Projet Hamam : Rapport Final

Projet Hamam

2 mai 2007

1 Introduction

1.1 Présentation

Hamam est un acronyme récursif signifiant Hamam Ajoute des Modules À Mathemagix. Mathemagix est un CAS (Computer Algebra System) libre développé par Joris Van Der Hoeven. Au même titre que Maxima, Maple, Mathematica, Ginac ..., Mathemagix a pour but de fournir un outil de calcul scientifique complet, homogène et simple à utiliser.

Mathemagix se place donc dans une offre logicielle nombreuse, mais a la particularité, outre d'être libre, d'être modulaire : une base très souple, sous la forme d'un interpréteur, est fournie et la résolution d'un problème particulier se fait à travers l'utilisation judicieuse de modules qui viennent se greffer dessus. Cette approche évite l'utilisation parfois malaisée et souvent hétérogène de divers logiciels spécialisés.

De plus cela permet à l'utilisateur averti d'ajouter ses propres modules, à l'aide de l'interface `C++` proposée.

Mathemagix a de plus été conçu pour fonctionner de paire avec TeXmacs qui est à la fois un éditeur de document scientifique et un environnement permettant de travailler avec plusieurs CAS. Le but est d'offrir une facilité d'utilisation maximale pour tous les utilisateurs potentiels, qu'ils soient des élèves, des étudiants ou des chercheurs.

1.2 Motivation

Lorsque l'on travaille à la résolution d'un problème et que l'on a besoin d'un CAS, on est souvent rebuté par l'inadéquation entre les outils proposés et nos besoins. Certes, dans le cadre des CAS libres, il est ultimement possible de les modifier pour les adapter à ce que l'on veut faire, mais l'utilisateur possède rarement les compétences et le temps requis pour de telles modifications. L'objectif de Hamam est donc d'anticiper ces besoins et de rendres les fonctionnalité correspondantes disponible dans Mathemagix à travers l'ajout de modules. Le but n'est pas tellement de le faire progresser pour lui ajouter ce qui lui manque par rapport aux autres CAS, qu'ils soient libres ou non, mais surtout d'ajouter des fonctionnalités novatrices.

1.3 Le choix de Mathemagix

Le choix de Mathemagix s'est fondé sur plusieurs critères. La première décision à prendre était de savoir si on partait de rien ou s'il on essayait d'améliorer quelque chose d'existant. Étant donné la courte durée de notre projet, partir de rien promettait de ne rien faire de vraiment intéressant, et de se limiter aux bases, déjà partout présentes. Il a donc été décidé

d'améliorer un CAS existant. Ce dernier devait donc être sous licence libre, notamment pour des raisons pratiques d'accès au code. Enfin, l'idée de devoir s'immerger dans le code complexe d'un CAS promettait d'être long et difficile. MATHEMAGIX proposait une structure modulaire, pensée pour accueillir des contributions externes à la manière de plugins. Cela promettait une intégration simple du travail, et une concentration des efforts sur le coeur des choses. Même si on va voir que ce n'a pas été le cas, cette supposition, fausse, a été un facteur très motivant. Enfin, les membres voyaient en MATHEMAGIX un projet porteur, et souhaitaient y participer.

1.4 Thèmes de travail

Il n'existe pas à l'heure actuelle de CAS permettant de manipuler facilement des arbres et des graphes, qui sont pourtant des structures fondamentales en informatique. L'une des quatre parties du projet est entièrement dédiée à cela.

L'une des autres lacunes des CAS existants est l'absence de certification des décimales. Cette fonctionnalité sera bientôt présente dans MATHEMAGIX, indépendamment de HAMAM. Malheureusement, MATHEMAGIX connaît de graves lacunes en termes de fonctionnalités mathématiques : l'intégration, les transformations de Fourier et la factorisation des polynômes ne sont pas implémentées. Un pan entier de HAMAM s'intéresse à l'implémentation de ces algorithmes, de sorte qu'au bout du compte, il existera un CAS où ces algorithmes produisent des résultats aux décimales certifiées.

De plus, le grapheur de TeXmacs est insuffisant et l'intégration de TeXmacs avec MATHEMAGIX peut être améliorée. HAMAM a donc aussi travaillé à écrire un grapheur 2D et 3D, et à l'amélioration de l'intégration de MATHEMAGIX dans TeXmacs, notamment au niveau de la visualisation des résultats.

Enfin, il n'était pas facile d'écrire des extensions de MATHEMAGIX. Une interface `C++` est présente, mais elle reste compliquée à utiliser. HAMAM s'est donc donné pour objectif de créer une interface `OCaml`, de sorte que les développeurs et les utilisateurs puissent facilement ajouter des extensions écrites dans ce langage.

Au final, l'objectif est de fournir un CAS qui permette de satisfaire des besoins qui ne sont pas encore satisfaits par les autres CAS, par une combinaison de modules complémentaires.

1.5 Description

HAMAM se décompose en cinq workpackages nettement distincts :

1. le WP1, qui développe une librairie C++ de manipulation d'arbres et de graphes.
2. le WP2, qui développe les modules de mathématiques, qui comprennent des algorithmes d'intégration, les transformées de Fourier rapides, et la factorisation de polynômes ainsi qu'une implémentation de la représentation des grands nombres.
3. le WP3, qui implémente un grapheur 2D et 3D, ainsi qu'une intégration de MATHEMAGIX dans TeXmacs ;
4. le WP4, qui s'occupe de construire une interface pour pouvoir ajouter des extensions écrites en `OCaml` à MATHEMAGIX ;
5. le WP5, qui s'occupe de superviser la documentation, la communication interne et externe.

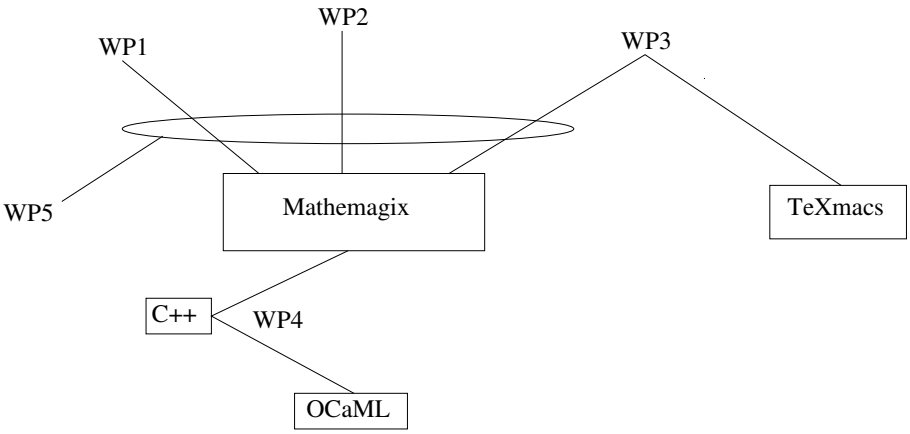

FIG. 1 – Architecture du projet HAMAM

2 WP1 : GT : Graphs & Trees

Responsable : Adrien Friggeri.

Membres : Fabien Benureau, Rémy Bergasse, Fabien Givors, Chantal Keller, Loïc Magnan, Lionel Rieg, Julien Provillard, Pascal Vanier.

2.1 Présentation

Faisant le constat qu'il n'existait aucun logiciel de calcul formel permettant d'utiliser de manière simple et intuitive des structures d'arbres, de graphes ou d'automates, ce *workpackage* visait à l'origine à fournir une librairie générique d'outils informatiques utilisable dans Mathemagix. Le public ciblé par ce logiciel étant relativement étendu et ne possédant pas forcément de connaissances poussées en informatique, il était nécessaire de fournir une syntaxe claire et simple permettant de créer et d'utiliser rapidement une structure de donnée.

2.2 Changement d'orientation

Néanmoins, cela n'a pas eu lieu. Devant les difficultés rencontrées lors des différents essais d'interfaçage avec Mathemagix (*glue* de la librairie GT au cœur de Mathemagix, se reporter au paragraphe 6.2.3 pour plus de détails), il a été décidé d'un changement d'orientation du workpackage : au lieu d'ajouter le support des arbres, des graphes et des automates à Mathemagix, le WP1 allait d'abord se concentrer sur l'écriture d'une librairie `C++` indépendante. De plus, en raison d'échéances à cours terme, nous n'avons pu mener à bien le développement du support des automates.

Il n'existe à l'heure actuelle aucune librairie `C++` implémentant les structures d'arbres et de graphes et offrant une possibilité d'utilisation très intuitive.

Il existe en revanche certaines librairies dédiées à l'une ou l'autre de ces structures. Citons parmi elles la librairie **BGL**[1] qui contient un module de graphes relativement complet ou la librairie **Tree.hh**[6] pour les arbres.

Le principal défaut des ces librairies est leur extrême spécialisation, et le fait qu'il n'existe rien pour les interfacer les unes avec les autres de manière transparente pour l'utilisateur. Par exemple, pour obtenir l'arbre couvrant minimal d'un graphe créé avec **BGL** en tant qu'arbre de **Tree.hh**, l'utilisateur devra lui-même écrire les fonctions de conversion. Ajoutons à cela que ces librairies n'ont pas été au départ conçues pour travailler les unes avec les autres, et donc leur utilisation conjointe n'est pas optimale.

2.3 Objectifs finaux

GT vise donc à implémenter une librairie générique gérant ces structures de données, permettant une utilisation conjointe très intuitive et transparente de celles-ci. L'idée étant de fournir une librairie utilisable aussi bien par des développeurs `C++` chevronnés que par des débutants.

Pour cela, nous proposerons pour chacune des structures de données les fonctionnalités suivantes :

- création et modification de la structure simplifiée et claire
- possibilité d'utiliser des parcours de structures *clés en main*
- possibilité de faire des conversions de structures de manière transparente

2.4 Réalisations

2.4.1 Classe graphes

Nous avons à l'heure actuelle la structure de graphe, avec possibilité d'utiliser des graphes orientés ou non orientés, et un certain nombre de fonctions utilitaires (création, suppression, modification de graphes, de noeuds, d'arêtes) et des itérateurs de parcours en profondeur et en largeur sur le graphe. Quelques algorithmes classiques sur les graphes ont également été implémentés. De plus nous disposons de fonctions d'importation et d'exportation vers le format dot.

Par ailleurs, l'accent a été mis sur le développement de la notion d'*application*. En effet, partant du principe que l'utilisateur doit pouvoir implémenter des algorithmes sur les graphes rapidement et efficacement, nous fournissons un cadre de travail intuitif.

Les applications sur les graphes permettent d'appliquer une fonction à chacun des noeuds, dans un ordre de parcours défini. Par défaut il existe trois types de parcours : en largeur, en profondeur, et par ordre d'insertion des noeuds, mais l'utilisateur a la possibilité de définir des parcours personnalisés. La fonction appliquée à chacun des noeuds peut prendre en argument un ensemble de `flags`, qui peuvent servir à marquer les noeuds.

Les applications permettent d'implémenter des algorithmes variés, en utilisant des fonctions sur les noeuds de types variés : il est possible de prendre une entrée en plus du noeud, il est aussi possible d'accumuler au fur et à mesure les résultats de la fonction appliquée au noeud. Enfin, il est possible, en utilisant les parcours personnalisés, de redéfinir les parcours en largeur et en profondeur, par exemple, ce qui en fait un outil puissant, permettant d'implémenter un grand nombre d'algorithmes de graphes de manière intuitive ; le code suivant, issu de la documentaition développeur, calcule la somme des poinds de noeuds d'un graphe, en le parcourant en profondeur.

```
typedef Node<int,int> t_node;
typedef Graph<int, int> t_graph;
typedef Application<int, int> t_app;

int sum_aux(t_graph* graph, t_node* node, int ac)
{ return node->get_value() + ac; }; // on renvoie l'accumuluteur mis à jour

int sum(t_graph* graph)             // Le déclarateur de l'application
{
t_app* app = new t_app(graph);      // on crée l'application
app->set_traversal("dfs");          // on demande un parcours en profondeur
return app->exec<int>(&sum_aux, 0); // on l'exécute
};
```

L'accent a été mis, lors du développement, sur la simplicité pour l'utilisateur final, qu'il soit aguerri ou non. Pour cela, nous avons adopté des conventions de nommage cohérentes, transverses aux différentes parties de la librairie.

2.4.2 Classe arbres

Dans un premier temps sont implémentées les fonctions de base concernant les arbres en général. Nous avons à l'heure actuelle un certain nombre de constructeurs et destructeurs

d'arbres, de fonctions pour modifier des noeuds , les déplacer, ou encore modifier l'arbre lui-même.

Le but est de pouvoir ensuite implémenter des fonctions utilitaires sur les arbres : rotation, parcours, ajout ou suppression de noeuds ou d'arbres ; et également, surcharge de certains opérateurs, comme l'égalité.

Des itérateurs préfixe et postfixe sur les noeuds sont disponibles. Mettre en place d'autres itérateurs est facile en utilisant les applications, dont l'interface, à quelques spécialisations près, est identique à celle des graphes. Il reste à implémenter des itérateurs sur les noeuds des arbres, pour les parcours préfixe, infixe et postfixe. Les fonctions d'importation et d'exportation vers le format dot et les graphes sont disponibles, assurant une souplesse maximale à la librairie.

2.4.3 Génération aléatoire de graphes

Pour l'expérimentation, les statistiques, ou la vérification d'hypothèses, il peut être utile de pouvoir générer des graphes aléatoirement, avec une efficacité correcte de construction. Une librairie de fonctions de génération aléatoire de graphes a donc été écrite. Le but était de pouvoir générer un graphe avec probabilité uniforme sur un sous ensemble des graphes défini par un ensemble de propriétés simples.

En pratique, pour construire un graphe tiré aléatoirement dans l'ensemble des graphes à n sommets, on effectue la construction simple qui consiste à ajouter chaque arête avec une probabilité 1/2.

Pour construire un graphe tiré aléatoirement dans l'ensemble des graphes à n sommets et m arêtes, le calcul du choix des arêtes se fait selon un algorithme dépendant de n et m. En effet, si m est petit devant n^2, il est plus efficace de tirer successivement les m arêtes aléatoirement parmi toutes les arêtes possibles, en annulant le tirage en cas de collision avec une arête déjà construite. En revanche, dans le cas contraire, les collisions étant trop nombreuses, on parcourt l'ensemble des arêtes possibles, qui sont construites avec une probabilité calculée au fur et à mesure en fonction du nombre d'arêtes restant à construire. La complexité de la construction est en $O(tailledugrapheconstruit)$.

On donne aussi une fonction pour créer des graphes en fixant l'arité a des sommets. Pour chaque sommet, on choisit a sommets parmi n choix possibles. Là encore, on utilise la méthode précédente qui consiste à choisir en premier lieu l'algorithme le plus efficace puis à effectuer la construction selon l'algorithme choisi. Là encore, la construction se fait en $O(tailledugrapheconstruit)$.

2.4.4 Documentation

Une documentation utilisateur a été faite et rendue disponible sur le wiki, présentant par l'exemple le fonctionnement de la librairie. De plus, un effort particulier à été entrepris pour doxygéner et documenter le code produit.

3 WP2 : Maths

Responsable : Olivier Schwander.

Membres : Chantal Keller, Pascal Vanier, Nicolas Estibals, Pierre-Etienne Meunier, Lionel Rieg, Tarik Kaced, David Grosdemange.

3.1 Objectifs initiaux

Un nouveau système de calcul formel se doit de proposer un jeu d'outils mathématiques de base. Mais, en l'état actuel du développement, certaines fonctionnalités majeures d'un CAS ne sont pas encore implémentées, comme l'intégration formelle ou la factorisation de polynômes.

D'autre part, une particularité de Mathemagix sera en outre de gérer les nombres réels. Cependant tous les calculs n'ont pas besoin d'être gérés avec une précision arbitraire. Il est donc intéressant de proposer à l'utilisateur des niveaux de précision intermédiaire entre les `float` et les `double` du système et les nombres réels, en particulier les quadruples et les octuples.

Un autre axe de développement est de rajouter des fonctions qui vont au-delà de l'acception traditionnelle du terme «logiciel de calcul formel» et qui se rapprochent plus de fonctionnalités offertes par des logiciels comme MATLAB (pour le traitement de données) ou LabVIEW (pour la mesure et l'acquisition de données). Cela se traduit dans Hamam par des travaux sur la FFT ainsi que sur un système acquisition et de génération de signal à l'aide d'une carte son.

3.2 Thèmes de travail

3.2.1 Intégration

Développeur : Chantal Keller

Présentation Tout logiciel de calcul formel se doit de posséder une fonction d'intégration.

Lorsque le projet Hamam s'est rattaché à Mathemagix, l'intégration n'était pas du tout implémentée. C'est pourquoi nous avons décidé de lui ajouter un module d'intégration, en restant cependant modeste dans un premier temps (intégration des polynômes et des fractions rationnelles tout d'abord, algorithme de Risch ensuite...), comme expliqué dans le paragraphe *Implémentation de l'intégration*.

Cependant, le concepteur de Mathemagix, Joris van der Hoeven, a continué à développer son logiciel pendant la période du projet, et a sorti une nouvelle version de Mathemagix contenant cette base de l'intégration. Mais cette version n'est sortie que pour les deux dernières semaines du projet, alors que l'intégration avait commencé à être implémentée par cette partie du WP2. De plus, M. Van der Hoeven n'avait pas tenu informés les membres du projet de cette avancée. C'est pourquoi nous présentons quand même notre travail.

Avec la nouvelle perspective qui s'offrait à nous, les possibilités d'implémenter des algorithmes d'intégration performants semblaient intéressantes. Cependant, il s'est avéré que l'intégration ne fonctionnait pas correctement dans Mathemagix : en effet, la fonction d'intégration `integrate` n'est jamais appelée...

Nous avons donc essayé de déboguer le programme, comme expliqué dans le paragraphe *Fonction d'intégration*, malheureusement sans succès pour l'instant.

Implémentation de l'intégration Vue la structure de MATHEMAGIX, cette implémentation devait commencer par la mise en place de la glue[1].

C'est cette partie qui s'est avérée la plus contraignante, aussi bien pour le WP2 que pour tous les workpackages. De fait, elle a pris beaucoup de temps à mettre en place pour écrire le prototype de la fonction d'intégration. De plus, la glue avait déjà été implémentée dans l'ancienne version de MATHEMAGIX, et il a fallu la porter dans la nouvelle version, d'où une perte de temps supplémentaire.

Pour le codage, nous avions choisi de nous pencher dans un premier temps sur l'intégration des polynômes, qui est une première étape nécessaire à la bonne mise en place d'une fonction d'intégration.

Un nouveau problème s'est alors posé : l'accès aux polynômes dans MATHEMAGIX. En effet, MATHEMAGIX possède un système de typage fort, et en particulier un type pour les polynômes. Il s'est cependant avéré difficile, voire impossible d'accéder à ce type dans MATHEMAGIX, c'est-à-dire par exemple de créer un polynôme.

Cependant, une partie de bibliographie avait été faite pour l'intégration : [8]

Fonction d'intégration L'intégration est certes implémentée... mais ne fonctionne pas dans MATHEMAGIX tel qu'il est implémenté par M. van der Hoeven. En effet, lorsqu'on essaie d'utiliser `integrate` dans MATHEMAGIX, la fonction interne d'intégration n'est pas appelée et l'expression est renvoyée telle qu'elle.

Nous avons essayé de comprendre comment les fonctions étaient appelées dans MATHEMAGIX, en prenant à nouveau l'exemple de la dérivation. Nous avons essayé de doubler toutes les fonctions faisant appel à la dérivation en les prenant pour modèle pour l'intégration, mais malheureusement sans succès pour l'instant.

3.2.2 Quadruples - Octuples

Développeur : Nicolas ESTIBALS

Présentation Les quadruples et les octuples sont des formats définis par la norme ieee754r. L'intérêt pour MATHEMAGIX de les implémenter est qu'il permettent d'augmenter la précision des calculs au besoin sans pour autant utiliser la bibliothèque `mpfr` qui est un peu lourde, puisqu'elle permet d'utiliser des précisions arbitraires, et nuit à la performance du CAS. En effet, les quadruples peuvent être implémentés de façon spécifique et efficace.

Réalisation Une étude bibliographique a été menée (notamment dans *The Art of Programming* [9]). Le principe simple d'implémentation est de couper le quadruple en deux doubles (et récursivement pour les octuples) représentant la partie haute et la partie basse du nombre.

[1] pour en savoir plus sur la glue, voir 6.2.3

Il existe une bibliothèque les implémentant ainsi qu'une série de fonctions s'appliquant dessus. Cette bibliothèque est codée en `Fortran`. Pour pouvoir l'adapter à MATHEMAGIX et pour des raisons d'efficacité, elle a été réécrite en `C++`.

Implémentation Une classe des quadruples a été définie ainsi que les opérateurs de base sur les quadruples. Tout ceci a été glué dans MATHEMAGIX mais sans succès pour des raisons des défauts de lien à la compilation.

Les performances de la classe `quad` sont relativement décevantes : une multiplication de deux quadruples est environ 25 fois plus lente qu'une multiplication de deux `double`. Cela est en partie dû à la lourdeur du processus de surcharge des opérateurs. Cependant les performances restent raisonnables et l'on peut partir de ce travail pour injecter du code assembleur optimisé.

Les octuples n'ont pas été implémentés du fait qu'ils utilisent les `quad` : on aurait eu les mêmes problèmes de performances et donc le temps de calcul n'aurait pas été raisonnable (par rapport à `mpfr`).

3.2.3 Interface avec des systèmes physiques

Développeur : Pierre-Etienne MEUNIER

Présentation Nous avons interfacé MATHEMAGIX avec un code `C++` qui gère des entrées/sorties matérielles sur une carte son, pour pouvoir faire du traitement du signal avec les fonctions mathématiques d'un CAS, comme la FFT, l'intégration, la dérivation. Nous arrivons à l'heure actuelle, en utilisant des outils rendus accessibles dans MATHEMAGIX, à générer des signaux sur une carte son, donc dans une plage de fréquences allant de 20 Hz à 20 kHz. Le but est à terme de permettre à un expérimentateur de remplacer un générateur de signaux par un outil plus puissant et plus souple d'emploi pour travailler sur un système physique, par exemple en électronique.

Implémentation Dans le cadre du projet, nous nous limitons à des entrées-sorties sur la carte son d'un ordinateur personnel. Cette approche permet de valider notre interface avec MATHEMAGIX sans se préoccuper de problèmes de pilotes de cartes d'acquisition. L'implémentation utilise la librairie SDL pour l'interface matérielle, et effectue des appels aux autres fonctions de MATHEMAGIX pour calculer les valeurs des fonctions.

Compte tenu des impératifs de temps nous avons renoncé à implémenter des fonctions de traitement (FFT, entre autres) à l'intérieur de MATHEMAGIX. En effet les problèmes rencontrés sont les mêmes que pour tous les autres workpackages au niveau de la glue.

3.2.4 Thèmes abandonnés

Factorisation En accord avec l'esprit de MATHEMAGIX, la partie travaillant sur la factorisation d'entiers et de polynômes devait initialement utiliser la bibliothèque PARI/GP.

Au vu des difficultés rencontrées par les autres groupes pour la glue et compte tenu du fait que le seul travail de ce groupe aurait consisté à écrire l'interface avec MATHEMAGIX, nous avons préféré renoncer à cette partie.

FFT L'implémentation de la transformée de Fourier parallèle devait permettre aux utilisateurs de MATHEMAGIX d'utiliser cette possibilité, mais aussi à court terme d'être utilisable par le groupe «interface avec des systèmes physiques». Cela n'utilisant pas, pour finir, les fonctions de MATHEMAGIX, l'implémentation de la FFT perdait déjà une certaine partie de son opportunité.

De plus, l'ensemble des membres du projet ne connaissait pas très bien les possibilités de parallélisation, et le groupe chargé de cette partie a passé une majeure partie du temps en bibliographie, ce qui ne laissait pas de place à l'écriture de code, dans le cadre du temps imparti pour ce projet.

3.3 Résultats

Le WP2 avait deux raisons d'être : faire de MATHEMAGIX un logiciel de calcul formel digne de ce nom et lui apporter des outils capables de le rendre attractif pour toutes sortes de public.

Le premier point n'a pas vraiment été atteint, faute de temps, par manque d'implication des membres du groupe et aussi, il faut le reconnaître, par excès d'ambition lors de la définition des objectifs.

Pour le second point, l'interface physique a montré qu'il était possible de faire de MATHEMAGIX plus qu'un simple outil de calcul : pour un expérimentateur, le calcul formel peut devenir plus qu'une calculatrice graphique évoluée.

Il faudrait maintenant un retour de la part d'utilisateurs pour savoir si ces innovations trouvent vraiment leur place dans le travail scientifique de tous les jours. Ceci ne pourra cependant se faire vraiment que lorsque MATHEMAGIX aura atteint des performances (au niveau vitesse mais surtout au niveau fonctionnalités) comparables à celles des logiciels de calcul formel existants sur le marché.

4 WP3 : Interface utilisateur.

Responsable : Alexandre Derouet-Jourdan
Membres : Emmanuel Lassalle, David Grosdemange.

4.1 Objectifs initiaux

Le but premier de ce workpackage était de fournir une interface graphique au projet, sous la forme d'un grapheur 2D, d'un grapheur 3D et d'une intégration de Mathemagix dans TeXmacs. En effet, à l'heure actuelle, Mathemagix ne dispose pas d'outil graphique. L'interface TeXmacs se présentait comme la meilleure solution pour fournir à Mathemagix l'interface graphique dont il a besoin. Toutefois, nous n'avons pas eu le temps et les moyens nécessaires à l'entière réalisation de cette partie du workpackage. De plus, pour compléter Mathemagix, nous avons ajouté des grapheurs de courbes et de surfaces, ceux-ci se présentant sous la forme de librairies externes de Mathemagix.

4.2 Le grapheur 2D

4.2.1 Description

Le grapheur 2D doit permettre le tracé de courbes en 2D. Le tracé est fait en utilisant les librairies OpenGL[3] (pour le rendu) et SDL[5] (pour la gestion de la fenêtre). Les fonctionnalités du grapheur 2D sont, en plus de tracer des courbes, d'exporter les résultats sous forme d'images (bmp, jpeg).

4.2.2 Réalisations

Nous avons réalisé un premier programme de tracé de courbe 2D. Celui-ci se base sur OpenGL, et utilise un algorithme élémentaire. L'interfacage avec Mathemagix est terminé et fonctionne, et permet l'utilisation de Mathemagix pour calculer les points de la courbe. En plus de la fonction de tracé, trois fonctions ont été implémentées, permettant de gérer la fenêtre d'affichage en fixant l'intervalle des abscisses, celui des ordonnées, ou en laissant le programme se charger lui même de l'intervalle des ordonnées. Une dernière fonction a été implémentée, permettant le tracé de deux courbes sur le même graphique. De plus, les résultats peuvent être exportés au format JPEG.

En outre, nous avons orienté notre travail sur la conception d'un second algorithme de tracé, plus efficace car calculant moins de points. En effet, lorsqu'il s'agit de tracer les courbes représentatives de fonctions telles que des sommes de séries ou des intégrales paramétrées, une définition de la courbe point par point s'avère coûteux en temps de calcul. Il faut donc trouver une méthode de tracé qui offre un rendu satisfaisant en calculant le moins de points possible. Un tracé segment par segment calcule des points selon un intervalle régulier et aboutit généralement à des courbes anguleuses aux endroits de grande variation de la pente. Au contraire, l'utilisation de courbes d'interpolation peut arrondir des endroits normalement anguleux (dans le cas de variation discontinue de la pente par exemple). Nous avons donc mis au point un algorithme qui trace la courbe par une suite de segments de taille variable. Il cherche à capter localement les variations d'une fonction et ajouter des points (réduire la taille des segments) dans les zones de grande variation de la pente pour supprimer l'effet anguleux de la courbe.

De plus, dans les zones à faibles variations (donc sans problème de courbe anguleuse), il va augmenter la taille des segments pour calculer moins de points.

L'algorithme qui optimise le nombre de points calculés opère de la manière suivante : on effectue un calcul grossier de la courbe segment par segment avec des intervalles assez grands (40-50 pixels). Ensuite, on détermine les endroits "trop anguleux" par un simple produit scalaire entre deux vecteurs (segments). L'angle correspondant est visible au-delà de 170° environs (valeur déterminée expérimentalement). Dans le cas d'un angle visible, on ajoute un point de chaque côté de l'angle, ce qui a pour effet de l'élargir (on ajoute des points uniquement lorsque la taille des nouveaux segments est supérieure à quatre ou cinq pixels, au-delà, les modifications ne sont plus visibles). L'affinage de la courbe se fait en un seul parcours. Finalement, on obtient une courbe sans angles visibles. Expérimentalement, comparé à une segmentation régulière serrée de la courbe (intervalles de quatre ou cinq pixels) on obtient un rendu équivalent avec jusqu'à trois à quatre fois moins de points calculés (suivant les parties régulières de la courbe).

4.2.3 Extensions possibles

Une extension possible serait la création d'une fonction permettant le tracé d'un nombre indéfini de courbes sur le même graphique, pour ne plus être limité à deux. De plus, la possibilité de fixer les abscisses et ordonnées de la fenêtre en les passant en paramètres de la fonction de tracé serait probablement plus intuitif et plus pratique pour l'utilisateur.

4.3 Le grapheur 3D

4.3.1 Description

Le grapheur 3D doit permettre le tracé de surfaces en 3D. Le tracé est fait en utilisant les librairies OpenGL et SDL comme pour le grapheur 2D. Les fonctionnalités du grapheur 3D sont, en plus de tracer des surfaces, la possibilité d'en tracer deux sur le même graphique en utilisant au besoin la transparence. De même que pour le grapheur 2D, on a la possibilité d'exporter les résultats sous forme d'images.

4.3.2 Réalisations

Le premier programme de tracé de surface que nous avons réalisé se base sur un algorithme élémentaire de calcul de surface par maillage régulier du plan.

Nous nous sommes penchés sur une possibilité d'extension de l'algorithme qui optimise le nombre de points calculés pour le grapheur 2D au grapheur 3D. Toutefois, la notion de précision de courbe n'est pas la même en 2D qu'en 3D et l'algorithme étendu calcule trop de points inutiles.

Le grapheur 3D contient ainsi, comme pour le grapheur 2D plusieurs fonctions. Nous trouvons deux fonctions de tracé, une traçant une seule surface, l'autre en traçant deux. Nous trouvons aussi les fonctions permettant la gestion de la fenêtre d'affichage.

Le rendu des courbes et des surfaces se fait à l'aide d'OpenGL qui est bien adapté à la gestion graphique (particulièrement en 3D). Les qualités d'OpenGL ne se ressentent pas beaucoup concernant le grapheur 2D puisqu'on ne fait appel qu'au tracé de segments. En revanche OpenGL a largement facilité la création du grapheur 3D, puisqu'il intègre les rotations, et la gestion des faces cachées. Il s'est aussi montré efficace pour la transparence en simplifiant la

technique employée : il s'agit de tracer les faces en désactivant le Z-buffer et en activant l'alpha blending puis de retracer les faces en réactivant le Z-buffer. On obtient ainsi une transparence assez nette.

4.3.3 Extensions possibles

Elles sont les mêmes que pour le grapheur 2D.

4.4 Autres Grapheurs

En plus du grapheurs 2D, qui ne permet de tracer que des fonctions du type $y = f(x)$, nous avons implémenté deux grapheurs 2D, qui permettent respectivement le tracé de fonctions paramétrées et le tracé de fonctions implicites en 2D. Leurs implémentations et leurs utilisations se rapprochent beaucoup de celles du grapheur 2D. Là aussi, des extensions sont possibles, en particulier l'implémentation de ces grapheurs en 3D.

4.5 TeXmacs

4.5.1 Description

Le but de cette partie était de travailler sur l'interface graphique de MATHEMAGIX qui aurait été en fait une interface entre TeXmacs et MATHEMAGIX. Nous nous sommes penchés sur le cas particulier de la sortie de MATHEMAGIX dans TEXMACS, l'élément clé de cette partie étant la possibilité de tronquer les résultats trop longs.

TeXmacs est une sorte d'éditeur de texte haut niveau, qui permet de lancer des sessions de différents CAS et d'afficher la sortie dans le style LaTeX. L'interactivité est rendu plus agréable à l'utilisateur grâce à toute une série de barre d'outils. Cependant, le fonctionnement reste assez basique : MATHEMAGIX est lancé avec l'option `-texmacs`, les sorties ne sont plus du texte simple mais un langage de script reconnu par TEXMACS, et inversement, des évennements tels le *double clic* sont converties par le plugin TEXMACS en une commande à envoyer à MATHEMAGIX.

4.5.2 Réalisations

Nous n'avons pas atteint tous nos objectifs. Nous avons intégré à MATHEMAGIX la troncature des résultats numériques. Son utilisation est assez laborieuse dans le sens où cet ajout complexifie la compilation de MATHEMAGIX. De plus, la troncature ne fonctionne que pour les entiers positifs. Toutefois, il est possible de fixer le nombre de chiffres affichés et de choisir si on tronque les résultats ou non.

4.6 Difficultés rencontrées

La principale difficulté rencontrée lors du travail sur la troncature a été de comprendre le code de MATHEMAGIX et localiser les lignes correspondant à l'affichage des résultats. C'est la raison pour laquelle cette partie du workpackage n'a pas pu se développer et atteindre ces objectifs. Concernant les grapheurs, la principale difficulté rencontrée a été l'écriture de glue pour interfacer les grapheurs avec MATHEMAGIX, l'évolution de MATHEMAGIX nous ayant contraints à l'écrire plusieurs fois, et attendre une bonne gestion des nombres flottants. Une autre difficulté rencontrée a été la transparence dans le grapheur 3D.

5 WP4 : Interface OCaml

Responsable : Chantal KELLER.

Membres : Olivier SCHWANDER, Pierre-Etienne MEUNIER, Lionel RIEG, David GROSDEMANGE.

5.1 Présentation

Le but de ce workpackage est de réaliser une interface entre MATHEMAGIX et des bibliothèques utilisateurs écrites dans d'autres langages.

Pour l'instant, MATHEMAGIX possède une interface avec `C++` mais pour assurer son succès et sa pérennité au sein de la communauté d'utilisateurs, MATHEMAGIX doit permettre d'interfacer le plus facilement possible des bibliothèques programmées dans des langages de programmation répandus et maîtrisés par leurs utilisateurs.

Vu le temps qui nous était imparti, un seul langage pouvait être traité et nous avons choisi `OCaml` : il s'agit en effet du langage avec lequel le plus de membres sont familiers (presque tous en fait) donc nous prévoyions au départ leur permettre d'écrire leurs bibilothèques en `OCaml` et non uniquement en `C++`.

5.2 Objectifs initiaux

Nous voulions donc réaliser une interface entre `OCaml` et MATHEMAGIX, afin que tout utilisateur puisse écrire, s'il le souhaite, une bibliothèque pour MATHEMAGIX en `OCaml`, et ce de manière (quasi) transparente.

Dans un premier temps, le workpackage s'est interrogé sur la meilleure façon de réaliser cette interface : interfacer `OCaml` directement avec MATHEMAGIX ou interfacer `OCaml` et `C++` afin d'utiliser la *glue* qui existe déjà ? Comment gérer la compatibilité entre le système de typage d'`OCaml` et celui de MATHEMAGIX ? ...

La décision finale a été d'interfacer `OCaml` avec `C++`, et d'utiliser ensuite le mécanisme existant de glue entre `C++` et MATHEMAGIX, notamment parce qu'il existe déjà une interface `C`/`OCaml` permettant d'utiliser avec efficacité les types simples de `OCaml` en `C` et qu'elle pouvait nous servir de base.

Cependant, notre méconnaissance de la *glue* entre `C++` et MATHEMAGIX ont rendu la génération automatique de *glue* impossible, et le travail du workpackage s'est un peu éloigné de MATHEMAGIX pour se concentrer sur l'interface `OCaml`/`C++`, et en particulier la gestion des types complexes. En dehors du projet initial, cela pourra être tout à fait diffusé et permettre aux utilisateur de `OCaml` et `C++` d'interfacer leur code très facilement.

La première partie de l'interface consiste à réaliser des fonctions de conversions pour les types `OCaml` plus évolués (des `array` ou des `list` habituels aux types sommes récursifs définis par l'utilisateur).

La seconde partie est de réaliser un analyseur automatique de module `OCaml` capable de générer tout le code permettant d'utiliser ce module au sein d'un programme `C++`. Cela consiste en fait à réaliser la conversion des arguments de `C++` à `OCaml`, puis d'appliquer la fonction `OCaml` voulue avant de convertir le résultat vers `C++`.

Le code généré *in fine* doit donc permettre d'appeler des fonctions `OCaml` par l'intermédiaire de fonctions `C++` en toute transparence pour l'utilisateur : ce dernier ne fait que remplacer

l'appel à la fonction OCaml sur un type de donnée OCaml par un appel à une fonction C++ sur le type de donnée C++ équivalent.

5.3 Architecture de l'interface

5.3.1 Entrée utilisateur

L'utilisateur écrit un module OCaml et décrit son interface dans un fichier .mli, qu'il doit fournir en paramètre du générateur de glue.

Le fichier spécifie le nom OCaml et les types des valeurs à convertir, bref il a la forme usuelle d'un .mli :

```
val hello : unit -> unit
val print_fibo : int -> unit
val fibo : int -> int
val toto : int -> char -> bool
val tata : int -> char list -> bool array
```

5.3.2 Sortie

Le générateur produit un fichier OCaml et deux fichiers C++ :

- un fichier .ml expliquant à OCaml les fonctions qui peuvent être appelées de l'extérieur (des *callbacks*)
- deux fichiers .hh et .cc qui décrivent les classes C++ utilisées pour les appels OCaml depuis C++

Dans l'exemple précédant du module Fibo, on obtient :

- callback_fibo.ml déclarant que hello, print_fibo, fibo, toto et tata sont externes
- CFibo.cc et CFibo.hh qui contiennent la définition d'une classe encapsulant les appels aux fonctions OCaml sus-nommées.

Le programme affiche ensuite des directives de compilation sur la sortie standard, dans une syntaxe utilisable directement dans un Makefile (il n'est pas possible de générer le Makefile automatiquement car on risquerait d'en écraser un autre mais une simple redirection de la sortie standard le permet).

Les fonctions utilisées convertissent les types OCaml vers les structures de données fournies par la STL de C++, de façon à rester aussi standard que possible mais pouvoir néanmoins utiliser la puissance des structures de donnés avancées. Par exemple, il est difficile de convertir un tableau C++ en tableau OCaml car la taille n'est pas connue par le tableau C++ et il serait très alambiqué de la passer en paramètre, notamment lorsqu'on manipule des listes de tableaux ou des structures encore plus complexes.

Cependant, il est très facile pour un utilisateur de modifier le générateur pour utiliser d'autres structures de données, à partir du moment où les fonctions de conversion sont fournies.

5.3.3 Utilisation du code généré

Il suffit à l'utilisateur d'instancier la classe générée pour utiliser de façon transparente les fonctions OCaml du module.

Reprenons notre exemple du module Fibo :

```
#include <iostream>
#include "CFibo.hh"

 int main(int argc, char ** argv)
 {
   CFibo essai(argv);

   std::cout << "*** hello : " << std::flush;
   essai.hello();
   std::cout << std::endl;

   std::cout << "*** print_fibo(10) : " << std::flush;
   essai.print_fibo(10);
   std::cout << std::endl;

   std::cout << "*** fibo(10) : " << essai.fibo(10) << std::endl;

   return 0;
 }
```

Les arguments `argv` passés au constructeur permettent de spécifier des options pour le toplevel `OCaml`. On peut très bien les laisser à `NULL`.

On peut remarquer l'utilisation de `std::flush`, qui est nécessaire pour permettre un affichage cohérent entre les deux langages. En effet, ils n'utilisent pas le même buffer d'écriture donc il est nécessaire de les vider manuellement sans quoi l'écriture attend le prochain passage à la ligne. D'où des problèmes lorqu'on souhaite un affichage sur la même ligne d'une chaîne venant de chaque monde.

5.4 Réalisations

5.4.1 mlglue.pl

Les modèles de code `C++` ont été développés à partir de la documentation `OCaml` pour l'interfaçage avec le langage `C` [2] .

Le langage choisi est `Perl`, un langage particulièrement adapté au traitement et à la génération de fichiers de texte : en effet, la simplicité d'utilisation des expressions régulières de `Perl` permet d'analyser facilement la syntaxe des fichiers d'interface `OCaml`. Qui plus est, la souplesse des structures de données `Perl` nous autorise à gérer dynamiquement des types définis dans les fichiers d'interface.

Avec la génération automatique de glue pour MATHEMAGIX (qui semble compromise au vu des difficultés rencontrées), la gestion dynamique des types constituera la dernière étape du développement de `mlglue.pl`

5.4.2 Conversion de types

Le document de référence décrivant l'organisation en mémoire des structures de données `OCaml` est *Développement d'applications avec Objective Caml* [7]. Nous nous sommes aussi appuyé sur la partie du manuel `OCaml` portant sur l'interface `OCaml`/`C` [2].

Nous avons écrit des classes permettant de convertir les types les plus importants de `OCaml` (types de base, `array`, `list`, `option` et surtout les types sommes).

Dans un soucis de cohérence et de facilité d'utilisation, nous avons regroupé toutes les formes de conversion (paramétrées ou non) sous un même modèle de classe :

```
class Convert_type {
public:
    value Cpp2Caml {type t};
    type Caml2Cpp {value v};
}
```

où `type` représente le type `C++` et `value` le type des représentations mémoires de `OCaml`.

Ce sont ces objets de type `value` qui sont passés à `OCaml` (et sont en fait des objets `OCaml` !). Dans le sens `C++` vers `OCaml`, le travail a donc consisté à partir d'un objet `C++` et à écrire proprement la représentation mémoire du type correspondant en `OCaml`. Dans l'autre sens, il fallait avant tout savoir parcourir les représentations mémoires de `OCaml` pour y lire les informations significatives puis construire la structure correspondante en `C++`.

Les templates de `C++` sont intensivement utilisés afin de simuler le polymorphisme de `OCaml`. Ils sont utilisés dès qu'il est besoin de connaître des informations sur le contenu de l'objet pour construire son type et pour convertir son contenu. Par exemple, construire le type d'un entier ne demande pas d'information alors que le type d'une liste ou d'un vecteur contient le type de ses éléments et que convertir une telle structure nécessite de savoir convertir ses éléments. Les détails de l'utilisation des templates sont discutés dans l'annexe A.

Nous comptions au départ ajouter encore la gestion des exceptions `OCaml` en `C++` : nous voulions pouvoir rattraper au retour dans `C++` une exception levée lors de l'appel `OCaml`. Après un bref examen de la manière dont il fallait procéder, nous nous sommes aperçu que cela allait être lourd, difficile et compliquerait grandement le code. Nous avons donc estimé que le jeu n'en valait pas la chandelle car les fonctions sont presque toujours totales. D'autant que si l'utilisateur veut absolument récupérer des exceptions `OCaml`, il peut le faire tout simplement en encapsulant le résultat dans un type somme dont les autres constructeurs correspondraient aux exceptions qu'il veut rattraper depuis `C++`.

5.5 Conclusion

À l'exception de l'interfaçage automatique, tous les objectifs du workpackage 4 ont été atteints.

Un résultat intéressant mais non prévu au départ a aussi été atteint : nous fournissons un système d'interfaçage entre `OCaml` et `C++` adapté au `C++` (notamment avec l'utilisation des classes et des templates) et beaucoup plus évolué que les fonctions de bases fournies par l'API `OCaml`.

Qui plus est le système de génération automatique de code permet aux développeurs d'utiliser de façon quasiment transparente un module `OCaml` au sein d'un logiciel écrit en `C++`.

Une restriction notable de notre système de conversion est l'absence de gestion des structures cycliques ou utilisant le partage. Mais comme celles-ci sont en général définies par l'utilisateur, c'est à lui de faire attention à ce qu'il n'y ait pas de bouclage ou de recopie inutile. En

effet, nous ne nous sommes proposés de permettre les conversions uniquement pour les types usuels déjà implementés, ce que nous avons fait.

Un projet d'informatique en L3 semblable au nôtre avait déjà essayé de développer une inteface `OCaml`/`C++`, sans succès. C'est pourquoi Lionel Rieg et Marc Lasson (ce dernier étant non membre du projet) projettent de soumettre l'interface `OCaml`/`C++` que nous avons réalisée à *la bosse de `OCaml`* [2] de façon à la diffuser et permettre qu'elle soit réutilisée le plus largement possible. Avant cela cependant, il est prévu de modifier le système de génération du code `C++` nécessaire aux appels de fonctions `OCaml` pour d'étendre les possibilités de notre interface. Plus précisemment, il s'agit d'écrire un petit langage de commande qui, inséré dans les commentaires des `.mli` de façon à rester compatible avec une compilation par `OCaml`, permettrait de paramétrer la génération de code.

Voici quelques exemples de paramétrisation possible :

- pouvoir changer les noms des fonctions ou en ignorer une
- pouvoir définir une fonction n'existant pas dans le module et qui serait la composée de deux fonctions du module, sans devoir modifier ce dernier dans le cas où il s'agit d'un module standard, ceci afin d'éviter des conversions intermédiaires inutiles.
- pouvoir spécifier au compilateur d'associer tel type `OCaml` à tel type `C++`, utile lorsqu'on utilise un type défini par l'utilisateur en `OCaml` mais qui possède un équivalent en `C++` déjà défini dans la `STL`.

[2] `http://caml.inria.fr//cgi-bin/hump.fr.cgi`

6 WP5 : Communication & Documentation

Responsable : Fabien Givors.

Membres : Fabien Benureau, Pascal Vanier, Fabien Givors, Olivier Schwander, Adrien Friggeri, Nicolas Estibals

6.1 Objectifs initiaux

Ce WP a pour but d'assurer la communication au sein des développeurs de l'équipe, de fournir une documentation aux développeurs extérieurs qui voudraient par la suite se greffer sur le projet, une documentation aux utilisateurs de HAMAM, mais aussi de garder le contact avec les partenaires intéressés.

La documentation développeur est effectivement nécessaire à tout projet de développement, et ce pour plusieurs raisons. Premièrement, il y a un objectif de formation : parmi les membres de l'équipe, tout le monde ne savait pas coder en `C++`, il fallait donc faire un tutoriel pour ce langage. Deuxièmement, il est nécessaire de rendre du code qui forme un tout homogène et lisible, de manière à ce que les développeurs futurs aient le moins possible de difficultés à se plonger dans les sources, ce qui est un des principes de base d'un logiciel *open source*. Un des objectifs de Mathemagix est d'être "grand public", c'est-à-dire utilisable dans le secondaire et dans les filières scientifiques non informatiques. Tout naturellement, HAMAM reprend pour lui ces objectifs et, pour ce faire, documente ses modules pour les utilisateurs. Le rôle de ce WP est d'homogénéiser les documentations fournies par les différents WP.

Les partenaires doivent être tenus au courant des évolutions de notre projet, et peuvent répondre aux questions liées au partenariat. Il est important qu'il y ait concertation tout au long du projet pour éviter les mauvaises surprises lors du rendu.

6.2 Réalisations

6.2.1 Formation aux développeurs : Généralités

Le travail de ce WP étant attendu par les développeurs pour commencer à coder, la pluspart des tâches de formation ont été réalisées les deux premières semaines. En particulier, des HOWTOs sur les outils utilisés par le projet (SVN, CVS) ont été écrits, ainsi qu'une aide à la compilation et à l'installation de Mathemagix. Un mini-cours de `C++` a également été écrit, de sorte que les néophytes puissent comprendre le code de leurs collègues et apprendre sur le tas plus facilement. Une liste des HOWTOs est donnée en annexe B.

6.2.2 Outils de communications internes et externes

Nous avons eu quelques difficultés techniques pour mettre en place un Wiki. Mais nous avons su réagir assez rapidement et maintenant, toute la documentation est présente sur celui-ci. Un Wiki est une aide précieuse pour les développeurs, car tout comme un forum, il permet de discuter de problèmes et d'autres et de donner une solution à ceux-ci, mais en plus de ça, il est fait pour... c'est à dire que les données sont structurées et facilement accessibles aux autres développeurs.

Une page d'accueil (bilingue) présentant le projet a été réalisée. Mais des problèmes de version de logiciels sur le serveur nous ont contraint à conserver un design assez sobre...

Un autre outil de communication a été mis en place et est désormais utilisé par le projet : IRC. Pour ce faire, une documentation a été réalisée.

6.2.3 Formation aux développeurs : MATHEMAGIX

Les plus grosses difficultés de ce groupe de travail ont été rencontéres lors de la réalisation de la documentation sur l'interfaçage avec MATHEMAGIX : la *glue*.

MATHEMAGIX a été conçu de façon modulaire, si c'est ce qui fait sa puissance, la contrepartie est la nécessité de l'interfaçage. La *glue* désigne l'ensemble du code liant le cœur de MATHEMAGIX aux modules utilisateurs : elle transforme les structures de données et les fonctions définies dans les librairies extérieures en *types* MATHEMAGIX et fonctions MATHEMAGIX sur ces types, accessibles depuis l'interpréteur.

L'auteur nous a fourni de la documentation sur l'ajout de modules. Malheureusement, elle s'est révélée insuffisante et incomplète, ne rentrant pas vraiment dans les détails, et n'expliquant pas clairement le rôle des différentes fonctions. Une autre barrière a été pour nous l'absence de commentaire dans le code, ce qui s'est traduit par une nécessité de comprendre *par le code* en recherchant les appels de la fonction ses surchanges possibles, et les bouts de code correspondants. Le fait que, à travers les différentes versions de MATHEMAGIX, la glue aie dû être refaite ne nous a pas vraiment facilité la tâche. De très longues séances de lecture et compréhension du code ont été nécessaires pour obtenir enfin des résultats. Finalement, nous avons pu fournir aux programmeurs une documentation par l'exemple assez propre et claire, et surtout une expérience qui leur a permis de gagner beaucoup de temps au débogage.

6.2.4 Uniformisation du code et de la documentation

Une relecture systématique du code avait été prévue, mais la réorientation du WP1 (qui a décidé de développer une librairie indépendante de MATHEMAGIX) et le retard du WP2 ont fait que cette relecture a été faite localement par les workpackages et non globalement par le WP5.

Pour les mêmes raisons, l'homogénéisation de la documentation utisateur n'a pas pu avoir lieu. Cependant, la documentation développeurs est maintenue grâce à Doxygen et est complétée par le Wiki (voir Annexe B).

6.3 État des lieux final

Mis à part le fait que nous n'ayons pas pu homogénéiser la documentation utilisateur, nous avons atteint nos objectifs de quelques mois de travail.

7 Conclusion

L'énorme demande actuelle en calcul scientifique, conjuguée à l'amélioration et l'évolution des processeurs, placent les CAS, ponts entre les deux mondes, sur le devant de la scène. Le monde des CAS progresse vite, et les besoins des utilisateurs étant de plus en plus poussés et spécifiques, beaucoup de nouveaux logiciels sont développés dans ce domaine. Les exigences sont de plus spécifiques : par exemple, l'équipe ARENAIRE[4] attend d'un CAS qu'il certifie les décimales de ses résultats.

L'idée de se lancer dans la réalisation d'un CAS nous est donc apparue comme intéressante et utile à la communauté scientifique. Cependant il s'agit d'un travail bien trop gros pour être réalisé pendant le laps de temps accordé pour le projet. Se greffer à un projet existant avait donc un intérêt à la fois pratique et pédagogique. Pratique, parce qu'il est certainement plus intéressant de rajouter des fonctionnalités novatrices que de recoder l'addition ! Pédagogique, car il est très formateur de devoir s'adapter à un projet existant.

La principale difficulté du projet a d'ailleurs été de s'approprier, au moins en partie, le code de M. Joris van der Hoeven. En effet, si comprendre le code d'autrui est un exercice ardu, et ce d'autant plus qu'il est long et compliqué, le maîtriser est encore bien plus difficile ! Il semblerait ici que MATHEMAGIX ait été déjà trop abouti par certains côtés, en matière de typage par exemple, pour que comprendre le code et se l'approprier soit réalisable pendant la durée impartie.

La gestion de l'avancement des différentes parties du projet n'a pas non plus été aisée. Une grande hétérogénéité caractérisait notre équipe, que ce soit au niveau des compétences spécifiques (le `C++` était mal connu du groupe en début de projet), que des motivations particulières de chacun.

Cependant le projet HAMAM a permis de rajouter un certain nombre de réalisations concrètes à MATHEMAGIX. En effet MATHEMAGIX est désormais un CAS doté d'un grapheur 2D, 3D et sonore intégré et pratique à utiliser dans TeXmacs. De même MATHEMAGIX a connu un enrichissement de sa documentation développeur. L'interfacage avec `OCaml` a abouti à un interfacage automatique avec `C++` et des documentations facilitant l'ajout de nouveau modules.

A WP4 : Annexe technique

Nous allons détailler ici un peu plus en détail l'utilisation des templates `C++` dans la conversion de types.

A.1 Types non paramétrés

Tout d'abord dans le cas de types non paramétrés, il n'est pas besoin de templates, la conversion peut se faire directement :

```
class Convert_type {
public:
    static value Cpp2Caml {type t};
    static type Caml2Cpp {const value v};
}
```

Les types non paramétrés que nous gérons sont :
- `int` et `long`
- les mêmes en version non signé
- `double`
- `char`
- `string`
- `bool`
- `void`
- les types sommes utilisateur non paramétrés

Enfin, comme la plupart des méthodes de conversion pour les type non paramétrés sont très simples, voire ne font qu'appeler des fonctions de l'interface `OCaml/C` de [2], elles sont explicitement inlinées.

A.2 Types paramétrés

À présent, voyons le cas des types paramétrés :

```
template<typename type,class conversion>
class Convert_TYPE {
public:
  static TYPE<type> Caml2Cpp(const value);
  static value Cpp2Caml(TYPE<type>);
};
```

où `type` est le type du contenu de la structure et `TYPE<type>` le type global de la structure, qui dépend du type du contenu.

Comme on peut le voir, les templates prennent deux paramètres : d'une part le type du contenu de la structure et d'autre part, la méthode qui permet de convertir ce contenu. Il n'est en effet pas possible d'avoir une « variable de type » appartenant à la classe de conversion et qui inclurait le premier paramètre dans le second. D'où la nécessité d'employer deux paramètres,

ce qui est assez fastidieux mais peu important car l'utilisateur n'est pas en contact avec cette syntaxe.

Comme aucune conversion de ces types évolués n'existait auparavent, le code de ces méthodes de conversion est non trivial. Prenons comme exemple celui des listes :

```
template<typename type,class conversion>
std::list<type> Convert_list<type,conversion>::Caml2Cpp(const value v) {

    conversion elts;
    value c = v;
    std::list<type> *l = new std::list<type>;
    typename std::list<type>::iterator it;

    while (Is_block(c)) {
        assert(Is_block(v) && Tag_val(c) == 0 && Wosize_val(c) == 2);
        l->push_back(elts.Caml2Cpp(Field(c,0)));
        c = Field(c,1);
    }
    return *l;
}
```

La boucle `while` et la mise à jour de `c` correspondent au parcours de la représentation mémoire de liste en `OCaml`. Au passage, on rempli la liste `C++` équivalente à l'aide du résultat de la conversion de l'élément de la liste `OCaml` (`Field(c,0)`).

```
template<typename type,class conversion>
value Convert_list<type,conversion>::Cpp2Caml(std::list<type> l) {

    typename std::list<type>::reverse_iterator it;
    conversion elts;
    value v;
    value l1 = 1;                       // [] for Caml
    value l2;

    for (it = l.rbegin(); it != l.rend(); it++) {
        v = elts.Cpp2Caml(*it);         // conversion of the element
        l2 = alloc_tuple(2);            // memory allocation for the list
        Field(l2,0) = v;                // the Caml value
        if (l1 == 1) Field(l2,1) = l1; // trick for the empty list
        else Field(l2,1) = l1;          // general case
        l1 = l2;
    }
    return l2;
}
```

On remarque à nouveau ici le parcours de la structure de liste mais elle fait partie du monde `C++` cette fois. À chaque étape, on convertit l'élément et on crée une cellule de liste que l'on rajoute à la liste déjà créée en positionnant la valeur et le pointeur vers la case suivante.

Il faut noter que cette utilisation des templates est récursive : si une structure paramétrée contient une autre structure paramétrée, la méthode de conversion des éléments de la première y fera elle-même appel.
Voici trois exemples de types de classes de conversion, par ordre croissant de complexité, le dernier étant vraiment extrême :

- pour une liste d'entier :
  `Convert_list<int, Convert_int>`
- pour un tableau de listes de chaînes de caractères :
  `Convert_array<std::list<string>, Convert_list<string, Convert_string> >`
- pour une liste de tableaux de listes de tableaux de références de chaînes de charactère :

```
Convert_list<
    std::vector<std::list<std::vector<char**> > >,
    Convert_vector<
        std::list<std::vector<char**> >,
        Convert_list<
            std::vector<char**>,
            Convert_vector<
                char**,
                Convert_ref<
                    char*,
                    Convert_string
                >
            >
        >
    >
>
```

Les types paramétrés que nous gérons sont :

- `array`
- `list`
- `option`
- `ref`
- les types sommes utilisateur paramétrés

B WP5 : Annexe documentaire

B.1 Le wiki

`http://graal.ens-lyon.fr/hamam/`

B.2 Le coin développeurs

`http://graal.ens-lyon.fr/hamam/mediawiki/index.php?title=HAMAM:Accueil`

HOWTO SVN et Mamadou

Une documentation rapide pour la prise en main de l'outil SVN (projet Hamam)
`http://graal.ens-lyon.fr/hamam/mediawiki/index.php?title=HOWTO_SVN_et_Mamadou`

HOWTO CVS et Mathemagix

Une documentation rapide pour la prise en main de l'outils CVS (projet Mathemagix)
`http://graal.ens-lyon.fr/hamam/mediawiki/index.php?title=HOWTO_CVS_et_Mathemagix`

HOWTO TeXmacs

Une simple reprise de la documentation officielle de TeXmacs
`http://graal.ens-lyon.fr/hamam/mediawiki/index.php?title=HOWTO_TeXmacs`

HOWTO Cpp

Un tutorial entièrement développé par le WP5 pour apprendre rapidement le `C++` [3]
`http://graal.ens-lyon.fr/hamam/mediawiki/index.php?title=HOWTO_Cpp`

HOWTO Glue

Une simple reprise de la documentation officielle.
`http://graal.ens-lyon.fr/hamam/mediawiki/index.php?title=HOWTO_Glue`

La glue par l'exemple

Guide simplifié pour l'ajout de glue à Mathemagix. Les fichiers sources ne sont pas sur le Wiki mais dans le SVN.
`http://graal.ens-lyon.fr/hamam/mediawiki/index.php?title=HOWTO_Glue_WP5`

HOWTO IRC

Guide rapide sur l'utilisation de l'IRC de l'école.
`http://graal.ens-lyon.fr/hamam/mediawiki/index.php?title=HOWTO_IRC`

[3] Une des références étant [10]

HOWTO Doxygen

Guide rapide sur la documentation de son code pour Doxygen.
`http://graal.ens-lyon.fr/hamam/mediawiki/index.php?title=HOWTO_Doxygen`

Références

[1] Boost graph library. `http://www.boost.org/libs/graph/`.

[2] Conversion caml/c. `http://caml.inria.fr/pub/docs/manual-ocaml/manual032.html`.

[3] Opengl. `http://www.opengl.org`.

[4] Projet arenaire. `http://www.ens-lyon.fr/LIP/Arenaire/`.

[5] Sdl. `http://www.libsdl.org/`.

[6] Tree.hh. `http://www.aei.mpg.de/~peekas/tree/`.

[7] B. Pagano E. Chailloux, P. Manoury. *Développement d'applications avec Obective Caml.* O'Reilly, 2000.

[8] Jürgen Gerhard Joachim von zur Gathen. *Modern Computer Algebra.* Cambridge University Press, second edition, 2003.

[9] D.E. Knuth. *The art of computer programming. 2. Seminumerical algorithms.* Addison-Wesley, 1969.

[10] Bjarne Stroustrup. *The C++ Programming Language, 3rd edition.* Addison-Wesley, 2000.