L'environnement générique pour spécifications formelles hétérogènes MSPEC

version provisoire

Patrick Amar[1,2] & Gilles Bernot[1]
pa@lri.fr, bernot@lami.univ-evry.fr

[1] Université d'Évry
LaMI, CNRS EP 738
Cours Mgr Roméro
91025 Évry CEDEX

[2] Université Paris-Sud
L.R.I. CNRS UMR8623
Bât. 490
91405 Orsay CEDEX

Résumé : l'environnement de spécification MSPEC a pour but d'outiller la définition de nouveaux langages de spécification formelle, d'assister l'écriture de spécifications structurées et leur raffinement, et enfin d'autoriser la cohabitation hétérogène de plusieurs langages au sein d'une même spécification. Cet article expose le résultat de nombreuses réflexions sur l'analyse des besoins pour un tel environnement et exploite le retour d'expérience de deux prototypes préalables. Il peut être également lu comme une version préliminaire du cahier des charges de MSPEC. On porte une attention toute particulière à la simplicité de mise en œuvre et d'utilisation de MSPEC. En l'état actuel, l'interface de MSPEC repose sur un éditeur de formules mathématiques particulièrement convivial développé dans ce but : MWX. Compte tenu de son rôle central, une présentation synthétique de MWX est également donnée.

1 Présentation du projet MSPEC

1.1 Atelier d'expérimentation

Dans le domaine des méthodes formelles pour le génie logiciel, on distingue classiquement deux familles de langages de spécification formelle : les langages de spécification "orientés modèle" comme VDM [DFNH93] [Daw91], Z [Spi89] ou B [Abr96] et les langages de spécification "axiomatiques" dont la classe la plus connue est celle des langages de spécification dits "algébriques" [GTW78] [EM85] [Wir90] [COMPASS91] [Ber94]. Ces méthodes de spécification sont maintenant largement sorties du milieu académique et sont couramment utilisées en industrie pour les composantes critiques des logiciel ou des matériels.

L'utilisation croissante des méthodes formelles dans l'industrie du logiciel s'accompagne de recherches actives dans le domaine de la conception de nouveau langages de spécification formelle. Contrairement aux spécifications orientées modèle [Abr96], les spécifications axiomatiques ne cherchent pas à étendre un langage de spécification donné vers des langages de plus en plus généraux. En fait, à l'heure actuelle, ils ne seraient pas outillables (preuve de théorèmes, tests, prototypage, etc.). Il s'agit ici plutôt d'offrir un éventail de langages *dédiés*, qui semblent faciles ou naturels à lire pour des spécialistes d'un domaine précis [AB96] [ABB+96a] [BD94].

Cette richesse de langages est fortement appréciée des chercheurs en milieu académique qui peuvent ainsi explorer les meilleurs compromis entre expressivité et outillage, en terme de preuves par exemple, de complétude ou d'extraction de jeux de tests [BGM91a] [AL97]. Malheureusement, en l'état actuel des connaissances et surtout de l'outillage, il n'existe au mieux que quelques environnements utilisables liés à un nombre relativement restreint de langages de spécification. La diversité des langages axiomatiques est de ce fait mal ressentie par les milieux industriels en raison de la dispersion des forces qu'elle entraîne.

En pratique, chaque industrie a un langage de prédilection qu'elle utilise systématiquement parce qu'elle possède les compétences nécessaires pour manipuler les outils correspondants. Cependant chaque langage de spécification est adapté à un éventail d'applications plus ou moins large et cela a pour conséquence que certains composants logiciels sont spécifiés selon un langage peu commode pour sa description. On gagnerait en clarté d'écriture des spécifications s'il était possible de choisir un langage de spécification *ad hoc* pour chaque composant. C'est en fait cela la motivation principale à l'origine du projet MSPEC.

Concilier unicité de l'environnement et multiplicité des langages formels est le but de MSPEC (pour "méta-spécification"). MSPEC se veut un environnement d'aide à trois activités :

- la conception de nouveaux langages de spécification formelle,
- l'écriture de composants de spécification dans un langage donné,
- l'intégration de plusieurs langages de spécifications au sein d'une même spécification hétérogène.

L'hétérogénéité des spécifications se justifie pleinement par la *réutilisation*, incoutournable en génie logiciel. En effet, qui dit réutiliser dit "ne pas changer une virgule", ni dans le code du composant réutilisé, ni dans sa spécification. De ce fait, l'hétérogénéité des spécifications est une fatalité, par réutilisation de composants dont on ne domine pas le choix du langage de spécification. Un autre cas d'hétérogénéité est la multiplicité des vues sur un même composant : chaque utilisateur peut en utiliser des aspects distincts (comportement fonctionnel, comportement temporel, aspects concurrents, etc.) qui peuvent avantageusement s'exprimer dans plusieurs langages dédiés plutôt qu'avec des "contorsions" dues à l'usage d'un langage inapproprié à l'expression d'un certain type de propriétés. De la même façon, si certains langages simples (par exemple sans traitement d'exceptions) sont bien adaptés à un haut niveau d'abstraction, il deviennent trop peu puissants pour un raffinement vers les spécifications détaillées (par exemple avec traitement des cas exceptionnels). Il est donc utile de considérer des raffinements hétérogènes, d'un langage abstrait vers un langage plus concret.

Toutes ces considérations plaident pour un environnement comme MSPEC, non pas unificateur mais seulement lieu de connexions entre plusieurs langages axiomatiques différents. Pour atteindre ce but il faut avant tout manipuler des *langages* de spécification (en particulier des formules dans plusieurs logiques) et en ce sens, MSPEC est aux langages de spécification formelle un peu ce que CENTAUR [JR92] [KLM83] est aux langages de programmation (voir section 4.1). De même, cela n'exclut pas de pouvoir "verrouiller" MSPEC sur un langage de spécification donné et fournir ainsi clef en main un environnement dédié.

Naturellement MSPEC restera encore longtemps seulement un *atelier d'expérimentation* : il aura déjà correctement atteint son but s'il peut être utilisé dans le cas typique d'un jeune doctorant qui voudrait essayer rapidement une nouvelle logique, qu'il conçoit pour un domaine dédié, et où MSPEC lui permettrait de s'attaquer à une étude de cas de bonne taille plutôt qu'à une simple structure de donnée ...

1.2 Utilisation simple

A la différence des environnements de spécification formelle dédiés à un langage donné [BC85] [GH86] [BDE87] [SELL97], on ne cherche pas pour MSPEC à avoir un nombre très grand de fonctionnalités (souvent propres à ce langage particulier), mais au contraire peu de primitives, avec une grande facilité d'utilisation et une ergonomie uniforme pour avoir une utilisation intuitive.

Pour qu'un utilisateur "naïf" puisse obtenir rapidement et simplement un environnement complet sans avoir besoin de passer trop de temps à lire une documentation, l'interface doit être minimale, et tirer partie des habitudes d'interactions qu'a déjà acquis l'utilisateur au contact des logiciels standard (éditeurs de textes, environnements de programmation, etc.).

Dans un but de lisibilité accrue des documents, la saisie et l'affichage des diverses expressions

entrant dans le composition d'une spécification formelle doivent pouvoir se faire en utilisant la présentation mathématique habituelle. Comme elle est constamment utilisée dans les articles scientifiques, c'est celle qui déroutera le moins les utilisateurs.

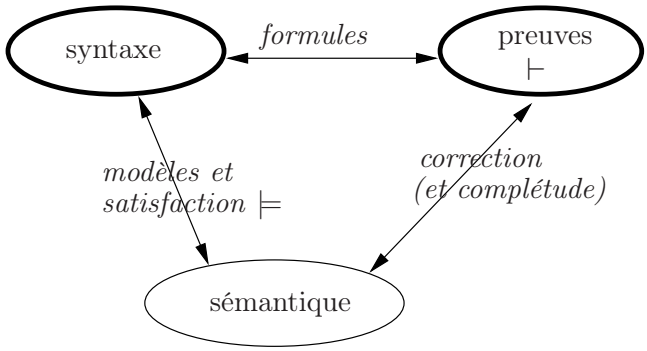
Toujours dans l'optique d'avoir un environnement simple d'utilisation, le méta-langage de description de formalisme doit être concis, lisible et particulièrement adapté à la génération d'analyseurs pour les langages de spécification formelle. Aussi bien dans le cas de la description d'un nouveau formalisme que dans celui de l'utilisation d'un formalisme existant, on doit pouvoir utiliser l'environnement par *mimétisme* en navigant simplement parmi des bibliothèques de langages de spécification, ou de composants de spécification, pour les adapter à notre cas particulier.

2 Fondements théoriques

MSPEC vise à accueillir une palette de langages de spécification formelle aussi large que possible. Il doit outiller : la définition de ces langages, l'écriture de composants de spécification en ces langages, et l'hétérogénéité des langages dans une même spécification. Pour ce faire, il faut naturellement délimiter ce que nous entendons par *langage de spécification formelle*.

Classiquement, trois aspects doivent être présents :

- une *syntaxe* qui définit rigoureusement ce qu'un spécifieur est en droit d'écrire.
- une *sémantique* qui définit mathématiquement le sens (l'ensemble des modèles) d'une spécification. Généralement l'ensemble des modèles d'une spécification est obtenu en au moins deux temps : une signature est issue de la spécification et on définit l'ensemble des modèles sur cette signature, puis on conserve les modèles qui satisfont les formules ("axiomes") et les contraintes (modularité par exemple) issues de la spécification.
- un *système d'inférences* qui fournit un moyen décidable de vérifier la validité d'une preuve à partir d'une spécification. Là encore, on passe généralement via des axiomes issus de la spécification et des règles d'inférences pour construire des arbres de preuve. On requiert naturellement que tout ce qui est prouvable est satisfait par tous les modèles (correction) ; la propriété inverse (complétude) est malheureusement rarement atteignable (souvent en raison des contraintes, par exemple l'encapsulation qui induit des règles d'induction structurelles intrinsèquement non complètes).



Parmi ces trois aspects, seulement deux relèvent réellement de MSPEC : syntaxe et preuves. En effet, MSPEC est un (méta)environnement de spécification et en tant que tel il ne peut bien sûr assister que des activités de "manipulations de symboles", par conséquent il n'assiste pas le chercheur dans sa définition d'une sémantique mathématique, ni dans les preuves de correction, de complétude, etc.

En fait, on reconnaît au travers de ces trois aspects la définition d'une logique (signatures, formules, modèles, satisfaction et inférences) [Mes89] à laquelle on ajoute des "contraintes". Ces contraintes sont généralement liées à la *structure* des spécifications qui interdit certains modèles

ne respectant pas, par exemple, l'indépendance des modules, l'autonomie des objets, etc. Elles induisent du même coup des inférences nouvelles au niveau des preuves ou de la génération de tests.

Les recherches actives du domaine des spécifications axiomatiques ont maintenant défini de manière relativement satisfaisante ce qu'est une logique [GB84] [Mes89]. Les notions de contrainte, de structure des spécification modulaires ou d'architecture des spécifications à objets et leur impact sur la sémantique et les preuves sont en revanche moins universellement définies [ST88a] [CoFI96]. Dans le cadre modulaire hiérarchique, [BCL96] [Cou98] propose une solution très élégante mais beaucoup reste à faire dans le cas général (par exemple objet).

Correlativement, MSPEC dispose d'une grande latitude pour définir ce qu'est l'architecture des spécifications. Pour nous, une spécification sera un *système* constitué de *composants* de spécification reliés par des *relations* d'utilisation de divers types. Définir complètement ce que sont chacune de ces trois notions reste du domaine des recherches en cours pour MSPEC, voir la section 4. En première approche,

- un composant devra contenir, en plus de sa signature et ses axiomes, des signatures importées et une ou plusieurs signatures exportées
- les spécifications étant par défaut hétérogènes, les relations d'utilisation doivent accepter des utilisations hétérogènes, éventuellement en produisant des obligations de preuves
- enfin les systèmes ne pourront être constitués qu'en respectant les conditions de "recollement" des éléments précédents (obligations de preuve, typage, ou autres).

D'autres relations entre composants devront également être gérées, par exemple le fait que plusieurs composants constituent des vues hétérogènes d'un même composant logiciel, qu'un système est un raffinement correct d'un composant plus abstrait, etc.

3 Outillage

Pour outiller les concepts exposés dans la section précédente il faut d'une part offrir les primitives de manipulation des composants, des relations, et des systèmes de composants, et d'autre part disposer d'une couche supérieure apte à définir et gérer plusieurs langages de spécification ainsi que contrôler l'hétérogénéité des langages au sein d'une même spécification.

Toutes ces activités requièrent en premier lieu d'éditer des textes formels, qu'il s'agisse des composants de spécification, des propriétés des relations entre composants, ou de la définition même d'un nouveau langage de spécification. Le premier outil indispensable de MSPEC est donc un éditeur de textes formels : MWX (voir sections 5 et 6). L'éditeur MWX est presque totalement finalisé, et il est essentiellement dédié à l'édition la plus ergonomique possible de formules et textes mathématiques.

L'édition des systèmes de composants avec leurs relations entre composants relève essentiellement de la manipulation de graphes. Cependant, dans notre cadre formel, l'expérience montre que de nombreuses annotations au sein des graphes, elles-mêmes des formules, sont nécessaires. Nous avons donc fait le choix d'un éditeur généraliste, et MWX peut indifféremment imbriquer des formules et des graphes.

Comme on souhaite manipuler des spécifications pour des études de cas réalistes, les préoccupations classiques en génie logiciel de réutilisation, cohérence, gestion de versions, etc conduisent à des bibliothèques de composants, systèmes, relations, langages de spécification, etc.

- Une bibliothèque de référence de composants classiques spécifiés dans des langages bien connus sera distribuée avec MSPEC afin de faciliter la "spécification par analogie" pour un utilisateur novice. Il en est de même pour une bibliothèque de langages classiques, relations standard, lemmes déjà prouvés, obligations de preuve, etc. Par ailleurs chaque utilisateur aura sa propre bibliothèque complémentaire.

– Les bibliothèques devront faciliter la réutilisation en offrant des fonctionnalités de recherche sur la forme des signatures, sur les propriétés des composants, sur leur appartenance à un système donné, sur le langage formel utilisé, etc.
– Comme dans tout atelier de génie logiciel, la cohérence des spécifications relève également de la bonne gestion de la bibliothèque : la destruction de composants, spécifications, langages, etc. doit être subordonnée au fait qu'ils ne soient plus utilisés par ailleurs ; et de même pour les opérations de modification, la gestion des obligations de preuve, etc.

La conception d'une spécification de grande taille induit une multitude d'actions élémentaires à effectuer pour en assurer la cohérence : obligations de preuve, vérifications des conditions autorisant certaines importations hétérogènes entre composants, composants restant à compléter, répercution d'une modification le long d'un système, etc. Empiriquement, on sait que ces actions élémentaires ne sont pas menées linéairement dans l'ordre ou elles arrivent, mais que certaines sont laissées en attente, voire même jamais effectuées. Disposer à tout moment d'un inventaire des actions pendantes pour un système donné est un service qui doit également être rendu par MSPEC.

Quels que soient le ou les langages de spécification intervenant dans un système, des outils génériques classiques dans le domaine des spécifications formelles doivent naturellement être disponibles par défaut dans MSPEC :
– générateurs de tests fonctionnels
– assistants ou vérificateurs de preuves
– aide à la conception de prototypes
– le cas échéant, animation de scénarios à partir de spécifications formelles, visualisation de liens causes / effets, etc.

Ces outils seront toujours proposés dans des menus standard même si pour certains langages de spécification ils sont inopérants parce qu'encore indisponibles (voir la section 4.2).

La généricité de MSPEC ne peut être obtenue que grâce à un méta-langage de spécification (MLS) dont le rôle est de spécifier la syntaxe d'un langage de spécification. A partir de ce langage on pourra automatiquement extraire un vérificateur de syntaxe naturellement, mais également :
– un mode d'édition de MWX dédié à ce langage
– de la même façon, un éditeur de règles d'inférences
– un vérificateur de preuves associé
– un générateur de tests
– etc.

Les premières études pour concevoir le langage MLS sont encourageantes (une version préliminaire de ce langage est exposée dans [Dav97]). La version actuelle reste cependant complexe (par exemple pas moins de trois types différents d'itérations y sont définis) et de nouvelles recherches pour simplifier ce langage sont souhaitables [Lau99].

Remarquons que MLS étant lui-même un langage formel, les outils mentionnés plus haut lui seront tout naturellement appliqués : un mode d'édition pour MLS, etc.

Pour tous ces outils nous feront un choix systématique de simplicité. En effet, le rôle de MSPEC est en premier lieu de démontrer la pertinence d'un nouveau formalisme de spécification avec le minimum d'investissement technique. Cela a pour conséquence, par exemple dans le cas des démonstrateurs de théorèmes, que l'utilisateur se contentera *a priori* d'un simple vérificateur de preuves avec des stratégies génériques ne tirant pas automatiquement parti des subtilités de la logique sous-jacente. Notre but n'est pas d'ultra-outiller des logiques particulières, mais d'offrir rapidement un atelier d'expérimentation dédié à un langage en cours de validation.

4 Premières expériences

4.1 Premier prototype

Dire que "MSPEC est aux langages de spécification formelle ce que CENTAUR est aux langages de programmation" serait réducteur, à la fois pour CENTAUR qui peut s'appliquer aussi à des langages de spécification [ASF+SDF96] et pour MSPEC qui a pour ambition de "pré-outiller" fortement tout nouveau langage de spécification (gestionnaire de composants et d'architecture de spécification, plusieurs types de relations entre composants, vérificateur de preuves, générateur de tests, etc.). Il n'en reste pas moins que la proximité des buts (manipulations et définitions de langages) et des concepts de base pour les atteindre rendait logique de faire appel au savoir faire de CENTAUR pour étudier la faisabilité de MSPEC.

Ce premier prototype écrit avec CENTAUR fut un succès en ce sens qu'il nous a à la fois convaincu de la faisabilité de MSPEC et de son intérêt spécifique.

En premier lieu, il convient de saluer la souplesse de CENTAUR : nous avons confié le premier prototype de MSPEC à un stagiaire ingénieur de l'INT et il a mené à bien ce projet en moins d'un semestre. Le prototype utilisait directement les langages de définition de langages offerts par CENTAUR [KLM83] et offrait un environnement pour une extension *ad hoc* des spécifications algébriques classiques. Parmi les outils développés, on peut mentionner :

- un éditeur assisté de spécifications,
- un éditeur de schémas de règles d'inférence,
- un embryon de gestionnaire de modules de spécification (restreint au cadre des spécifications hiérarchiques sans paramétrisation),
- un assistant à la preuve en logique du premier ordre équationnelle avec quelques facilités de visualisations des preuves inspirées de Larch [GG89], des possibilités de retour arrière en cours de démonstration, etc.

La leçon principale que nous avons tirée de cette expérience est que le fait de se limiter à des langages de spécifications doit permettre de simplifier considérablement la présentation, l'utilisation et les primitives mêmes de manipulation de l'environnement.

Au niveau de l'ergonomie par exemple, les besoins de visualisation de formules, de textes ou de figures sont devenus clairement identifiables (et identifiés, voir section 4.2 ci-dessous) et peuvent être obtenus sans aucune prolifération de fenêtres, avec une présentation standardisée (place relative des fenêtres, etc.).

De même pour la standardisation des commandes utiles quel que soit le langage de spécification considéré : le seul fait d'avoir essentiellement affaire à des logiques typées induit des modes opératoires restreints face auxquels la généralité de CENTAUR donne une impression d'"usine à gaz".

Au niveau des meta-langages pour définir les langages de spécifications par exemple, on remarque que le typage intervient dans les formules d'une manière spécifique aux logiques typées. Une grammaire standard est somme toute assez lourde à manipuler dans ce cadre, et d'un langage à l'autre on constate des "redites" qui démontrent l'intérêt d'un *méta-langage de spécification* (MLS) dédié à la définition de la syntaxe de logiques typées. Par ailleurs les grammaires standard sont peu fidèles aux habitude de définition rencontrées classiquement dans les articles de recherche en spécification axiomatique.

Enfin grâce à ce premier prototype nous avons pu tester le comportement des spécifieurs expérimentés de l'équipe face à l'outil. Il est entre autres devenu tout à fait évident que la présentation de formules sur un mode alphanumérique est un facteur considérable de lassitude, et finalement d'abandon de l'outil au profit du papier. . .Présenter et éditer les formules exactement comme on peut les voir dans un manuel mathématique s'est avéré clairement le seul moyen de mener à terme des spécifications de taille crédible. En somme, la concentration que demande l'écriture et la compréhension des formules ne souffre pas un effort supplémentaire de "parsing

mental".

4.2 Début du projet

L'expérience acquise lors de la réalisation du premier prototype nous a conduit à spécifier l'interface de MSPEC en évitant la prolifération des fenêtres et des menus. On a donc décidé d'étudier l'interface *en premier lieu* [Sil98].

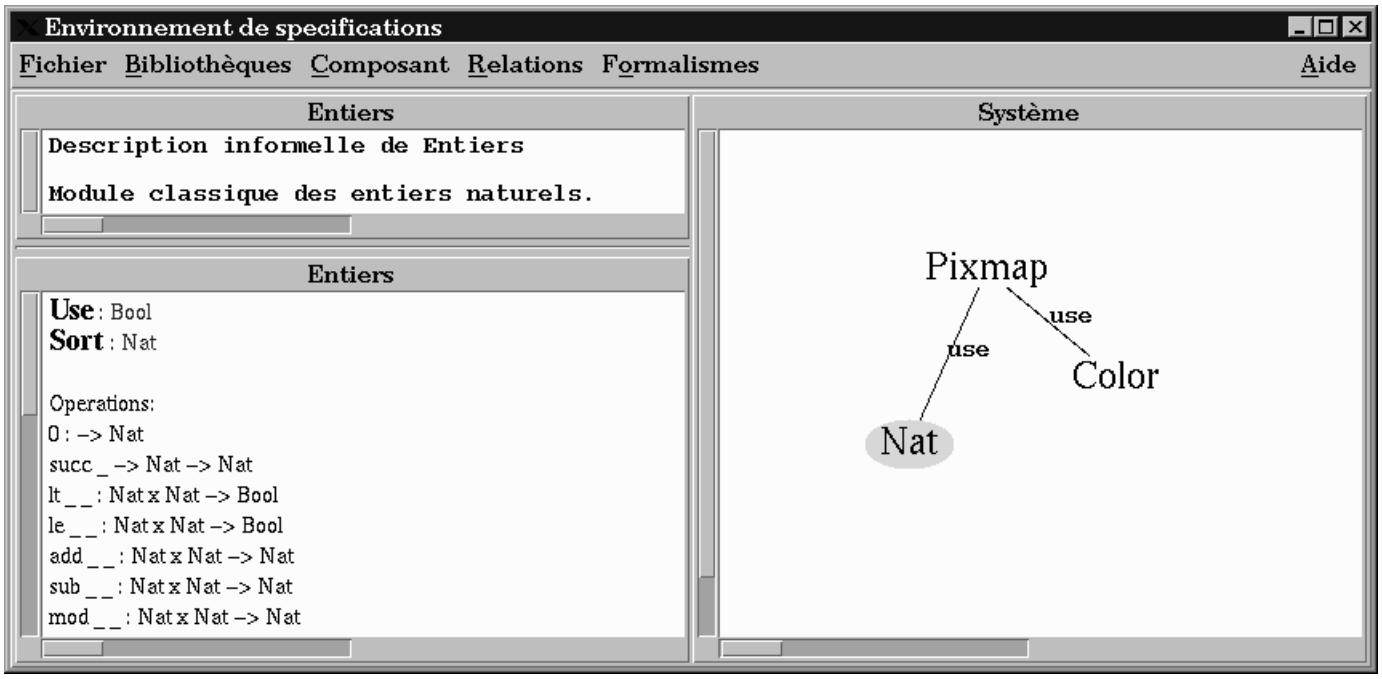


FIG. 1 – Maquette de l'interface de MSPEC.

L'environnement a deux modes de fonctionnement : on peut travailler soit sur une spécification, soit sur la définition d'un nouveau langage formel. L'interface de ces deux modes est très semblable :

- D'expérience, une spécification est toujours constituée de composants combinés en "système". MSPEC est conçu pour éditer des spécifications structurées en favorisant la réutilisation de parties déjà spécifiées (importation de composants, raffinements, etc.). Comme on n'a de visible à l'écran que les spécifications d'un composant à la fois, le système complet est représenté dans une zone sous la forme d'un graphe (les noeuds sont les composants et les arcs les relations). Le choix du composant actuellement édité pouvant se faire par sélection directe sur le graphe représentant le système complet. Par ailleurs on a toujours besoin d'une description informelle correspondant au texte formel qu'on est en train d'éditer. La fenêtre principale est donc découpée en trois zones.
- Dans le cas de la définition d'un nouveau langage la fenêtre graphique n'est pas toujours utile *a priori*. Cependant pour certains formalismes de spécification comme ETOILE il est intéressant d'avoir une vision graphique de la signature. L'utilité des fenêtres formelle et informelle ne fait en revanche aucun doute.

Dans les deux modes d'utilisation de MSPEC le découpage en trois vues de la fenêtre semble le plus ergonomique. Comme dans la quasi totalité des applications, une barre de menus déroulants est placée dans la partie supérieure de la fenêtre. Chacun des deux modes d'utilisation de MSPEC (édition de spécification / de formalisme) aura sa barre de menus type. Toujours dans

un souci d'ergonomie, les items correspondant à des opérations non applicables dans un contexte particulier resteront affichés, mais en grisé et non sélectionnables.

FIG. 2 – Une boîte de dialogue.

En l'état actuel seul le mode d'édition de spécification est simulé. On n'a pour le moment qu'une maquette de l'interface montrant la fenêtre principale, les diverses boites de dialogue, ainsi que leurs interactions.

Afin d'avoir un retour d'expérience rapide préalable à des développements généraux plus complexes, nous prévoyons de développer une instance particulière de l'environnement pour les ETOILE-spécification. L'interface précédemment définie sera réutilisée et complétée d'un vérificateur syntaxique (lié à la fenêtre d'édition formelle) pour ce formalisme.

5 Structure des objets graphiques

La faible lisibilité des spécifications formelles éditées dans les environnements classiques sous forme textuelle alphanumérique a toujours nuit aux méthodes formelles elles-mêmes. Si l'on veut une utilisation réelle en milieu industriel, il faut déjà supprimer l'effet rébarbatif de cette notation alphanumérique.

La notation mathématique standard (comme dans un livre), élaborée depuis des siècles, est la notation la plus lisible pour toute personne ayant une formation scientifique (ingénieur). Par conséquent on veut pouvoir éditer directement des formules mathématiques représentées à l'écran avec la typographie standard.

Une étude simple de la structure d'un texte mathématique montre qu'en fait il s'agit d'une *suite de lignes* contenant chacune des caractères alphanumériques standard et quelques formules comme $(\sum_{j \in \cup A_i} X_j)$. On se rend compte qu'on peut considérer cette formule comme une structure arborescente dont la racine est le caractère `sigma`, qui a un premier sous arbre qui est lui-même une ligne représentant la formule $j \in \cup A_i$ et un deuxième sous arbre qui est la ligne représentant X_j.

Cela conduit à une *notion étendue de caractère* qui peut être lui-même un arbre. Parmi les types de caractères étendus on a besoin en particulier de :

- caractères grecs
- caractères comportant des indices, exposants, bornes supérieures et inférieures, etc.
- matrices, tableaux, accolades multilignes, etc.
- racines, sommes, intégrales, valeur absolue, barre de règles d'inférences

– symboles graphiques, images, etc. (comme c'est par exemple le cas pour le langage dédié à la spécification du multi-modeleur géométrique en cours d'élaboration au sein du PPF regroupant les universités d'Evry, Poitiers et Strasbourg)

les caractères étant affichés dans diverses fontes, de diverses couleurs, surlignés, etc.

Par ailleurs les aspects de structuration des spécifications nécessitent un certain type d'édition de graphes de dépendances (tels les diagrammes UML, les structures modulaires des spécifications, etc.). Il s'agit de graphes orientés étiquetés. Les outils habituels d'édition de figures (comme `xfig`), parce que trop généraux, s'avèrent beaucoup trop lourds à utiliser dans le contexte de la représentation de l'architecture des spécifications. De plus, on ne peut pas exploiter la structure logique du graphe pour les outils de MSPEC.

On veut pouvoir aussi utiliser cette structure logique en cours d'édition. Par exemple on a besoin :

– d'étiqueter les arcs et les noeuds par des formules
– que lors de l'édition du graphe, les arcs restent attachés en temps réel aux noeuds
– que des sommets puissent contenir d'autres graphes, etc.

On considère également un graphe comme un caractère étendu. Cela conduit de manière naturelle à une imbrication arbitraire des différents types de caractères (formules en indice, matrices de formules, étiquettes des sommets de graphes, etc.)

6 Edition des objets formels par MWX

MWX est la continuation du *Widget* WIX développé dans [Amar96] ; c'est un composant d'interface basé sur le modèle de toolkit graphique (*XToolkit*) utilisé pour l'éditeur WIX. Il réalise les fonctionnalités d'édition des objets nécessaires à l'environnement MSPEC (section 5).

MWX est écrit en C++ et utilise le mécanisme de classes du langage : les différents types de caractères étendus sont implémentés sous la forme de classes C++. Le buffer d'édition de texte (objet de la classe `Editor`) est représenté de façon interne comme une suite de lignes (vecteur d'objets de la classe `Line`) ; les méthodes de la classe `Editor` représentent les fonctionnalités de base de l'éditeur (insertion, destruction, remplacement de lignes) ainsi que l'organisation de l'affichage des lignes et l'interface avec les composants externes (scrollbars, etc.).

La classe `Line` est une suite de caractères étendus (vecteur d'objets de la classe `Character`. Les méthodes de la classe `Line` implémentent les manipulations de caractères (insertion, destruction, positionnement, etc.) ainsi que l'affichage de la ligne elle-même.

La classe `Character` est polymorphe ; elle sert à représenter toutes les constructions typographiques que l'on voudra éditer. Ses diverses incarnations sont :

– caractère ISO standard avec sa fonte et sa couleur
– caractère complexe avec attributs (indice, exposant, bornes inférieure ou supérieure etc.)
– matrice bi-dimensionnelle où chaque élément est une ligne, avec une décoration (accolades, crochets, surlignage, etc.)
– images au format `jpeg`
– figures (graphes orientés étiquetés).

Toutes ces classes ont en commun des méthodes génériques pour donner le rectangle englobant le caractère, l'afficher dans la fenêtre, le traduire en Postscript, répondre aux clics de la souris, etc. Par exemple, la méthode d'affichage de la classe `Editor` appelle celle de la classe `Line` pour chacune des lignes actuellement visibles dans la fenêtre ; la méthode d'affichage de la classe `Line` appelant la méthode d'affichage de chacun des caractères qui composent la ligne en utilisant les informations de géométrie fournis par chacun de ces caractères.

Cette façon de faire permet d'ajouter facilement de nouveaux types de caractères dans MWX : il suffit d'écrire une nouvelle sous classe de `Character` qui implémente les méthodes nécessaires pour la classe `Line`.

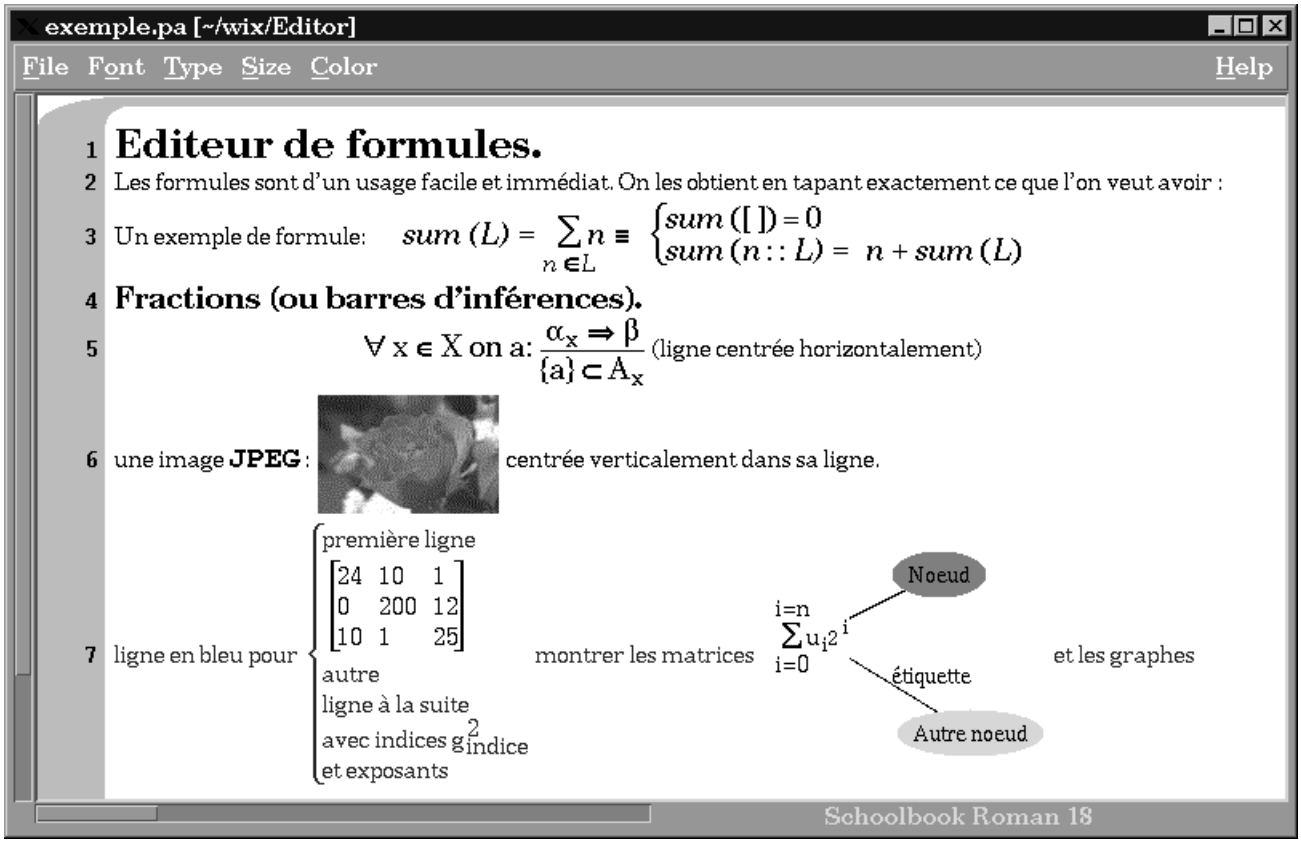


FIG. 3 – Exemple d'objets éditables par MWX.

Pour les caractères composés (caractères complexes et matrices) chacun des attributs est en fait de type ligne, ce qui permet d'avoir récursivement des indices d'indices, des matrices de formules, des graphes étiquetés par des images, et plus généralement n'importe quelle structure récursive formée à partir des caractères étendus de base. Cette représentation interne arborescente devrait faciliter le travail de l'analyseur syntaxique fourni par l'environnement pour valider une spécification.

L'utilisateur de MSPEC ayant à coup sûr besoin d'imprimer tout ou partie de ses spécifications, l'éditeur est capable de produire un fichier Postscript qui est l'exacte copie de ce qu'il y a à l'écran en utilisant les fontes et les fonctionnalités de dessin de haute qualité de Postscript. Ce fichier pourra soit être imprimé tel quel, soit inclus dans un document produit par un système de traitement de texte quelconque : en effet quasiment tous les formatteurs de texte savent insérer des fichiers Postscript dans un document.

Du point de vue des commandes, l'éditeur est très paramétrable : une partie des commandes sera directement interprétée par MWX (insertions, destructions de lignes, de caractères, etc.) tandis que l'autre partie des commandes sera interprétée par l'environnement (validation syntaxique, etc.) avec pour certaines un *feed-back* (changement de fonte ou de couleur par exemple). Certaines constructions représentées dans l'éditeur pourront même éventuellement être automatiquement validées par l'environnement (de façon analogue à ce qui se passe dans les éditeurs de programmes qui vérifient que les expressions sont correctement parenthésées par exemple).

L'éditeur est souple, non syntaxique strict, mais qui "connait" la syntaxe et aide l'utilisateur à éditer sa spécification, avec des squelettes de constructions du langage dont on n'a plus qu'à remplir les trous, un système d'auto expansion des noms (table des mots clés du langage, des

mots qui sont déjà présents dans la spécification actuellement éditée, ou dans un autre composant qui est d'une façon ou d'une autre référencé par cette spécification). On veut pouvoir admettre momentanément des constructions syntaxiquement incorrectes pour ne pas *contraindre* l'utilisateur, mais plutot lui *proposer* des constructions correctes qu'il devra compléter ; la validation se faisant explicitement ou automatiquement par la suite.

7 Quelques choix techniques pour l'environnement

MSPEC est un projet académique de longue haleine, pour pouvoir le développer jusqu'au bout avec peu de moyens, il faut éviter les écueils habituels :

- récritures du système complet dans un nouveau langage très différent du langage initial.
- se baser sur des langages ou des outils qui ne sont pas stables : il ne faut pas dépendre de versions successives trop fréquentes. On a donc intéret à utiliser un langage d'implémentation standard ainsi que des outils *standard* ou des outils dont on a la maîtrise complète.

Dans cette optique on a choisi d'implémenter directement des sous-ensembles de langages de programmation adaptés aux manipulations formelles, langages de la famille ML entre autres, d'une part pour assurer la portabilité maximale, et d'autre part pour faciliter la communication entre ces divers langages et l'interface de l'environnement.

Le langage qui a été retenu pour implémenter tous nos sous-systèmes est C++ pour le premier prototype ; Java est envisagé à moyen terme (sous réserve de stabilité) pour son indépendance de la plateforme. On utilisera aussi des outils standard tels que *lex* et *yacc* qui produisent des analyseurs écrits dans la plupart des langages, dont C++ et Java.

Pour avoir une IHM moderne et attrayante, on a choisi pour l'interface la toolkit graphique développée dans [Amar96] qui est basée sur le modèle objet standard proposé avec le système de fenêtrage *X Window*.

Enfin on souhaite que la procédure d'installation du logiciel MSPEC soit la plus simple possible (très peu de fichiers à installer, voire un seul si possible : l'exécutable) dans le but que les utilisateurs potentiels n'hésitent pas à tester MSPEC.

8 Conclusion

Le projet MSPEC s'insère dans l'ensemble des recherches sur les spécifications formelles axiomatiques effectuées depuis 1992 au sein de l'équipe de spécifications formelles de l'université d'Evry. Il s'agit de développer en un temps raisonnablement court des langages de spécifications *ad hoc* dédiés à des domaines d'application particuliers. Le "savoir faire" théorique dans ce domaine atteint maintenant un niveau qui permet d'outiller cette démarche. En retour, l'environnement MSPEC permettra une définition plus rapide et plus ergonomique des nouveaux langages dédiés.

L'existence de MSPEC, à la disposition des théoriciens des spécifications formelles, permetta une convergence plus rapide des définitions et des résultats vers des formalismes exploitables. Par conséquent non seulement MSPEC bénéficiera en temps réel des développements théoriques sur les méta-spécifications, mais réciproquement, ces développements théoriques seront motivés et dynamisés par MSPEC. Aborder les théories de spécifications formelles par le coté expérimental constitue en fait toute l'originalité des travaux menés à Evry dans ce domaine.

Cet article n'est qu'une description préliminaire de MSPEC. Il s'agit en fait d'un projet d'envergure à long terme, avec un nombre non négligeable d'outils devant se greffer sur l'interface décrite ici. Sur ce long terme, les recherches sous jacentes sur les méta-spécifications d'une part et l'hétérogénéité d'autre part modifieront tout naturellement les fondements de MSPEC. Nous avons

donc, dans la mesure du possible, pris soin d'éviter des hypothèses abusives sur ces fondements, ce qui contribue à une certaine incomplétude intentionnelle du cahier des charges de MSPEC.

Le méta-langage MLS de description des langages de spécification formelles jouera un rôle important dans la facilité d'utilisation de MSPEC. MLS n'est pas l'objet principal de cet article, c'est pourquoi il est peu explicité ici, mais il constituera un passage obligé très conséquent à ne pas négliger dans le projet MSPEC.

Dans cet article l'accent à été porté sur la condition *sine qua non* de réussite du projet MSPEC : une interface "naturelle" et d'utilisation simple. Nous avons souligné l'importance de motiver les utilisateurs par un mode de communication proche des habitudes mathématiques classiques bien connues de tous les ingénieurs. Cela a motivé le style "livre de mathématiques" offert par MWX.

Quiconque a utilisé des éditeurs classiques pour des textes contenant beaucoup de formules a acquis la dure expérience d'être obligé :

- soit d'abandonner le "WYSIWYG"[1] (par exemple en utilisant un langage comme LaTeX)
- soit d'abandonner la simplicité des commandes (par exemple en devant faire appel à de nombreuses boites de dialogue et manipulations de souris pour atteindre les symboles mathématiques, les indices, les exposants etc.)

L'apport de MWX est de concilier une vision mathématique des formules avec des commandes d'édition rapides et ergonomiques. Nous pensons en effet que l'un des obstacles majeurs à l'utilisation des spécifications formelles en génie logiciel est manque de lisibilité et de facilité de manipulation des formules mises en jeu dans ce type de spécifications. MSPEC apporte également une première réponse à ce problème, ce qui permettra d'aborder avec beaucoup plus de facilité des exemples de spécifications de taille crédible pour chaque nouveau langage.

Remerciements : Bon nombre de réflexions préliminaires sur la notion de composants au sein d'un système ont été menées avec Pascale Le Gall et Marc Aiguier. Le premier prototype a été écrit par Tony Santoro encadré par Catherine Dubois. La réalisation de la première version de MSPEC est en grande partie due à Elisabeth Da Silva. Enfin les développements théoriques sous jacents à MSPEC sont effectués par les membres de l'équipe *Spécifications formelles et transformations de programmes* du LaMI. C'est avec reconnaissance que nous tenons à les citer ici.

Références

[AB96] M. AIGUIER, G. BERNOT, *ETOILE-specifications : an Object Oriented Way of Specifying Systems*. Proc. of the Workshop on Proof Theory of Concurrent Object-Oriented Programming, Linz, Austria, p.50-57, J-P. Bahsoun, J. Fiadeiro, D. Galmiche, A. Yonezawa eds., 10th European Conf. on Object-Oriented Programming (ECOOP 96). 1996.

[ABB+96a] M. AIGUIER, J. BENZAKKI, G. BERNOT, S. BEROFF, D. DUPONT, L. FREUND, M. ISRAËL, F. ROUSSEAU, *ECOS : A Generic Codesign Environment for the Prototyping of Real Time Applications, "From Formal Specifications to Hardware-Software Partitioning"*. Current Issues in Electronic Modeling (CIEM), Issue No.8 : Hardware/Software Co-Design & Co-Verification, J.M. Bergé, O. Levia, J. Rouillard eds., Kluwer Academic Publishers, Dordrecht, The Netherlands. 1996.

[AL97] A. ARNOULD, P. LE GALL, *Some aspects of Test Data Selection from Formal Specifications*. Proc. of the Intl Conf. Quality Week Europe (QWE 97), Software Research Institute,

[1] What You See Is What You Get.

Brussels, Belgium. 1997.

[ASF+SDF96] ASF+SDF team, *Language Prototyping : An Algebraic Specification Approach*. World Scientific Publishing Co., AMAST Series in Computing, van Deursen, Heering, Klint (editors). 1996.

[Abr96] J-R. Abrial, *The B-Book — Assigning programs to meanings*. Cambridge University Press. 1996.

[Amar96] P. Amar, *Étude et réalisation de logiciels pour les environnements de programmation et de spécifications formelles*. Thèse de doctorat, Université Paris-Sud, Orsay, France. 1996.

[BC85] M. Bidoit, C. Choppy, *Asspegique : an integrated environment for algebraic specifications*. Proc. of the 1st International Joint Conference on Theory and Practice of Software Development (TAPSOFT CSE), Berlin, March 1985, Springer-Verlag LNCS 186 (H. Ehrig, C. Floyd, M. Nivat, J. Thatcher eds.), p.246-260. 1985.

[BCL96] G. Bernot, S. Coudert, P. Le Gall, *Towards heterogeneous formal specifications*. Proc. of the 5th Intl Conf on Algebraic Methodology And Software Technology (AMAST 96), Munich, Germany, Springer-Verlag LNCS 1101, p.458-472. 1996.

[BD94] Y. Bertrand, J-F. Dufourd, *Algebraic Specification of a 3D-Modeller Based on Hypermaps*. Computer vision, graphical model, and image processing, Vol.56, No.1, p.29-60,. 1994.

[BDE87] D. Bert, P. Drabik, R. Echahed, *Manuel de référence de LPG*. IMAG-LIFIA Technical Report, No.17. 1987.

[Ber94] G. Bernot, *Formal specifications and algebraic specifications*. Invited paper in Proc. of the 7th International Software Quality Week, May 1994, San Francisco, California, Software Research Institute, San Francisco, California, USA. 1994.

[COMPASS91] *Algebraic System Specification and Development – A Survey and Annotated Bibliography*. Springer-Verlag LNCS 501 (M. Bidoit, H-J. Kreowski, P. Lescanne, F. Orejas, D. Sannella eds.). 1991.

[CoFI96] CoFI group, *Common Framework Initiative*. EATCS Bulletin. 1996.

[Cou98] Sophie Coudert, *Cadre de spécifications hétérogènes*. Thèse de doctorat, Université d'Évry, France. 1998.

[DFNH93] C. Dubois, P. Facon, Q. Nguyen, N. Hadjrabia, *The centaur-VDM environment*. Tool demonstration, Intl conf. on Formal Methods Europe (FME 93), april. 1993.

[Dav97] E. David, *FORMAL*. LaMI Report, Université d'Évry, France. 1997.

[Daw91] J. Dawes, *The VDM-SL reference guide*. Pitman. 1991.

[EM85] H. Ehrig, B. Mahr, *Fundamentals of Algebraic Specification 1. Equations and initial semantics*. EATCS Monographs on Theoretical Computer Science, Vol.6, Springer-Verlag. 1985.

[GB84] J.A. Goguen, R.M. Burstall, *Introducing institutions*. Proc. of the Workshop on Logics of Programming, Springer-Verlag LNCS 164, p.221-256. 1984.

[GG89] J.V. Guttag, S. Garland, *An overview of LP, the Larch Prover*. Proc. of the 3rd Intl Conf on Rewriting Techniques and Applications (RTA), p.137-151, Springer-Verlag LNCS 355. 1989.

[GH86] J.V. Guttag, J.J. Horning, *Report on the LARCH shared language*. Science of Com-

puter Programming Journal, Vol.6, No.2, p.103-134. 1986.

[GTW78] J.A. Goguen, J.W. Thatcher, E.G. Wagner, *An Initial Algebra Approach to the Specification, Correctness, and Implementation of Abstract Data Types*. Current Trends in Programming Methodology, R.T. Yeh ed., Printice-Hall, Vol.IV, p.80-149. 1978.

[JR92] I. Jacobs, L. Rideau, *A CENTAUR Tutorial*. 1992.

[KLM83] G. Kahn, B. Lang, B. Mélèse, *METAL : a formalism to specify formalisms*. Science of Computer Programming (SCP), Vol.3, p.151-188, North Holland. 1983.

[Lau99] R. Laurence, Mémoire de DEA à paraître, Université d'Évry, France. 1999.

[Mes89] J. Meseguer, *General logics*. Proc. Logic. Colloquium'87, North-Holland, Amsterdam. 1989.

[SELL97] J. Souquières, N. El Cadi, T. Lambolais, N. Lévy, *PROPLANE : Aide au développement de spécifications formelles : Application aux protocoles de communication*. Actes de séminaires Action Scientifique "Logiciels pour les télécommunications", Revue France-Télécom No 14, Mars 1997, pp. 21-43. 1997.

[ST88a] D. Sannella, A. Tarlecki, *Specifications in an arbitrary institution*. Information and Computation (formerly Information and Control), Vol.76, p.165-210. 1988.

[Sil98] E. Da Silva, *Prototype d'un atelier graphique de spécifications formelles structurées et hétérogènes*. Mémoire de DEA, Université d'Évry, France. 1998.

[Spi89] J.M. Spivey, *The Z notation : a reference manual*. Prentice Hall. 1989.

[Wir90] M. Wirsing, *Algebraic specifications*. HandBook of Theoretical Computer Science, North Holland, p.675-788. 1990.