une nouvelle application sur un formalisme de haut niveau, il est intéressant de pouvoir réutiliser des algorithmes travaillant sur des formalismes proches. Pour cela il faut construire une "bibliothèque d'algorithmes et de structures de données" et disposer de langages permettant de les mettre en œuvre commodément.

Pour fabriquer moins de logiciels, il faut faciliter la **réutilisation** des outils (algorithmes, programmes) existants; mais l'apparition de nouveaux modèles pose même des problèmes de réutilisation et de compatibilité des outils déjà réalisés. Il faut donc être capable de fournir aux utilisateurs, l'inventaire détaillé des connaissances et des outils disponibles (outil de type **browser**). Il ne s'agit pas là d'une description au niveau de la programmation comme dans les ateliers de génie logiciel mais plutôt de l'existence de services effectuant un certain calcul sur un formalisme. Par exemple, un concepteur d'outils peut chercher, pour son application, un service de recherche de composantes fortement connexes d'un graphe.

Dans un atelier de spécification, la réutilisation passe obligatoirement par une **communication de données entre les outils** et la possibilité à un outil d'exécuter de manière transparente, toute autre application de l'atelier.

2.3. Besoins des administrateurs

L'administration n'est pas un besoin: c'est la conséquence des besoins exprimés plus haut (gestion multi-utilisateurs, multi-formalismes etc…), nous n'en déduirons donc aucun objectif.
La tâche administration doit faciliter l'**extensibilité** de l'atelier de spécification en permettant les évolutions et les adaptations à des modifications ou à des extensions de ses spécifications.

Pour gérer, l'environnement système, les modèles théoriques, les utilisateurs et leurs données, il est nécessaire d'avoir des **outils d'administration**. Il existe plusieurs administrateurs: l'administrateur système et réseau, l'administrateur des formalismes, l'administrateur des applications. Nous les décrivons maintenant.

2.3.1. Formalismes

Faire cohabiter différents formalismes et applications dans un même environnement nécessite de gérer leurs relations éventuelles. Naturellement, la gestion des différents formalismes et l'exploitation des applications constituent deux tâches d'administration distinctes.

2.3.2. Structure d'accueil

Définition: Gestion de configuration

La **gestion de configuration** d'un atelier de spécification est le contrôle des évolutions de systèmes complexes. Elle doit être mise en œuvre par l'administrateur de systèmes et de réseaux grâce à des outils de descriptions des système des sites d'exécution (noms, types de machine, logiciels…).

La gestion de configuration est un problème qui n'apparaît qu'à la deuxième génération de logiciels complexes. C'est à dire que c'est un problème qui n'existait pas lorsque les logiciels étaient autonomes et ne s'exécutaient que sur un poste de travail destiné, parfois à un unique utilisateur.

En tant que responsables d'un site informatique, nous avons souvent été confrontés à l'installation de logiciels de provenances diverses (logiciels commerciaux ou du domaine public) qui nous ont souvent montré la difficulté des **procédures d'installation** et de mise en œuvre. Ces difficultés sont encore plus grandes dans une architecture distribuée (interconnexion de réseaux) de machines hétérogènes. Nous avons souvent constaté que ces installations devaient être implantées dans des endroits prédéterminés de l'espace de fichiers du système distribué remettant ainsi en cause le système de fichiers initial.

Lors des mises à niveau de matériels (évolution du système d'exploitation), pour assurer la **maintenance** des outils, l'administrateur de l'atelier doit être capable de les régénérer sans la présence des concepteurs d'applications.

Dans une **architecture distribuée**, le parc de machines est amené à évoluer en nombre, type et en version de système. Les environnements logiciels devraient prendre en compte automatiquement ces évolutions de configuration. La gestion de configuration d'un tel environnement doit être mise en œuvre par l'administrateur de systèmes et de réseaux grâce à des outils de descriptions système des sites d'exécution (noms, types de machine, logiciels…). D'autres outils doivent permettre la recherche des problèmes logiciels et faciliter le suivi des performances de l'environnement (tuning) et d'optimiser le système par rapport aux ressources existantes. L'administrateur doit pouvoir fournir aux concepteurs d'applications des méthodes propres à rendre transparente cette architecture **hétérogène**.

La gestion de configuration peut être réalisée à l'aide d'outils basés sur les graphes (réseaux locaux) [Heimbigner, 1988].

2.3.3. Utilisateurs

Pour que les utilisateurs n'aient accès qu'aux outils autorisés versions de services dûment testées, l'administrateur doit mettre en place une hiérarchie d'utilisateurs. Un des problèmes à résoudre est la **gestion de droits d'accès multi-utilisateurs**. L'administration d'un atelier de spécification est donc étroitement liée à l'administration système.

Pour permettre la coopération entre modélisateurs et pour assurer une gestion cohérente des espaces de fichiers, les outils ne doivent pas gérer eux-même l'emplacement de leurs données. C'est à l'administrateur de mettre en place l'accés au système de fichier.

2.3.4. Outils logiciels

Pour augmenter la réutilisabilité des outils, notre expérience nous a montré qu'il est nécessaire d'obtenir des concepteurs d'outils, une documentation précise des services fournis ainsi qu'une description de leurs résultats. Dans la mesure du possible, il faut définir un langage décrivant de manière formelle les résultats. L'administration d'un nombre croissant d'outils disponibles nous a amenés à définir une **documentation** standard pour chaque outil. Cette documentation doit contenir en particulier l'ensemble des résultats produits par les outils (**description des résultats**). Ces résultats peuvent éventuellement être des données d'entrée pour d'autres outils. Ces enchaînements d'outils sont une partie de l'expertise des modélisateurs et des théoriciens.

Un facteur d'acceptation d'un service est **l'adéquation et la qualité des services** offerts. Lorsque l'administrateur des services intègre un nouveau service, il doit s'assurer que celui-ci offre bien le service décrit (**correction**). Pour cela l'administrateur et le concepteur doivent disposer d'outils de tests (aide à la validation, procédures de tests, modes traces, déverminage).

Il faut remarquer que dans notre cas le temps de réponse (attente d'un utilisateur) d'un outil de spécification n'est pas du ressort de l'administrateur mais il contribue à fournir aux utilisateurs une **qualité de service** par une forte disponibilité des services. Par exemple l'administrateur devra assurer la disponibilité d'un service même en cas de panne temporaire du site le supportant. Ceci exige des mécanismes de reprise.

3. Analyse des besoins en systèmes

3.1. Ouverture et extensibilité

Objectif 23: Atelier ouvert et extensible

Un atelier de spécification doit être **ouvert** et **extensible** en terme de génie logiciel.

La construction d'un atelier de génie logiciel est comparable à celle d'un mécano. La base doit être solide pour supporter l'ajout de nouveaux éléments. Les pièces ajoutées s'insèrent aux bons endroits afin de compléter l'objet réalisé par des éléments de base. Dans notre cas, l'AGL a pour base une **structure d'accueil** facilitant l'ajout d'éléments logiciels qui représentent les outils intégrés. Compléter le mécano correspond à enrichir les connaissances de l'atelier. Un environnement de génie logiciel global sera donc constitué par l'intégration d'outils d'origines différentes. L'ouverture de l'environnement devient donc une propriété déterminante. Nous avons remarqué qu'un des besoins à court terme est le plus souvent l'intégration des outils existants [Muenier, 1989]. Il s'agit, dans un atelier de spécifications, de **récupérer** le maximum de **logiciels** déjà écrits. L'intégration devra aussi, dans la mesure du possible, permettre l'ajout d'outils qui ne sont pas ouverts.

Définition: Ouverture

L'ouverture est la capacité d'un environnement à permettre l'intégration de nouveaux outils. Elle concerne les ajouts de nouvelles applications de manière externe.

Il nous semble donc important que la **structure d'accueil** facilite l'**ouverture** de l'atelier. Contrairement au mécano qui ne peut accepter que des pièce "de mécano", l'**ouverture** sera d'autant plus grande s'il n'y a pas d'à priori sur les outils (outils non prévus au départ pour l'intégration).

Définition: Extensibilité

L'extensibilité est la facilité avec laquelle le système peut être adapté à une modification ou à une extension de ses spécifications. Elle concerne des changements effectués de manière interne.

Si un atelier est extensible [Meyer, 1988], son cahier des charges peut évoluer sans perdre tous les outils intégrés et sans les modifier. Il nous semble primordial que des modifications du cahier des charges n'aient une influence que sur **l'architecture** de la structure d'accueil.

3.2. Répartition et tolérance aux pannes

Notre environnement de travail aussi bien en recherche qu'en utilisation (traitement de texte…) est constitué d'un parc de micro-ordinateurs (Macintosh) reliés en **réseau**, connectés à un réseau de machines sous Unix (Sun). Ce dernier réseau étant lui même interconnecté à un réseau principal nous donnant accès à l'extérieur (réseau fnet et bitnet). Il a fallu rendre transparents les accès aux données des utilisateurs aussi bien sur les micro-ordinateurs que sur les stations de travail. Cette expérience, nous a montré les avantages de l'utilisation couplée de micro-ordinateurs facilitant l'utilisation et de stations de travail pour la puissance de calcul.

Dans notre environnement de travail, les micro-ordinateurs servent d'interface utilisateur et nous avons remarqué la faiblesse des micro-ordinateurs vis à vis de la **tolérance aux pannes**.

La plupart des systèmes à l'heure actuelle ne permettant pas de rattraper sur ces machines une erreur logicielle, il faut donc penser à mettre en place des moyens de sauvegarde d'état des outils pour ne pas perdre les calculs en cours et rendre le travail plus sûr. La sûreté doit être prise, dans notre cas, comme une combinaison de fiabilité, de disponibilité et sécurité. Il faudra donc spécifier un mécanisme tolérant aux pannes capable de gérer des points de reprise dans un environnement distribué sujet à des défaillances.

Dans le cas où l'on dispose de plusieurs serveurs de calcul, il faudra assurer l'accès aux différents services par une méthode de duplication des programmes et de données sur les différents serveurs de l'environnement distribué.

Les éventuelles défaillances (pannes de sites, ruptures transitoires de liaisons, …) ne doivent pas introduire d'incohérence et il est nécessaire d'assurer un fonctionnement minimal de consultation (même en mode dégradé). La tolérance aux pannes [Laprie, 1984] est un problème crucial, que l'atelier doit intégrer en faisant appel aux services des systèmes d'exploitation support. On tient compte usuellement de deux sortes de défaillances, la défaillance de l'interface utilisateur et la défaillance de machines supportant l'atelier. Assurer une **sûreté de fonctionnement**, dans un tel environnement est un travail difficile auquel nous avons proposé des solutions. Elles sont implémentées dans le prototype AMI.

Objectif 24: Utilisation coopérative de réseaux de machines

L'architecture d'un atelier de spécification doit prendre en compte la notion de réseau de machines coopérantes pour utiliser au mieux les caractéristiques des machines par rapport au travail demandé.

D'autre part, un travail sur réseau est plus efficace, la mise en commun des serveurs garantit plus de disponibilité et prépare un travail coopératif.

Un AGL gère la notion de **groupe d'utilisateurs** afin de permettre la coopération entre les différentes personnes utilisant l'atelier. Les utilisateurs peuvent travailler en même temps sur plusieurs outils à la fois. Ces outils sont souvent consommateurs de grosses **puissances de calcul** et de mémoire de masse. L'atelier doit donc bénéficier d'une très forte puissance de calcul tout en garantissant un **temps de réponse** uniforme. Une architecture distribuée pour un AGL [Bourgeois, 1988] permet d'**optimiser** l'utilisation des ressources par la répartition des puissances de calcul. L'architecture de cet atelier doit naturellement se construire dans un environnement distribué hétérogène.

Une telle architecture logicielle et matérielle permet une ouverture tout en conservant la notion de **stabilité**.

3.3. Architecture hétérogène

L'architecture d'un atelier de spécification doit permettre d'assurer une certaine pérennité des investissements et la possibilité d'intégrer les **avances technologiques**.

Dans le domaine des matériels informatiques, nous constatons que les avancés technologiques apportent une remise en cause tous les six mois (tous les constructeurs informatiques comme Sun, Apple, Digital, IBM, annoncent de nouveaux produits tous les six mois). A l'opposé, la production de logiciels est incapable de suivre ce rythme. Les **évolutions des systèmes d'exploitation** liées aux évolutions de matériels sont **extrêmement pénalisantes** pour les équipes de développements (par exemple le passage de Sun3 en Sun4), cela implique une description des machines et des systèmes.

> **Définition:** Efficacité d'un atelier
>
> **L'efficacité d'un atelier** dépend de sa capacité à s'adapter à l'**évolution** non seulement des outils mais aussi du **matériel**.

Pour prendre en compte, ces problèmes d'évolutions de matériels dans un AGL, une solution consiste à utiliser une **architecture distribué hétérogène**. Ce type d'architecture offre des avantages comme l'élimination du portage de logiciels [XICH, 1988]. En effet, les outils utilisant des environnements spécifiques liés à un type de matériel ne pourraient pas être intégrés dans l'atelier si son architecture n'était pas distribuée. Or la génération actuelle des ateliers de génie logiciel oublie souvent cette distribution qui permet aussi la "**récupération**" de **matériel**.

Nous avons envisagé une architecture distribuée hétérogène qui supporte le fonctionnement de l'atelier. Le chapitre précédent nous a montré qu'une telle architecture était fondamentale pour un atelier de spécification.

Les principales caractéristiques d'un tel environnement système sont:

- l'utilisation de réseau de machines,
- le support d'environnement hétérogène d'exécution,
- le traitement coopératif,
- un système d'exploitation permettant la distribution.

4. Synthèse

L'analyse de nos expériences de réalisation et d'exploitation d'outils de modèlisation nous a amené à compléter les besoins initiaux des utilisateurs par des critères concernant les concepteurs de logiciels et les administrateurs. Ces nouveaux critères sont des apports importants pour le cahier des charges de l'environnement.

Dans l'étude précédente, nous avons fait apparaître des objectifs, à nos yeux essentiels, d'un atelier de spécification. Ces objectifs ont été classés selon quatre axes qui sont la base d'un atelier de spécification: les utilisateurs, les formalismes, la plate-forme et les outils logiciels. Ces quatre points contribuent à l'ouverture de l'atelier pour obtenir un atelier multi-utilisateurs fonctionnant dans un environnement distribué hétérogène, permettant le suivi de notre domaine de recherche: un atelier de Modélisation, d'Analyse et de Réalisation de Système MARS.

Nous regroupons l'ensemble des objectifs, mis en évidence lors de l'analyse des besoins, en thèmes fonctionnels (Figure 4.0):

Formalismes de haut niveau	**Objectif 2:** Multi-formalismes **Objectif 3:** Paramétrage des modèles **Objectif 4:** Correspondance spécification-modèle **Objectif 15:** Neutralité vis à vis des formalismes
Edition des données et représentation des résultats	**Objectif 10:** Interface utilisateur graphique et textuelle **Objectif 19:** Vérifications syntaxique et sémantique
Gestion globales des données, coopération entre utilisateurs	**Objectif 4:** Correspondance spécification-modèle **Objectif 5**: Multi-utilisateurs **Objectif 6:** Gestion globale des données sur un modèle **Objectif 8:** Multi-linguismes
Puissance de calcul, multitraitements,	**Objectif 12**: Multi-sessions, parallélisme des traitements **Objectif 13:** Fournir une puissance de calcul **Objectif 24:** Utilisation coopérative de réseaux de machines

Ergonomie et facilité d'utilisation	**Objectif 9:** Interface utilisateur homogène et privilégiée **Objectif 11:** Environnement logiciel
Qualité: qualité du logiciel, génération de prototype	**Objectif 1:** Analyser le plus tôt possible **Objectif 7:** Prototypage et animation
Extensibilité, réutilisation, communication et intégration	**Objectif 14:** Communication entre les outils **Objectif 16:** Ouverture **Objectif 17:** Interface utilisateur générique **Objectif 18:** Séparation physique de l'interface utilisateur et des outils **Objectif 20:** Pas de limitation dans les types de langages **Objectif 21:** Utilisation d'un système expert **Objectif 22:** Réutilisation des outils entre eux **Objectif 23:** Atelier extensible

Figure 4.0: Les objectifs regroupés en thèmes.

5. Composantes systèmes d'un atelier de spécification

5.1. Introduction

Afin de situer le cadre de l'étude, nous définissons notre vision de l'environnement système de l'**atelier de spécification MARS**, nous présentons les concepts fondamentaux et en déduisons les différentes composantes. Une composante résout un ou plusieurs besoins énoncés dans l'étude préalable (I.1).

Les composantes que nous avons retenues dans cette étude sont des composantes orientées **système** pour la description d'un environnement de spécifications.

L'environnement de spécifications est une **structure d'accueil** d'applications portant sur différents formalismes théoriques. Les composantes systèmes sont divisées en deux classes: les composantes liées à **l'infrastructure système** et celles imposées par le **domaine applicatif**. Le côté applicatif du projet MARS n'est bien sur que la partie émergée de l'iceberg, l'infrastructure système étant la partie la plus importante de notre travail.

Le cadre de notre étude porte essentiellement sur l'infrastructure système. Certaines composantes (**composantes résultantes**) ne sont pas nécessaires à l'élaboration d'un tel environnement mais sont une conséquence directe de la définition des composantes système. Par exemple, la composante résultante "outils d'administration" est une conséquence des composantes "gestion des formalismes" et "répartition".

Nous présentons dans ce chapitre les éléments systèmes de base d'un atelier de spécification. Les **composantes** que nous avons mises en évidence proviennent de la décomposition système des ateliers de génie logiciel.

Structure d'accueil	Une structure d'accueil est l'environnement logiciel qui permet l'intégration et l'enchaînement d'outils logiciels et la gestion globale des données des outils.
Interface utilisateur	L'interface utilisateur est la vue externe qu'ont les utilisateurs de l'atelier. Elle regroupe tous les outils facilitant les communications entre l'usager et les applications qu'il faut exécuter
Station de travail virtuelle	L'interface utilisateur dispose d'un système de multi-fenêtrages local qui est rendu transparent aux applications et à la plate-forme par la notion de station de travail virtuelle.
Gestion des formalismes	Un environnement de spécification doit prendre en charge la gestion des formalismes des spécifications des utilisateurs. Le problème de l'utilisateur est décrit suivant un certain formalisme. Un atelier de spécifications possède un méta-modèle pour la description des formalismes.
Répartition	Les principales caractéristiques systèmes d'un environnement de spécifications sont le support d'environnement hétérogène d'exécution (répartition).
Système de gestion de fichiers virtuels	Dans un environnement distribué, il est nécessaire de gérer l'accès transparent aux ressources "disques" du réseau. La gestion des fichiers virtuels assure un partage d'informations entre sites de manière transparente.
Multi-utilisateurs	La notion de groupes d'utilisateurs (multi-utilisateurs) doit être gérée afin de permettre la coopération entre les différents utilisateurs.
Outils d'administration	Il est indispensable, pour un tel environnement, de disposer d'outils d'administration permettant la gestion de configuration (sites, logiciels, fichiers) ainsi que des données de l'atelier (formalismes, outils, utilisateurs, droits ...). Une conception moderne d'un environnement logiciel implique que toutes les questions d'administration soient non seulement gérées par des applications indépendante mais de plus que les données correspondantes soient externes et explicites.

Méthode de conception interne	Il est nécessaire de fournir au concepteur de logiciels une méthode de conception pour permettre l'ajout de nouveaux outils dans l'atelier.

Nous allons maintenant résumer l'état actuel de nos analyses:

Le tableau synthétique (Figure 5.1) représente l'ensemble des caractéristiques d'un atelier de spécifications.

Chaque case est la contribution d'une composante à une classe de besoins.

Les cases grisées à l'intersection d'un besoin et d'une composante, montrent, par leur intensité, l'importance de la composante pour résoudre ce besoin.

Lorsqu'une composante, bien que nécessaire pour satisfaire un besoin, complique la prise en compte du besoin, l'intersection est remplie de "----". Par exemple, la répartition de l'environnement rend plus difficile la gestion globale des données ainsi que la coopération entre utilisateurs. Ce problème est désormais classique dans les systèmes distribués.

Objectifs n° / Classe de BESOINS / COMPOSANTES	**2,3,4,15** Formalismes de haut niveau	**10,19** Edition des données et représentation des résultats	**4,5,6,8** Gestion globale des données Coopération entre utilisateurs	**12,13** Multi-traite-ment Puissance de Calcul	**9,11** Ergonomie et facilité d'utilisation	**1,7** Qualité Génération de prototypes	**14,16,17,18 20,21,22,23** Extensibilité Réutilisation Communication Intégration
Structure d'accueil		majeure	majeure	majeure		indispensable	indispensable
Interface Utilisateur	majeure	indispensable	mineure		complication		
Multi-utilisateur			indispensable	mineure			
Gestion des formalismes	indispensable	mineure	majeure				majeure
Répartition		indispensable	indispensable	indispensable	majeure	majeure	mineure
Station de travail virtuelle		majeure	majeure		mineure		indispensable
Outils d'administration	majeure			majeure	mineure	indispensable	majeure
Méthodes de conception					majeure	indispensable	majeure
Système de gestion de fichiers virtuels		indispensable	indispensable		majeure	mineure	indispensable

indispensable majeure mineure complication

Figure 5.1 : Adéquation des besoins aux composantes

En prenant les composantes ligne par ligne, nous pouvons analyser si la composante est importante. Si nous classons les besoins par ordre d'importance, nous pouvons mesurer si la composante est nécessaire.

Par exemple:

- La composante multi-utilisateurs résout le besoin de coopération et celui de partage de ressources. Or si cette coopération n'est pas primordiale, la composante n'a pas à être spécifiée.
- Comme nous l'avons déjà souligné, la répartition est importante en cas de calculs importants mais pose des problèmes pour la coopération entre utilisateurs et pour la recherche d'informations.

Le tableau 5.1 représente toutes les composantes que nous avons retenues pour un atelier de spécification. La plupart des composantes répondent bien aux besoins. Nous aboutissons ainsi un ensemble de recommandations pour l'élaboration d'un cahier des charges. Ce tableau peut être utilisé comme un guide pour l'évaluation d'autres ateliers de spécification de systèmes. Ces recommandations sont des hypothèses ou de choix stratégiques que nous nous sommes fixés pour obtenir un atelier de spécification respectant notre analyse des besoins des utilisateurs (§2 Analyse des besoins des utilisateurs) et des besoins en systèmes (§3 Analyse des besoins en systèmes).

Nous allons maintenant détailler les différentes composantes pour obtenir une description plus fine et monter comment elles peuvent résoudre les besoins des utilisateurs.

5.2. Structure d'accueil

5.2.1. Introduction

Un atelier de spécification de systèmes étant par nature évolutif, il doit permettre d'accueillir de nouveaux outils. Pour cela, il doit disposer d'une structure d'accueil sur laquelle vont s'ajuster les différents outils. Ces outils vont des programmes d'analyse à des paquets de règles de systèmes experts, et à de nouveau type d'interface utilisateur.

5.2.2. Multi-formalismes de haut niveau

L'ensemble des formalismes gérés par l'atelier sont centralisés au niveau de la structure d'accueil. Ceci permet d'obtenir une gestion globale pour faire communiquer entre eux différents outils et pour passer d'un formalisme vers un autre formalisme par des transformations de représentation.

Cette gestion globale des formalismes est utile pour ajouter, au niveau de la structure d'accueil des outils intermédiaires de test de cohérence et de validité de modèles. Ces tests étant dans le passé effectués dans chacun des outils travaillant sur un formalisme particulier, nous nous sommes aperçus qu'ils devaient être regroupés pour assurer leur complétude. En les regroupant, on uniformise et on précise le formalisme d'entrée des modèles. Par exemple, rien ne fixe le format d'un objet, c'est une donnée interne pour le formalisme. Or si la taille ou la struture de l'objet n'est pas fixée de manière standard, le résultat d'une application n'est pas traduisible automatiquement pour être utilisé par d'autres applications que l'on voudrait compatibles.

5.2.3. Communication entre outils

Lorsque l'on divise le traitement à effectuer pour analyser, on s'aperçoit que l'on a besoin d'algorithmes élémentaires qui existent peut-être dans d'autres applications. Si ces algorithmes sont suffisamment ciblés, il est possible de les réutiliser. Il faut alors communiquer avec ces algorithmes et la structure d'accueil offre un moyen uniforme de communication. Cette communication sera transparente car l'outil demandeur est vu par l'outil demandé comme une interface utilisateur.

5.2.4. Intégration d'applications

La structure d'accueil est le point central pour l'intégration d'applications. L'intégration se fait en ajoutant les applications les unes à coté ou au dessus des autres. La structure d'accueil doit permettre d'intégrer des applications compilées dans différents langages de programmation comme des applications interprétées. D'autre part, l'utilisation d'un système expert intégré dans l'atelier de spécification, est une voie d'écriture d'applications sous forme de paquets de règles.

Dans tous les cas, les applications intégrées n'ont pas à se soucier des problèmes de graphique et la structure d'accueil offre un ensemble de moyens pour transmettre des résultats à l'utilisateur.

5.2.5. Qualité: Génération de prototypes

Avec l'utilisation d'une structure d'accueil ouverte, la génération de prototype accède à une autre échelle. En effet, un prototype généré par l'atelier de spécification, peut s"auto" intégrer et devenir un outil à part entière de l'atelier. C'est en particulier le cas lors de l'exécution de prototypes dont les résultats se traduisent en animations graphiques.

5.2.6. Coopération entre utilisateurs

Pour la coopération entre utilisateurs, la structure d'accueil permet d'effectuer une gestion globale des données des utilisateurs, ce qui facilite la communications de ces données.

Accessoirement, cette centralisation permet de gérer le multi-linguismes par des traducteurs intégrés. Les utilisateurs peuvent donc utiliser l'atelier de spécification dans leur langue.

5.2.7. Puissance de calcul

L'utilisation d'une structure d'accueil permet de gérer la tolérance aux pannes lors de calculs très longs. En effet cette structure est vue comme une machine virtuelle pour l'utilisateur. La localisation de l'exécution ne lui incombe pas et la structure peut donc s'auto-configurer pour offrir un maximum de disponibilité à l'utilisateur. En cas de panne d'un matériel, un autre matériel équivalent peut être employé.

Lorsque l'utilisateur requiert des calculs importants, la structure peut, de manière transparente, répartir au mieux la charge en fonction des puissances de machines disponibles et même procéder à une **parallèlisation des calculs** [Eager, 1986, Foliot, 1989] de l'utilisateur en utilisant au mieux le parc de machines disponibles.

5.3. Interfaces Utilisateur

5.3.1. Introduction

L'apparition d'écrans "bitmap" chez tous les constructeurs d'ordinateurs permet à un utilisateur de travailler simultanément avec un grand nombre d'applications sur un même écran. Cette évolution technologique [Salmon, 1987] est entrée dans les mœurs des utilisateurs et il n'est plus possible de proposer de logiciels n'ayant pas d'interface graphique conviviale. Or cette interface, bien que nécessaire, n'est pas une chose facile à réaliser. Il faut pourtant la fournir en standard dans tout environnement destiné à supporter de nombreuses et nouvelles applications.

Les travaux de recherche sur la construction d'interfaces homme-machine [Beaudoing-Lafon, 1988, Coutaz, 1985, Shneiderman, 1987] ont porté sur des sujets comme l'introduction graphique de données, les transformations et l'édition graphique des résultats ou la normalisation des interfaces utilisateur. Elles ont mis en évidence des principes parmi lesquels la nécessité d'une séparation entre interface graphique (menus etc…) et les applications graphiques. Nous avons retenu ce raisonnement pour proposer une séparation physique entre les applications de calcul et l'application graphique.

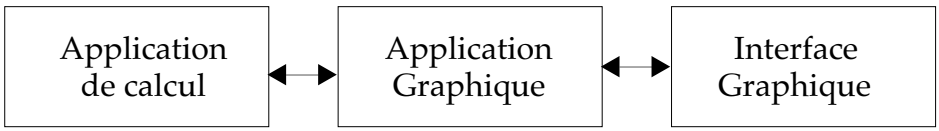


Figure 5.3: Séparation interface graphique - application de calcul

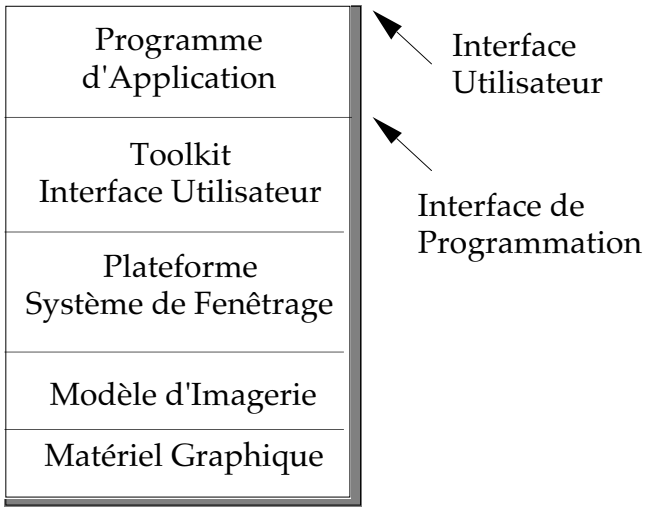
La plupart des logiciels pilotent l'utilisateur [Scapin, 1986]: le choix des questions possibles à un moment donné est assez réduit et connu lors de la conception du logiciel. Cette approche convient très bien aux personnes peu expérimentées ou non familières du domaine (novices). D'autres peuvent être vraiment pilotés par l'utilisateur comme les systèmes experts ou les logiciels conçus avec une approche objet. Cette approche est mieux adaptée aux utilisateurs experts.

La sécurisation d'une interface utilisateur évite les erreurs en ne permettant que ce qui est licite, en fournissant des mises en garde, en ayant des diagnostics et des aides, en garantissant des sauvegardes et des retours en arrière.

La réalisation d'interfaces est une tâche fastidieuse qui se pose à chaque nouvelle application: saisie des données, fonctionnalités (menus, diagnostics, aides), récupération des résultats. La complexité de la réalisation provient surtout du grand nombre d'événements intervenant dans n'importe quel ordre. Il est possible d'utiliser des générateurs d'interface utilisateur qui permettent une modélisation du dialogue. Cette méthode permet de réaliser des interfaces paramètrées déclaratives que le développeur particularise pour chaque application par des descriptifs (utilisation d'un automate). Il faut aussi penser lors de la conception d'une interface utilisateur à la portabilité :

- sur une autre machine de type différent
- sur une machine ayant une autre configuration (écran, petit ou A3).

L'utilisateur voit l'interface utilisateur (UI), les barres de défilement, les menus et les boutons qui contrôlent le programme. Le programmeur voit une interface de programmation, les noms des fonctions, les séquences d'appel et les structure de données (API) qui doivent être intégrés dans les programmes pour leur donner une interface utilisateur. Le "Toolkit" se situe entre ces deux interfaces. Il construit l'interface utilisateur par l'interface de programmation (API). L'API est donc définie par le toolkit. Il en résulte que l'API et UI sont séparées et distinctes. Il est possible que plusieurs API soient utilisées par une interface utilisateur unique.



Actuellement, la consistance de l'interface utilisateur ne peut pas être entièrement définie par une API, il est nécessaire de respecter un guide de style [APPLE, 1987, OPEN LOOK, 1989a, OSF/Motif, 1989a] écrit en langage naturel qui permettent d'obtenir un "Look and Feel". Il est donc très difficile de contrôler si une interface utilisateur respecte un certain "Look and Feel". Le guide de style est non seulement ce qui dicte le comportement des écrans (le Feel) mais aussi ce qui en donne l'aspect général (le Look). L'interface utilisateur est donc définie par les spécifications (API) et le guide de style.

5.3.2. Interface utilisateur privilégiée

Pour obtenir une interface utilisateur interactive agréable, il faut rendre la machine disponible le plus rapidement possible pour l'utilisateur et lui permettre de voir immédiatement les effets de ses directives. Tout doit être mis en œuvre pour améliorer la vitesse d'exécution des commandes ou des actions les plus utilisées.

Nous avons pris en compte dans l'architecture cette gestion privilégiée du travail de l'utilisateur par rapport à celui des programmes d'applications. Cette caractéristique peut être facilement mise en œuvre par l'utilisation d'un poste de travail dédié à l'interface utilisateur, permettant aussi à l'utilisateur de travailler (édition) en **mode autonome** sans connexion à l'atelier. Ce poste de travail peut être orienté graphique par opposition aux machines de traitement qui doivent être orientées puissance de calcul. Une des principales caractéristiques d'une telle architecture est alors le support d'environnement hétérogène d'exécution et l'utilisation transparente d'un réseau de machines.

Nous ne conseillons pas de faire exécuter des applications coûteuses (qui demandent des ressources systèmes cpu, mémoire) sur la station supportant l'interface utilisateur. Le cahier des charges ainsi que la réalisation AMI vont prendre en compte cette recommandation. Cette autonomie du poste de travail est une des originalités de notre travail.

5.3.3. Entrée des données, représentation des résultats

Le mode de dialogue mis en place doit favoriser la désignation directe à l'écran des commandes à activer; la manipulation directe offre l'avantage de "reconnaître et pointer" à l'inverse du mode qui consiste à "se souvenir et taper".

Le processus de **visualisation des données** entre en très grande partie dans l'utilisation du graphisme pour que l'homme puisse mieux les visualiser et les analyser: Une image parle mieux que des chiffres. Par exemple, dans les simulations pour l'analyse de systèmes, l'évolution de paramètres dans le temps est plus facile à appréhender sous la forme d'un graphique en deux dimensions qu'en tableaux de chiffres. Certaines erreurs sont plus facilement détectables. L'importance du caractère esthétique de la présentation est primordiale comme les préférences des utilisateurs.

Cependant il existe des limites aux représentations graphiques: un graphe est plus lisible qu'un ensemble de relations mais si il y en a trop, le graphe devient illisible. Il faut donc ajouter d'autres mécanismes de filtrage, de condensation ou de vision partielle. Une autre solution peut être la visualisation multiple sous différents points de vue, à différents niveaux de détail.

La représentation graphique est plus lisible pour certains modèles de petite taille ou pour certains résultats. Par contre, pour l'entrée de modèles plus importants, une **représentation textuelle** ou une **représentation graphique hiérarchique** doit être envisagée.

Les formalismes reposant sur une représentation graphique du système, il est nécessaire d'**homogénéiser** leur utilisation c'est-à-dire de permettre au concepteur de les utiliser à partir d'un même poste de travail, selon une approche uniformisée.

5.3.4. Ergonomie d'utilisation

L'utilisation d'une interface unique pour l'accès à tous les outils d'un atelier de spécification augmente son ergonomie: interface homogène, compatible avec l'environnement de travail. De plus, si on porte ses efforts sur l'évolution de cette interface vers une plus grande facilité d'utilisation, toutes les applications bénéficient des améliorations. L'utilisateur final a l'impression de n'utiliser qu'un seul logiciel très puissant contenant toutes les fonctionnalités qui lui sont nécessaires.

5.3.5. Puissance de calcul

Les algorithmes de traitement des spécifications de systèmes demandent souvent de longs temps de calculs. Ces temps de calcul ne doivent pas pénaliser l'interface utilisateur qui doit être privilégiée. Pendant un calcul, le modélisateur doit pouvoir faire autre chose sur sa station de travail, tout en étant informé du déroulement du calcul.

Plusieurs niveaux d'interaction sont nécessaires. La visualisation d'une information de terminaison du calcul ou de progression est utile si l'utilisateur attend le résultat pour continuer son travail. Lorsque le résultat risque de demander trop de temps, il faut pouvoir se "déconnecter" complètement pour revenir plus tard et être alors informé de son évolution.

5.3.6. Intégration d'applications

L'atelier de spécification est une structure d'accueil pour l'intégration de nouvelles applications. Afin de faciliter cette intégration, l'atelier offre une interface graphique commune aux applications évitant l'écriture d'une interface pour chacune d'elles et garantissant l'homogénéité globale. Cette interface est une interface générique et paramètrable par l'environnement ce qui rend l'atelier extensible. En effet, en cas de changement sur le cahier des charges de l'atelier, il est possible d'ajouter des paramètres rendant l'interface compatible avec les anciens modèles tout en évoluant vers de nouveaux modèles.

5.3.7. Multi-formalismes de haut niveau

Les formalismes de haut niveau doivent être simples à employer et ils reposent très souvent sur des **représentations graphiques**. Il est parfois difficile de faire des analyses et traitements directement sur les formalismes de haut niveau aussi les traduit on automatiquement dans des formalismes de plus bas niveau sur lesquels s'appliquent des algorithmes de preuve. Pour faire remonter ces preuves jusqu'au formalisme amont, l'interface utilisateur doit elle aussi gérer le **multi-formalismes** et en particulier exploiter plusieurs formalismes simultanément et les lier interactivement afin d'aider le modélisateur à interpréter ses résultats.

5.3.8. Qualité: Génération de prototypes

Un atelier de spécification a pour but de générer des prototypes des systèmes réalisés. Rien de tel que de "voir" l'exécution du prototype pour apprécier son fonctionnement. Pour voir l'exécution, deux solutions sont envisageables: la simulation et l'exécution réelle. Dans les deux cas le résultat doit être graphique et l'interface doit donc être capable de visualiser des **animations** des systèmes modélisés.

5.3.9. Coopération entre utilisateurs

La coopération entre utilisateurs, au niveau de l'interface graphique consiste à se transmettre des éléments de modèles (graphique). Elle peut être statique ou dynamique. Dans une **coopération statique**, un utilisateur va copier une partie de modèle pré-existant et l'inclure dans son nouveau modèle (Couper/Coller). Dans une **coopération dynamique**, plusieurs utilisateurs construisent simultanément un modèle ou utilisent des parties d'autres modèles; ces parties pouvant évoluer et donc modifier les modèles les utilisant (liens dynamiques).

Au niveau purement graphique, les différents types de liens peuvent être effectués sans collaboration avec la structure d'accueil, le résultats étant en effet un nouveau modèle complet.

5.4. Multi-utilisateurs

Une gestion utilisateur, dans un atelier de spécification consiste, en particulier, à identifier les utilisateurs et à gérer les données des utilisateurs, en entrée comme en résultats.

5.4.1. Coopération entre utilisateurs

Il est indispensable d'avoir une gestion multi-utilisateurs pour permettre une coopération efficace entre utilisateurs. Si la structure d'accueil ne connaissait pas la notions d'utilisateurs, il ne pourrait pas y avoir de collaboration automatique; c'est à dire que la seule collaboration aurait lieu par l'échange de documents et de fichiers au niveau du système d'exploitation sous-jacent..

Cette coopération pose aussi des problèmes des droits d'accès des utilisateurs. Lors de l'importation de modèles, il peut y avoir importation en lecture seule ou en lecture + écriture.

5.4.2. Sécurité

Plusieurs utilisateurs peuvent travailler avec l'atelier, il est donc nécessaire de les identifier et de sécuriser l'accès à l'atelier. Nous avons montrer dans l'analyse des besoins, qu'il y avait plusieurs catégories d'utilisateurs : les modélisateurs et théoriciens, les concepteurs de logiciels et les administrateurs. Chaque catégorie d'utilisateur a une vue et des droits différents dans l'atelier.

5.4.3. Puissance de calcul

Le fait d'avoir à effectuer des calculs très importants dans un environnement multi-utilisateurs pose un problème de puissance de calcul. En effet, un utilisateur pourra gêner les autres. Il faut donc le prévoir et soit le limiter, soit répartir les calculs si on dispose d'un parc de machines en réseaux.

5.5. Gestion des formalismes

5.5.1. Introduction

Dans la plupart des logiciels, les types de modèles utilisables sont définis implicitement par la collection des programmes de construction et d'analyse disponibles. Une extension de modèle coûte alors des modifications inextricables de structures impliquant une réécriture de nombreux programmes.

Nous préconisons donc qu'un atelier de spécifications ait une **gestion explicite des formalismes** utilisés.

5.5.2. Multi-formalismes de haut niveau

L'expression du formalisme est alors faite sous forme d'un **méta-modèle** défini pour représenter les formalismes. Cette notion de méta-modèle est aussi employée pour des environnements de Conception Assitée par Ordinateur [Tomiyama, 1989].La plupart des méthodes de modélisation sont fortement reliées au concept de graphe. Le concept de graphe est un outil universel pour représenter les formalismes. L'atelier étant basé sur les graphes structurés typés (**modèle de base**), il n'est couplé à aucun formalisme particulier (neutralité).

Les formalismes étant décrits de manière explicite dans l'atelier, il est plus facile d'introduire de nouveaux formalismes ou de nouvelles catégories de formalismes.

5.5.3. Communication entre outils

Il faut pouvoir faire communiquer et coopérer des outils sur des formalismes différents.

La communication entre outils est facilitée car la gestion des formalismes impose une description précise des instances des objets du formalisme. Cette description étant normalisée, cela garantit une facilité de description des modèles d'entrée.

5.5.4. Entrée des données, représentation des résultats

La gestion des modèles théoriques consiste, en particulier, en une description graphique (esthétique), syntaxique et sémantique des modèles. Suivant le niveau de finesse de cette description, il sera possible de vérifier une partie de l'énoncé dès la saisie. Certaines vérifications syntaxiques, comme les types de nœuds reliés par des arcs, seront facilement vérifiées quand le modèle d'entrée est graphique. Lorsque l'énoncé est textuel, il est plus difficile d'effectuer une vérification syntaxique à la saisie; cela risque d'entraîner une trop grande rigidité. On tombe ici dans le problème des éditeurs syntaxiques, satisfaisants pour l'esprit mais souvent détournés de leur utilisation dès qu'ils possèdent un mode "non strict". Ces éditeurs sont encore plus difficiles à réaliser avec l'utilisation de la souris et du couper/coller qui entraîne l'existence temporaire de modèles incomplets ou incorrects.

La description esthétique des formalismes permet aussi de spécialiser l'interface utilisateur: les communications entre l'interface et les programmes d'application n'auront pas à connaître l'esthétique, cela diminuera les échanges d'information et permettra de particulariser complètement l'interface pour un utilisateur.

5.5.5. Coopération entre utilisateurs

Le choix d'un modèle basé sur les graphes permet de faciliter la communication entre utilisateurs car il est possible de définir dans un graphe des boites noires (sous-graphes) qui sont réalisées par d'autres utilisateurs et définies par une interface (ensemble de nœuds d'entrée et de sortie).

La gestion des formalismes apporte de la rigueur à cette coopération en imposant des règles de construction de graphe: par exemple, en contrôlant que l'importation d'une boite noire appartient au bon formalisme ou à un formalisme compatible (en cas de hiérarchie de formalisme au sein d'une catégorie donnée).

5.6. Répartition

5.6.1. Introduction

Nous envisageons la **répartition à deux niveaux**: D'abord, la répartition entre l'interface utilisateur et les programmes de calcul, et ensuite, la répartition de charge des applications de calculs sur un réseau d'ordinateurs.

5.6.2. Puissance de calcul

La répartition des applications de calcul est indispensable, lorsque certains calculs deviennent très importants ou que certaines applications ne peuvent s'exécuter que sur des machines dédiées (utilisation d'un environnement logiciel particulier). La gestion de la répartition au niveau de la structure d'accueil d'applications permet ainsi d'effectuer des traitements sur des machines hétérogènes.

La séparation physique de l'interface utilisateur (**poste de travail dédié**) est un principe que nous avons introduit afin qu'un calcul ne pénalise pas l'interaction utilisateur. Cette solution s'impose actuellement, puisque les outils graphiques intégrés dans certains ordinateurs (Macintosh) sont plus ergonomiques et plus efficace que ceux disponibles sur des stations de travail (Sun). Par contre l'inconvénient d'une telle solution réside dans la quantité d'informations échangées. Elle n'est efficace que si il y a minimisation de cette quantité d'information.

5.6.3. Communication entre outils

La gestion de la répartition sur des machines hétérogènes ouvre bien plus les possibilités de communications entre outils. Mais la structure d'accueil devrait plus évoluer pour assurer cette communication avec éventuellement des changements de format des données et des gestions transparentes de liaisons inter-machines.

Par contre l'hétérogénéité rend plus difficile la gestion globale de résultats, il faut un système sophistiqué pour permettre le rapatriement de fichiers résultats. Un solution consiste en une centralisation du système de gestion de fichiers.

5.6.4. Qualité: Qualité du logiciel

Dans un atelier de spécification, il faut rendre le problème de répartition transparent aux utilisateurs. L'utilisateur attend un résultat, si possible le plus rapidement, mais il ne veut pas ni ne doit pas s'intéresser à la localisation de son calcul. De plus les défaillances du réseau et ses reconfigurations doivent être transparentes. Si une gestion de copies multiples ou de sauvegardes est intégrée dans le réseau alors les utilisateurs gagnent en sécurité.

5.6.5. Moyen de communication

Les communications sont une nécessité pour la répartition. Plutôt que des communications point à point par liaisons spécialisées, les solutions modernes sont basées sur des réseaux et même sur des ensembles de réseaux. Dans une telle optique toutes les couches de transport sont prises en charge par le gestionnaire standard de réseaux. L'usager y gagne alors un enrichissement des possibilités de connexion à distance et d'échange d'informations avec d'autres ateliers munis d'autres outils.

5.7. Station de travail virtuelle

5.7.1. Introduction

Nous avons introduit la notion de Station de travail Virtuelle (SV) pour affranchir les applications de la localisation de l'interface utilisateur. Ainsi, les applications n'ont même rien à connaître de la gestion d'écrans et du graphique puisqu'il se peut même que le travail leur ait été demandé par une autre application. L'atelier de spécification rend les applications indépendantes du système de fenêtrage du poste de travail. Tout le graphique étant reporté sur l'interface utilisateur, il y a minimisation des données échangées entre cette interface et les applications de calcul. La portabilité des applications est ainsi améliorée et leur interface est simplifiée.

De plus la répartition de l'atelier offre une sorte de super système de fenêtrage rendant transparent l'accès à l'ensemble des sites d'exécution (ce qui n'est pas le cas dans les système de fenêtrages distants comme X-Window). Il y a non seulement transparence pour les applications mais aussi transparence pour les utilisateurs. En effet le simple fait de changer de fenêtre sur l'interface, effectue une **commutation de session** et éventuellement de machine.

5.7.2. Puissance de calcul

La possibilité de commutation de session augmente grandement l'interactivité de l'interface en permettant en cas de calculs très importants de passer à la conception ou à l'analyse d'un autre modèle.

5.7.3. Qualité: Génération de prototypes

Lors de l'exécution de prototypes, la multi-sessions permettra à l'utilisateur de voir simultanément l'exécution de son modèle avec son animation ce qui laisse la possibilité d'agir sur le modèle pour changer ou diriger l'animation.

5.8. Outils d'administration

La conséquence directe d'avoir une gestion des formalismes est la nécessité est de les administrer. Les outils d'administration sont aussi indispensables lorsque l'on travaille dans un environnement multi-utilisateurs.

5.8.1. Multi-formalismes de haut niveau

L'administration des formalismes est un problème crucial dans un atelier de spécification. En effet, il faut décider s'il faut ou non créer un nouveau formalisme ou si le nouveau formalisme n'est pas une généralisation d'un formalisme existant. En effet, créer un nouveau formalisme trop proche d'un ancien formalisme, ne permettra pas de réutiliser d'anciennes applications pourtant peut-être adéquates. Il faut alors faire appel à des outils d'administration des formalismes.

La gestion de formalismes va aussi faire apparaître la nécessité de gérer les applications.

5.8.2. Intégration d'applications

Gérer les applications nécessite donc une description externe de leurs fonctionnalités, de leurs limites dans les formalismes d'entrée et les formalismes de leurs résultats. Cette gestion globale des applications simplifie la réutilisation d'applications.

Une administration des formalismes facilite l'ajout d'applications dans la mesure où une description précise est faite. Cette intégration devient plus sûre si on génère automatiquement des jeux d'essai pour vérifier le bon comportement de l'application.

Les fonctionnalités des applications doivent apparaître sur l'interface utilisateur. Des outils d'administration sont là aussi nécessaires pour concevoir l'enchaînement des différentes fonctionnalités.

5.8.3. Coopération entre utilisateurs

Pour faciliter la coopération entre utilisateurs, des outils de type messagerie peuvent être mis en place. Nous pensons à une gestion automatique de message informant les concepteurs d'applications lors de problèmes systèmes survenus lors de l'utilisation de leurs outils.

5.8.4. Répartition

Les calculs importants conduisent à la distribution et cette répartition ne peut pas être faite sans outils: interrogation sur des données distribuées, sauvegarde de versions de données avec leurs résultats…

5.8.5. Ergonomie d'utilisation

Les outils d'administration doivent eux-aussi avoir une interface conviviale, il faut veiller à ce qu'ils soient conforme à l'interface utilisateur du poste de travail des administrateurs.

5.9. Machine Abstraite

5.9.1. Introduction

La machine abstraite permet l'échange d'information entre les différentes entités de l'atelier par un représentation intermédiaire commune. Cette représentation intermédiaire est un langage exprimant les objets manipulés, ainsi que les opérations possibles sur ces objets, les méthodes d'accès à ces objets et les différentes manipulations possibles. Nous avons défini un langage pour décrire des modèles et des données dans des formalismes.

5.9.2. Communication entre outils

Le langage de communication sert d'abord à faire communiquer des applications éventuellement réparties. Il est indépendant des machines utilisées se qui permet de s'affranchir d'une architecture ou d'un type de machine donné.

Pour favoriser l'interopérabilité, une application peut générer un résultat dans ce langage défini; ainsi toute autre application saura utiliser ce résultat en données d'entrée. Dans le cas où le langage de communication ne permet pas la représentation d'un résultat, l'interopérabilité est mise en défaut. Dans ce cas une application ne pourra accéder à ce résultat que sous une forme textuelle non standardisée.

5.9.3. Qualité: Qualité du logiciel

L'utilisation d'un langage de communication permet de maintenir l'intégrité des applications. Les risques de mauvais fonctionnement sont diminués car ce langage est interprété par chaque application pour être transformé en structures de données internes. Ainsi, les applications ne travaillent pas sur des structures de données communes et ne risquent donc pas d'interférer entre elles.

5.9.4. Intégration d'applications

L'utilisation d'un langage de communication facilite l'intégration d'applications puisqu'il définit une normalisation. Cela facilite aussi les tests d'applications avant intégration et les comparaisons d'applications puisque l'on peut définir des jeux de test simulant l'interface utilisateur et demandant d'effectuer des calculs. Si les résultats sont normalisés, la comparaison peut être automatique ce qui facilite grandement la vérification de "correction" après modifications.

5.10. Méthodes de conception

Les méthodes de conception et de programmation sont nécessaires dès que l'on travaille sur de grands projets comme un atelier de spécification. Le plus dur est de forcer les concepteurs d'applications à avoir une bonne méthode.

5.10.1. Communication entre outils

La communication entre outils est facilitée par une normalisation des interfaces et l'usage d'un langage pivot. Des logiciels sont plus aisément réutilisables lorsque cette communication est normalisée.

5.10.2. Intégration d'applications

L'utilisation d'un atelier de spécification permet d'améliorer l'intégration d'applications en particulier en réutilisation des outils déjà existants. La qualité de programmation améliore la maintenance des logiciels. En séparant les messages du corps des applications, on obtient des applications plus faciles à traduire en différentes langues (multi-linguismes).

5.10.3. Puissance de calcul

Lors de calculs importants, si une bonne structuration a été utilisée, le parallélisme sera plus facile à effectuer et la distribution entre différents sites sera alors possible. Il est important d'avoir décomposé son logiciel.

5.11. Système de gestion de fichiers virtuels

5.11.1. Introduction

Le système de gestion de fichiers virtuels peut être vu comme une base de données des énoncés de problèmes des utilisateurs ainsi que des résultats. C'est un système de gestion de fichiers posé au dessus du gestionnaire de fichiers du système d'exploitation. Nous l'appelons gestion de fichiers virtuels par opposition au système de gestion de fichiers classiques (réels).

5.11.2. Puissance de calcul

Lors de calculs importants, les risques de pannes augmentent et si on veut garantir un niveau supérieur de tolérance aux pannes, il faut pouvoir recommencer les calculs en cas de panne. Pour cela il faut que les données de l'utilisateur soit de nouveau accessibles et ceci est possible si elles sont gérées dans la base de donnée qu'est le système de gestion de fichiers virtuels.

5.11.3. Intégration d'applications

Lors de l'intégration d'applications, le système de gestion des fichiers virtuels va faciliter et rendre uniforme les accès aux fichiers. Il permettra aussi d'assurer une gestion globale des données des applications et de traiter les problèmes d'accès concurrents.

5.11.4. Coopération entre utilisateurs

La coopération entre utilisateurs ne peut se faire que si l'atelier est capable de retrouver des données d'un utilisateur, ou les différentes versions de ses données. Il doit y avoir gestion des utilisateurs mais aussi gestion des fichiers des utilisateurs.

La gestion des utilisateur dans un environnement multi-utilisateurs permet de leur associer des niveaux d'utilisation et des droits d'accès, ce qui permet de créer différentes vues de l'atelier.

5.11.5. Qualité: Génération de prototypes

La génération d'un prototype ne comporte pas seulement le code du prototype, mais aussi des données externes. Le système de gestion de fichiers virtuels permet également de gérer les données de ces prototypes. C'est en particulier utile lorsque le prototype va s'exécuter sur une machine non fixée au départ, le prototype généré n'aura pas à se préoccuper de l'emplacement de ses fichiers.