REPUBLIQUE DU SENEGAL

MEMOIRE DE FIN DE CYCLE

Pour l'obtention du :

DIPLOME D'INGENIEUR TECHNOLOGUE (DIT)

SUJET :

CONCEPTION D'UNE NOUVELLE VERSION DE YAO : Logiciel de modélisation et de génération de code pour l'assimilation de données variationnelles.

Lieu de stage : **LTI Laboratoire de Traitement de l'Information**

Période stage : **04/2009 – 10/2009**

Présenté et soutenu par	*Professeurs encadreurs*	*Maitres de stage*
MAYOMBO Alexis	*Dr. Alex Louis CORENTHIN*	*Dr. Awa NIANG*

Année universitaire : 2008 – 2009

REPUBLIQUE DU SENEGAL

MEMOIRE DE FIN DE CYCLE

Pour l'obtention du :

DIPLOME D'INGENIEUR TECHNOLOGUE (DIT)

SUJET :

CONCEPTION D'UNE NOUVELLE VERSION DE YAO : Logiciel de modélisation et de génération de code pour l'assimilation de données variationnelles.

Lieu de stage : **LTI Laboratoire de Traitement de l'Information**

Période stage : **04/2009 – 10/2009**

Présenté et soutenu par	*Professeurs encadreurs*	*Maitres de stage*
MAYOMBO Alexis	*Dr. Alex Louis CORENTHIN*	*Dr. Awa NIANG*

Année universitaire : 2008 – 2009

DEDICACES

Je dédie ce mémoire :

Je dédie ce mémoire :

*A ma chère mère et complice **NIANGUI Rosalie** qui a toujours été présente, **Maman** je t'adore.*

*A mon père et ami **MOMBO Victor** pour avoir toujours cru en moi, **Papa** je t'estime beaucoup*

*A ma grande sœur **MBOUTOU Jeanne** pour ses conseils et son soutien,*

*A ma grand-mère **MOUGHOULA Jeanne***

A toute ma famille : mes tantes, mes oncles, mes cousins et cousines,

A tous mes amis,

A tous les étudiants des promotions.

REMERCIEMENTS

J'adresse mes sincères remerciements à toute personne ayant contribué de près ou de loin à la réalisation de ce travail, notamment :

⇒ *Mon Père* ***MOMBO Victor****,*

⇒ ***Mme, Awa NIANG,*** *Maître de Conférences, LOCEAN, UPMC/Paris et LTI, ESP (UCAD)/Dakar*

⇒ ***Dr, Alex Louis CORENTHIN****, professeur au LTI, ESP(UCAD)/Dakar*

⇒ ***Dr. Samuel OUYA,*** *responsable de la formation, professeur au LIRT, ESP (UCAD)/Dakar*

⇒ ***Mr Ahmadou Bamba MBACKE****,* professeur au département génie informatique, pour toute sa disponibilité et ses conseils.

⇒ *Mr,* ***Luigi NARDI****, Etudiant en thèse, EDITE, CNAM, CDD LOCEAN*

⇒ *Tout le personnel du laboratoire LTI, ainsi que l'ensemble des étudiants chercheurs du LTI.*

⇒ *Tout le corps enseignant ainsi que l'ensemble du personnel de l'ESP et l'ESMT,*

Toutes les personnes qui de prés ou de loin, ont contribué à la réalisation de ce document.

AVANT-PROPOS

Dans le cadre de ma formation d'ingénieur en Téléinformatique à l'École Supérieure Multinationale de Télécommunication de Dakar en partenariat avec l'Ecole Supérieure Polytechnique de Dakar, j'ai effectué un projet de fin d'études au sein du Laboratoire de Traitement de l'Information (LTI). Le sujet porte sur la conception d'une nouvelle version de Yao : logiciel de modélisation et de génération de code pour l'assimilation de données variationnelles.

Le Laboratoire LTI est l'un des plus important de l'Université Cheick Anta Diop de Dakar (UCAD) et de la sous-région, notamment grâce à sa participation dans l'élaboration de nombreux projets de la sous région, le LTI est également en partenariat avec d'autres grands centres de recherches à travers le monde dont pour ce projet je citerais le laboratoire le LOCEAN en France, ainsi que le partenariat avec l'IRD qui finance le projet. Le LTI soutient et en cadre de nombreux étudiants et chercheurs grâce à son savoir faire mais aussi grâce à la compétence de ses chercheurs qui ont fait leurs preuves dans de nombreux domaines et organisent des séminaires à travers le monde.

Table des matières

Chapitre 3 : Conception de la plate-forme

Chapitre 4: La réalisation

Nous présentons ici certains sigles et abréviations que nous utiliserons dans le document.

ANTLR	**A**nother **C**ompiler **C**onstruction **T**ool Set
ASA	**A**rbre de **S**yntaxe **A**bstraite
CIM	**C**omputation **I**ndependant **M**odel
GCC	**GNU C**ompiler **C**ollection
IHM	**I**nterface **H**omme **M**achine
IRD	**I**nstitue de **R**echerche en **D**éveloppement
JavaCC	**J**ava **C**ompiler **C**ompiler
LTI	**L**aboratoire de **T**raitement de l'**I**nformation
MDA	**M**odel **D**riven **A**rchitecture
MVC	**M**odèle **V**ue **C**ontrôleur
PIM	**P**latform **I**ndependant **M**odel
PSM	**P**latform **S**pecific **M**odel
PM	**P**latform **M**odel
POO	**P**rogrammation **O**rientée **O**bjet
PUMA	**P**ure **M**anipulator

Table des figures

INTRODUCTION

YAO est un outil de développement informatique de modèles numériques en vue de faire de l'assimilation de données. Il est né du constat de la difficulté de coder le modèle adjoint. Une méthodologie de graphe modulaire est utilisée pour palier cette difficulté. Par ailleurs, Yao a été conçu de façon générique pour s'adapter le plus possible à une variété de modèles en particulier pour les schémas aux différences finies.

- YAO utilise des spécifications pour générer un code adéquat. Ces spécifications servent à :
- informer des trajectoire(s), espace(s) et dimension(s) de l'application

  - Déclarer les modules et leurs caractéristiques

  - Décrire le graphe modulaire et l'ordonnancement des calculs.

  L'un des buts recherché est aussi que la programmation des modules, qui code la physique du modèle soit l'essentiel point de concentration de l'utilisateur qui se trouve ainsi soulagé de taches informatiques les plus rébarbatives.

  YAO utilisera l'ensemble de ces éléments pour créer l'exécutable de l'application.

  Cet exécutable pourra ensuite être piloté par un fichier d'instructions dont les séquences servent par exemple à :

- Initialiser le modèle

  - Charger des observations ou se placer dans le cas d'une expérience jumelle

  - Vérifier la validité du code (test de la fonction objective, de l'adjoint, du tangent linéaire).

  - Lancer un run d'assimilation.

Ce présent mémoire se propose, non seulement, d'apporter une solution, fiable basée sur des solutions libres, pour la nouvelle architecture Yao. Pour mener à bien cette étude, nous ferons une présentation générale du sujet pour poser les bases de notre argumentation. Ainsi, nous présenterons dans le chapitre 1 intitulé « **Présentation générale** » le problème et ses objectifs.

Au chapitre 2 intitulé « **Choix de la méthode d'analyse et de Conception**», Nous détaillerons tout d'abord l'ensemble des outils et méthodes pour la réalisation de ce projet et ensuite nous explorerons les solutions existantes. Ceci permettra de mieux cerner et

comprendre la réelle nécessité de cette étude, et surtout de choisir parmi les solutions existantes celle qui semblera pertinente.

Aussi, passerons-nous au peigne fin la solution retenue au point d'en trouver des insuffisances vis-à-vis de nos objectifs, et nous proposerons des améliorations à cet effet.

Dans le chapitre 3, Une étude détaillée sur la conception sera faite, celle-ci nous permettra d'avoir une large vision sur l'architecture de l'application à développer et de comprendre son fonctionnement. D'où l'intitulé « **Conception** ».

Enfin, dans le chapitre 4, la « Réalisation », impliquera la mise en œuvre d'un environnement pour l'assimilation et la simulation des données. Nous présenterons quelques fenêtres de la nouvelle version de Yao qui portera désormais le nom de « Visual_Yao ». Peut-on automatiser les étapes d'installation et de configuration de notre application? Dans l'affirmative, quelle interface proposer à l'utilisateur pour faire de l'assimilation ?

CHAPITRE 1

Présentation générale

I. Le Laboratoire de Traitement de l'Information (LTI)

I.1. Présentation

Créé en janvier 2004 le LTI est devenu l'un des plus importants laboratoires de l'Ecole Supérieure Polytechnique, unité de recherche, transversale sur les Sciences de l'Ingénieur. Les activités du LTI couvrent un large spectre qui part de l'océanographie pour aboutir à des systèmes complexes de transport, en passant par tous les composants de la chaîne informatique : réseaux, systèmes répartis, calcul numérique et calcul formel, logiciels de la recherche d'information et d'aide à la décision, codage des langues nationales sur internet. Le LTI se veut centre africain de référence et d'excellence au cœur de la recherche et de ses applications. C'est dans cette optique qu'ont été créées des collaborations avec les grandes écoles de la sous région, unissant leur compétence à celles du LTI.

Le laboratoire est également largement impliqué dans les formations par la recherche, à travers le Mastère Recherche SPI.
Les chercheurs du LTI participent à de nombreux programmes de recherche nationaux ou internationaux (AUF, CRDI, IRD, etc...). Ils sont les acteurs majeurs de l'activité scientifique internationale par le biais de l'animation de comités éditoriaux de revues, de l'organisation de nombreuses conférences, l'accueil des chercheurs étrangers etc. enfin, le LTI entretient un tissu de relations industrielles et des collaborations académiques avec des universités ou des organismes de recherche comme l'IRD à travers des projets communs. Ainsi, JEAI, une équipe associée à l'IRD intitulée Modélisation et Traitement de Données Environnementales a été créée au LTI en 2007

I.2. Problématique et objectifs de recherche de la JEAI

Les sciences de l'environnement font face à des problèmes complexes tant du point de vue théorique (inter-actions de phénomènes complexes d'échelles différentes) qu'expérimental dû au nombre considérable de données recueillies par les satellites.

Le JEAI a pour objectif de développer à l'UCAD une expertise dans le domaine des méthodologies de traitement et de modélisation statistiques permettant une exploitation optimale de ces données.

La JEAI a pour vocation d'appréhender les impacts des modifications climatiques sur l'environnement dans la région du Sénégal et plus généralement dans le Sahel, ceci dans une problématique de développement durable. Les études s'organisent autour de trois axes de recherche dans le traitement des données d'environnement :

1 : Techniques Avancées en Mathématiques,

2 : Océanographie et impacts (Espaces Littoraux, Ecosystèmes,…),

3 : Météorologie et impacts (Pluviométrie et hydrologie).

I.3. Activités

Activités de formation à la recherche par la recherche

Les membres de la JEAI encadrent des étudiants du Master de Recherche de l'ESP. Cette année deux étudiants effectuent leur stage sur les deux thèmes ci-dessus. Des sujets de thèse ont été définis en cotutelle avec d'autres laboratoires.

II. Problématique

La modélisation numérique est de plus en plus utilisée comme complément à l'étude d'un phénomène physique se déroulant dans le temps, dans l'espace à une, deux ou trois dimensions (1D, 2D, 3D) ou bien tout à la fois dans le temps et l'espace.

Dans une première étape le modèle numérique permet de choisir et d'adapter les équations permettant de reproduire les lois physiques qui sous tendent le phénomène. Une fois vérifié le bon comportement de ces lois il est alors utilisé pour l'étude du phénomène lui même.

Cependant, les modèles ainsi créés restent toujours imparfaits. Le paramétrage des phénomènes, les discrétisations, l'incertitude sur les conditions initiales et les conditions aux limites introduisent des approximations dont il faudra tenir compte au moment d'étudier le comportement du phénomène ou de prédire les évolutions de celui-ci. De nouvelles techniques qui utilisent à la fois la modélisation et la résolution de problème inverse ont vu le jour, elles se regroupent sous le nom d'assimilation des données. Celles-ci tentent de faire évoluer le modèle physique ou les paramètres qui le contraignent (conditions initiales, ...) en introduisant un nouveau type d'information : les observations. Elles utilisent alors le modèle du phénomène ou modèle direct et se servent des observations pour contrôler les modifications nécessaires. Il existe deux grandes familles de technique pour faire de l'assimilation de données : les méthodes séquentielles et les méthodes variationnelles, les secondes étant plus adaptées si l'on veut prendre en compte d'une manière globale les variations du phénomène pendant une période de temps donnée. Toutes sont aujourd'hui de plus en plus utilisées car elles permettent de faire interagir l'ensemble des informations disponibles (connaissances théoriques sur le phénomène et observations de celui-ci) dans l'espace et dans le temps. L'efficacité de l'approche est visible si l'on considère leurs bonnes performances, on peut faire référence, parmi d'autres, à l'amélioration actuelle obtenue sur les prévisions météorologiques qui assimilent les observations satellitaires depuis une vingtaine d'années. La technique choisie par les centres de prévision météorologique est en général l'assimilation variationnelle dont le principe repose sur la minimisation d'une fonction de coût (objective). Il faut pour cela déterminer le gradient de la fonction par rapport aux variables à contrôler. Ce calcul nécessite la programmation du code adjoint du modèle direct (parfois aussi celui du linéaire tangent de ce même modèle pour certains algorithmes) qui permet de rétropropager le gradient de l'erreur sur les valeurs de sortie du modèle vers les variables d'entrée. Il faut donc décider d'investir dans la réalisation de ces développements, survient alors la difficulté de programmer ces codes (code adjoint, tangent linéaire) qui ne relèvent pas d'un sens physique commun, mais plutôt d'une démarche purement mathématique. Jusqu'à présent, deux approches sont utilisées pour créer ces codes adjoints à partir du code du modèle direct :

- Le code adjoint peut être généré à la main. Dans ce cas, il est souhaitable de choisir le nom des variables en suivant des règles précises qui facilitent la dérivation composée. Après cette première phase d'analyse et d'écriture qui est longue et délicate, le code doit être entièrement écrit.

- Le code adjoint peut être généré d'une manière automatique à partir du code de l'application directe. Cette approche répond à un besoin actuel qui consiste à exploiter de très gros code existant dont on est sur du bon fonctionnement. Le code adjoint généré de manière automatique est alors dépendant du langage de programmation dans lequel il est écrit. Il contient des instructions qui ne sont pas nécessaires et doit être nettoyée avant son utilisation. Cette opération doit d'autre part être recommencée si l'on fait évoluer le code direct et que l'on désire faire évoluer également son adjoint. Il est difficile de modifier directement le code adjoint, son écriture n'étant pas obligatoirement très lisible.

Une méthode nouvelle d'écriture de codes de modèles numériques a été développée au LOCEAN pour l'assimilation variationnelle de données. Elle est basée sur la représentation du modèle numérique du phénomène physique étudié sous forme de graphe modulaire. Ce formalisme a amené à la conception d'un outil général : Le logiciel YAO. En déchargeant l'utilisateur de l'implémentation des différents algorithmes nécessaires, YAO se propose d'être une aide à la réalisation d'inversion variationnelle et d'une manière plus générale à l'assimilation variationnelle. Il permet à l'utilisateur de réduire sa part de programmation et de ne se concentrer ainsi que sur la spécification de son problème, en particulier sur les dépendances physiques qui existent entre les différentes variables qui constituent le phénomène physique. Tout code écrit à partir de YAO bénéficie de façon quasi-automatique du code adjoint qui permet de calculer le gradient d'une fonction de coût. Un grand avantage de cette méthode est sa flexibilité et la facilité avec laquelle il est possible de modifier le modèle pour le faire évoluer. La méthodologie a été testée sur différents exemples (modèle Météo ISBA, modèle traceur OPA) ; elle a montré sa capacité à produire facilement des modèles adjoints et à permettre l'assimilation des données en 4D-Var (prise en compte des observations à n'importe quel moment durant le temps d'assimilation). Le champ d'utilisation de YAO n'est cependant pas limité aux seules simulations physiques.
Il peut être utilisé pour d'autres domaines d'application. YAO s'est donc avéré être une solution efficace pour surmonter plus aisément l'investissement nécessaire à la réalisation des modèles (direct, tangent et adjoint) dont il est reconnu qu'il s'agit habituellement d'un travail lourd et contraignant.

Ainsi, pour une vision plus large l'équipe de la JEAI du LTI de l'ESP en partenariat avec le LOCEAN et l'IRD, c'est proposé d'ajouter en plus des fonctionnalités existantes dans YAO d'améliorer de manière plus efficace la réalisation d'un projet dans YAO qui sera en

plus doté d'une interface graphique permettant à l'utilisateur non informaticien de pouvoir aisément réaliser des projets sous cette nouvelle version qui portera le nom de VISUAL_YAO.

III. Critique de l'existant

La solution actuelle est manuelle :

- La compilation et l'installation se font manuellement, en plus l'utilisateur doit parcourir plusieurs répertoires pour faire ses compilations.
- L'utilisation de la version actuelle demande une bonne connaissance en informatique notamment, l'utilisateur doit comprendre la programmation shell, chose qui nous le savons n'est pas facile.
- Lors de l'édition du fichier de description plusieurs erreurs d'inattentions peuvent se glisser rendant ainsi le projet d'assimilation pas fiable du tout.
- La gestion et la prise en main d'un projet dans l'actuel Yao, ne permettent pas d'optimiser en gain de temps.
- La compilation d'un projet ne se fait pas de manière aisée car il faut être dans le répertoire du projet pour pouvoir le compiler.
- La structure de Yao n'a pas été construite sur une logique de conception rigoureuse, pour d'autres développeurs il serait très difficile de comprendre la logique de Yao.

IV. Travail demandé

YAO est un logiciel d'implémentation informatique pour la simulation des modèles, il permet de générer du code C/C++ pour le modèle direct, l'inversion (adjoint) et l'assimilation de données.

Le travail consiste à concevoir et à développer une version graphique de YAO. En effet actuellement YAO est en mode texte (13 000 lignes de code) qui a été testé avec succès sur des problèmes de taille importante. Cependant cette utilisation demande une solide maîtrise de la programmation, ce qui est un frein à son appropriation par la majorité des chercheurs non informaticiens de formation, notamment ceux de la JEAI « Modélisation et Traitement de données Environnementales » MTE du LTI/ESP/UCAD. En effet l'un des thèmes de recherche de la JEAI concerne la détermination de la chlorophylle-a, dans les zones

océaniques au large du Sénégal, en utilisant un algorithme d'inversion neuro-variationnelle (neuro-varia) basé sur le logiciel YAO.

Autrement dit notre but est de concevoir et développer un logiciel pour faire des assimilations/simulations variationnelles adaptables aux conditions citées précédemment.

Cette version graphique de YAO, est en effet, un projet collaboratif entre le LTI et l'équipe MMSA, de LOCEAN créateur du YAO textuel ce qui faciliterait grandement ce travail et dont la nouvelle version portera le nom de Visual_Yao. D'autre part nous pensons que sa diffusion pourrait intéresser un grand nombre de scientifiques désirant se mettre à l'assimilation des données. **Ce travail aboutira à un produit diffusable, encore améliorable, mais final pour une première version de diffusion**.

V. Approche de solution

En tenant compte des critiques et des besoins d'automatiser et de rendre plus aisé les services cités ci-dessus la solution est de concevoir et de développer une application permettant de satisfaire au maximum possible l'utilisateur.

Dans l'état actuel l'application est lancée directement à partir de la ligne de commande et les résultats affichés dans la même fenêtre sous forme de trace peu lisible. L'interface graphique devrait permettre l'exécution et la visualisation des résultats de façon claire. Enfin, l'interface graphique faciliterait grandement l'utilisation de YAO et donc la mise en œuvre de l'assimilation de données pour les non informaticiens.

L'analyse d'erreur dans une application de grande dimension avec YAO se révèle très complexe (on ne sait pas où se trouve l'erreur et sur quel attribut il faut opérer). Cette complexité peut être largement réduite si on utilise VISUAL_YAO qui aura une interface graphique qui contrôlera la syntaxe à l'écriture, mettra immédiatement en évidence l'incohérence des attributs et ne permettra pas la continuation de la définition de l'application.

Une visualisation du graphe modulaire peut elle aussi s'avérer une aide importante pour la compréhension de l'application et doit en conséquence intéresser un grand nombre de scientifiques, utilisateurs potentiels de YAO. Le but est d'avoir un I.D.E.

(Interface Development Environment) pour le développement des applications. Cette interface graphique accompagnera l'utilisateur dès la création de l'application jusqu'aux tests et à la vérification des résultats.)

Pour cela l'application doit répondre aux besoins suivants :

- Avoir un logiciel performant.

- Avoir un logiciel qui respecte les principes des Interfaces Homme/Machine (IHM) tels que l'ergonomie et la fiabilité.
- Réduire les tâches manuelles qui permettraient de gagner en spatio-temporel.
- Obtenir de l'aide et d'autres informations supplémentaires au besoin.
- Avoir un logiciel évolutif et paramétrable.
- Permettre une gestion et un contrôle d'erreurs aussi performent que possible.
- Pour réaliser nos objectifs avec le maximum de rigueur possible et aux normes en actuelles, une démarche conceptuelle doit être réalisée. C'est ainsi, que nous allons présenter le modèle MDA.

V.1. Les Modèles MDA

V.1.1. Problématique

En effet, Yao depuis sa première version jusqu'à sa version 9, ne s'est pas doté d'une base conceptuelle solide, mais Yao s'est vu subir plusieurs évolutions aux fils des années. Cet aspect des choses fait que Yao est difficilement compréhensible par les développeurs, ainsi sommes-nous amener à trouver une méthode de conception répondant aux normes les plus universelles du moment. C'est ainsi, que pour ce faire nous avons utilisé le modèle MDA.

V.2.Présentation de la problématique Générale lié au MDA

• Croissance de la complexité des systèmes
– Volume de données, de code
– Hétérogénéité des langages et des paradigmes, des technologies …
→ Les outils ne répondent plus aux besoins. Que Faire?
Inventer de nouvelles solutions?

Réponse de l'OMG

L'OMG propose une nouvelle Vision: Le MDA (Model Driven Architecture)
• Changement de paradigme → objets aux modèles
• Changement de stratégie → interprétatif au transformationnel
• Changement de vision → centré code au centré aspect

Concepts

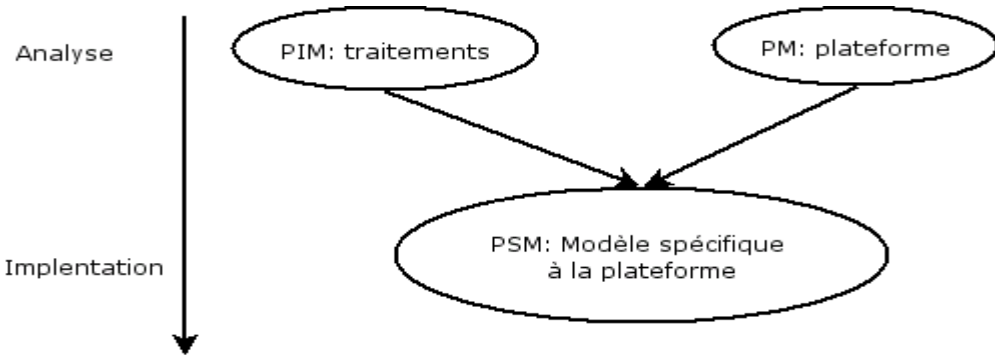


Figure I.a : schéma du concept

• Séparer les spécifications fonctionnelles des spécifications de son implémentation sur une plate-forme donnée.

• Permet l'interopérabilité des applications.

• Elaboration de modèles indépendants des plates-formes (PIM) + modèles dépendants des plates-formes (PSM).

• Techniques de modélisation + techniques de transformation.

Principe

Il s'agit de modéliser l'application que l'on veut créer de manière indépendante de l'implémentation cible (niveau matériel ou logiciel). Ceci permet une grande réutilisation des modèles.

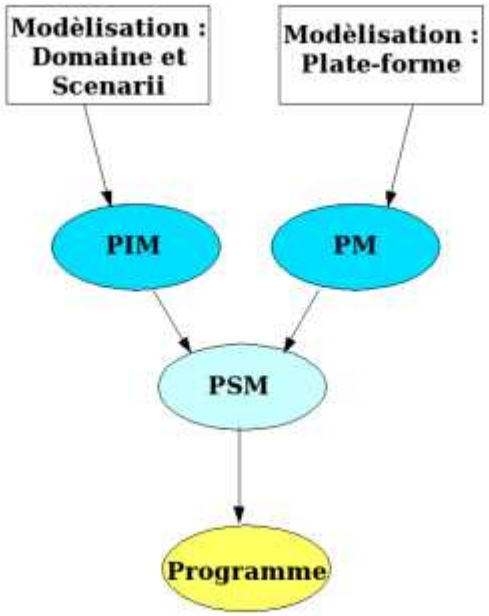


Figure I.b : Présentation réduite du Modèle

Les modèles ainsi créés (PIM - Platform Independant Model) sont associés à des modèles de plate-forme (PM - Platform Model), et transformés, pour obtenir un modèle d'application spécifique à la plate-forme (PSM - Platform Specific Model).
Des outils de génération automatique de code permettent ensuite de créer le programme directement à partir des modèles.
Cette approche permet de plus de faire évoluer facilement, à partir des modèles, les applications : le développement d'un nouveau module, quand tous les modèles nécessaires sont disponibles, peut ne pas prendre plus de quelques minutes.

V.3. Les Outils MDA

Pour obtenir une telle efficacité, plusieurs outils conceptuels sont mis à disposition. La technologie MDA (Model Driven Architecture) est supportée par l'OMG (Object Management Group), qui propose également UML (Unified Modeling Language) et Corba (Object Request Broker).
Ces outils sont :

- **UML**, largement utilisé par ailleurs, qui permet une mise en œuvre aisée de MDA en offrant un support connu,
- **XMI**, XML Metadata Interchange, qui propose un formalisme de structuration des documents XML de telle sorte qu'ils permettent de représenter des méta-données d'application de manière compatible,
- **MOF**, Meta Object Facility, spécification qui permet le stockage, l'accès, la manipulation, la modification, de méta-données,
- **CWM**, base de données pour méta-données.

L'OMG n'a pas jugé utile de standardiser un processus associé à ces outils. Leur rôle est de répondre aux besoins des utilisateurs de manière générique, et non de proposer de solutions définitives pour certains types d'applications précises.
Un processus de génie logiciel exploitant les possibilités de MDA a cependant été proposé : le Model-Driven Software Development.

V.4. La démarche à suivre dans le modèle MDA

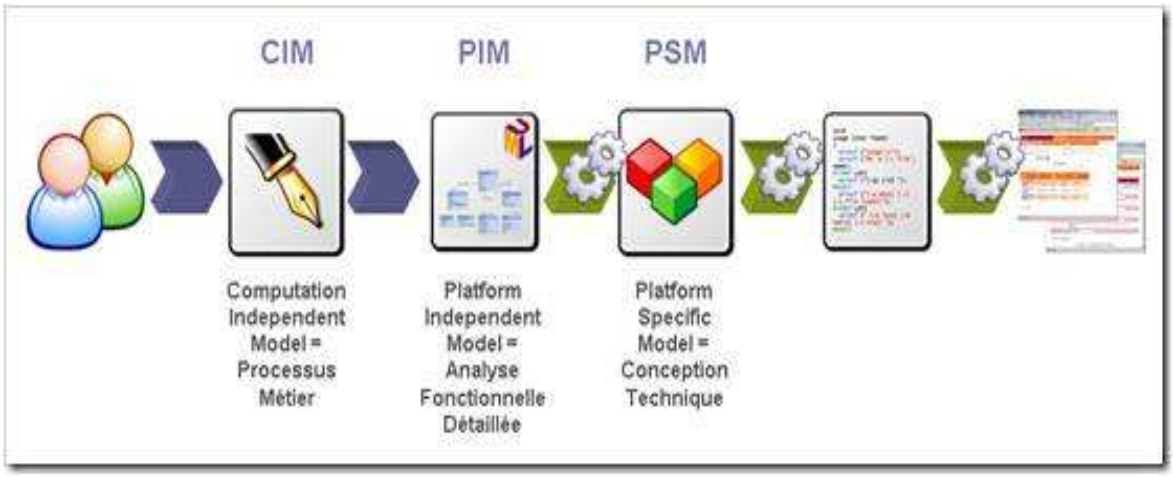


Figure I.c : processus de réalisation d'un modèle MDA.

Le processus métier indépendant de toute automatisation d'où est issue l'expression de besoin est décrite sous la forme d'un "CIM" (Computation Independant Model). L'analyse fonctionnelle détaillée, cœur du processus est concentrée dans le "PIM" (Platform Independent Model), qui, comme son nom l'indique, est strictement indépendant de l'architecture technique et du langage cible. Le "PSM" (Platform Specific Model) est le modèle de conception technique obtenu par transformation du PIM par projection sur l'architecture technique cible. C'est sur ce modèle que s'appuie la génération de code.

Je passe sous silence les MOF, QVT, OCL et autres ASL, mais, on le voit, MDA est un bon réservoir de TLA (Three Letters Acronyms) :

Pour résumer, les caractéristiques structurantes du MDA sont :

- L'énergie mise dans le projet (i.e. l'argent, les gens, l'intérêt, les contraintes...) est principalement affectée à la phase d'analyse fonctionnelle détaillée (et non à la phase de développement).
- La complexité fonctionnelle est "concentrée" dans les PIM, la complexité technique est concentrée dans l'outillage de génération/transformation, et dans le framework cible.
- Les PIM sont strictement fonctionnels et constituent le code source de l'application.
- Le cycle de développement est très court et l'approche par transformation/génération permet de confronter très souvent l'utilisateur avec l'application.

Le point le plus important dans ce qui vient d'être dit, c'est cette volonté de traiter explicitement, indépendamment et en profondeur, la complexité fonctionnelle. C'est ce qui fait vraiment la différence et l'intérêt de l'approche. Toutes les autres caractéristiques du MDA en sont les conséquences.

CHAPITRE 2

Choix de la méthode d'analyse et de Conception

I. Avantages de l'approche orientée objet :

Parmi les avantages de cette approche, on peut citer : la réutilisabilité des éléments (objets), l'avantage d'utiliser un objet de base afin de produire un autre qui peut être une amélioration de cet objet (phénomène d'héritage), etc.

L'objet est le cœur de cette approche. Tout objet donné possède deux caractéristiques :

· Son état courant (attributs).

· Son comportement (méthodes).

En approche orientée objet on utilise le concept de **classe,** celle-ci permet de regrouper des objets de même nature.

Une classe est un moule (prototype) qui permet de définir les attributs (champs) et les méthodes (comportement) à tous les objets de cette classe.

II. Model View Control (MVC)

Le modèle vue contrôleur est souvent décrit comme simple design pattern (motif de conception) mais c'est plus un architectural pattern (motif d'architecture) qui donne le ton à la forme générale d'une solution logiciel plutôt qu'à une partie restreinte.

Comme l'architecture 3-tiers il possède trois parties qui sont :

· *Model* : le modèle défini les données de l'application et les méthodes d'accès. Tous les traitements sont effectués dans cette couche.

· *View* : la vue prend les informations en provenance du modèle et les présente à l'utilisateur.

· *Controller* : le contrôleur répond aux événements de l'utilisateur et commande les actions sur le modèle. Cela peut entrainer une mise à jour de la vue.

III. Choix d'une méthode d'analyse et de conception

III.1. Conception globale (architecturale)

Cette conception consiste à scinder les tâches de l'application en différentes petites parties afin de mieux organiser et développer le logiciel. Ça se base sur la technique « **Diviser pour mieux régner ».**

Les retombés directs de cette technique ne sont pas négligeables, on peut mentionner quelques uns :

- Le développement de l'application peut être partagé par plusieurs groupes de travail.
- La possibilité de réutiliser les composantes dans d'autres applications.
- La portabilité de l'application.

Dans notre cas, on va utiliser entre autre le **MVC** (le Modèle Vue Contrôleur). On va essayer de scinder cette dernière en trois partie une partie de présentation (représentée par les interfaces), une partie qui permet l'accès au modèle de Yao et une dernière partie composée par le contrôleur.

Mieux encore le partage de l'application en sous systèmes, dans le but permettre de faire une conception détaillée de chaque partie.

III.2. Choix du principe et du logiciel de modélisation

Merise et UML sont deux grands principes de « traduction » ou modélisation d'un système d'information. Néanmoins, ils ne sont pas aussi proches qu'on pourrait le penser.

Le choix de l'un ou de l'autre se fait selon trois axes à savoir l'accessibilité, la précision et l'exploitabilité.

Pour le premier axe (accessibilité) MERISE présente l'intérêt d'avoir des modèles logiques moins détaillés facilement compréhensibles par un utilisateur moins avisé.

Tandis qu'UML conçu pour s'adapter à n'importe quel langage de programmation orientée objet (POO), présente plusieurs modèles (diagrammes) dont leurs compréhensions nécessitent une grande attention.

En ce qui concerne le deuxième critère (précision), MERISE est décevant. Malgré sa clarté, il lui manque une précision du fait qu'elle est éloignée du langage donc difficile à implémenter

alors qu'UML intègre les éléments communs des différents langages, sa volonté est d'être fidèle à la réalisation finale. Elle est beaucoup plus complète avec ses différents diagrammes.
Pour en finir avec l'exploitabilité, MERISE est une méthode plus généraliste. Elle donne une vue globale de la solution sans autant rentrer dans les petits détails. Contrairement à UML qui est conçu pour l'implémentation objet avec ses différents détails et sa portabilité (s'adapte à n'importe quelle plateforme) elle est donc plus exploitable.
L'une ou l'autre présente des avantages et des inconvénients. Il est réservé au concepteur de choisir la méthode la mieux adaptée pour son cas. Si on cherche la précision et l'exploitabilité comme dans notre cas UML devance de loin MERISE. Tandis que, si c'est la clarté et l'accessibilité qui sont en question MERISE est préférable.

La conception de notre application mérite bien une grande précision et une exploitabilité maximale. C'est la raison pour laquelle on va retenir UML. Les différences entres les logiciels de modélisation UML sont infimes. N'empêche de mentionner quelques logiciels qui sont à notre connaissance : Agro UML (open source), Poseidon UML, PowerAMC, Rational Rose et Bouml.
La facilité dotée aux deux derniers (Rational Rose et Bouml) de pouvoir faire une « **ingénierie** », une « **re-ingénierie** » et une « **reverse ingénierie** » a influée sur notre choix. Cependant, il fallait choisir entre Rational Rose et Bouml; étant donné que ce projet ce veut libre, et que les initiateurs du projet ne veulent pas des outils propriétaires Bouml a finalement été préféré grâce à son appartenance au monde de l'Open source, mais aussi grâce à sa portabilité, en effet ; Bouml permet d'utiliser des diagrammes fait sous linux que sous Windows et vis versa.

III.3. Choix des outils de développement

Un parmi les avantages qui nous ont permis de choisir UML comme méthode de modélisation est l'orienté objet. Cette approche influe aussi sur le choix du langage à adopter on peut rajouter quelques uns à savoir la portabilité, la facilité, la multidisciplinarité et pas mal d'autres comme la sécurité.

III.3.1. Choix du langage de programmation :

Le schéma suivant nous fait un bref aperçu concernant quelques langages. Il montre le domaine principal d'application, de l'année de l'essor du langage ainsi que l'interdépendance entre les différents langages.

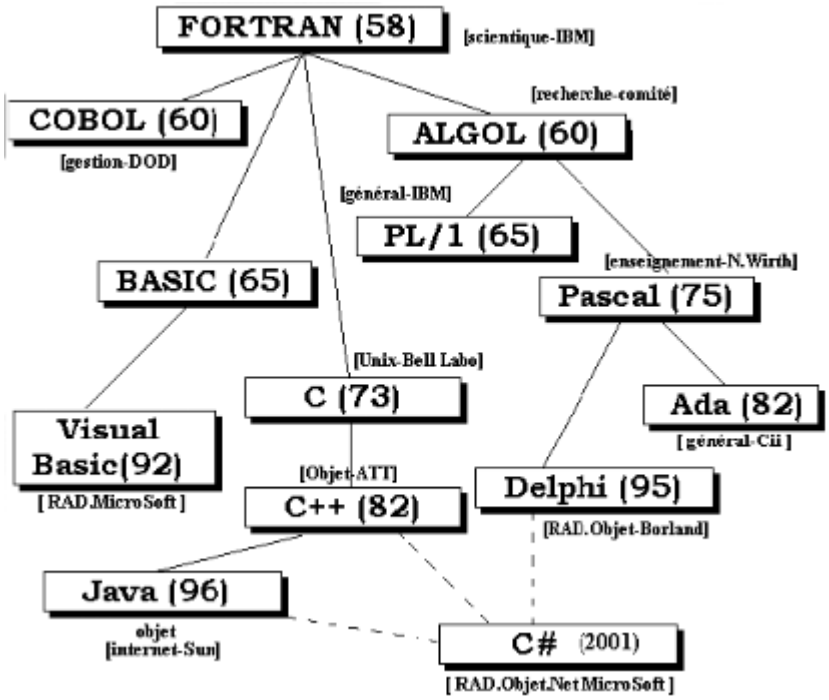


Figure II.a. *Relation entre les langages*

Comme on l'avait dit, le schéma ci-dessus nous donne une vue globale de l'évolution des langages. La plupart des langages présents dans ce schéma sont développés par des sociétés privées et sont donc destinés un large public, ils subissent alors la loi du marché (des hauts et des bas).

Souvent la sortie d'un nouveau langage n'est pas un fruit du hasard mais il s'appuie sur les anciens en profitant de leurs qualités et en essayant de remédier les défauts.

Ici on va essayer de faire une étude comparative sur les langages de programmation orientés objets qui sont en vogue sur le marché et essayer d'en sélectionner un qui répondra au mieux aux besoins d'implémentation de notre application.

On va s'intéresser surtout sur les langages Java, C++ et Visual Basic.

- **Java :**

Java est pourvu d'une grande sécurité, la richesse de ses bibliothèques, son adaptation à plusieurs plateformes, la qualité présentée par ses composantes graphiques (Swing) qui

suivent le modèle MVC, sa facilité de déploiement en réseau (RMI) et le fait qu'on peut avoir plusieurs « Look And Feel », en font de lui un langage redoutable puissant et performant. Une grande partie de sa syntaxe est empruntée de C et C++. La lenteur de sa machine virtuelle (JVM) constitue son principal défaut.

- **Visual Basic (VB) :**

VB est un langage issu du langage Basic, il a un environnement de développement intégré (EDI) qui permet de développer facilement des interfaces graphiques.
En appliquant quelques propriétés à ces dernières et en écrivant quelques petits bouts de codes on obtient des résultats satisfaisants.
VB est facilement accessible et assimilable

- **C++ :**

Le fait que notre projet ne soit pas orienté web, en C++ le code est natif ; on n'a pas besoin d'installer une runtime, mais aussi que le C++ est relativement proche de la machine et permet donc de nombreuses optimisations de bas niveau ; on peut avoir une gestion manuelle de la mémoire. En plus le C++ est à 95% compatible avec le C et à 100% compatible avec les librairies du C. Or, une partie de Yao se trouve malheureusement encore en C.

Les comparaisons faites ci-dessus nous permettent de choisir C++ comme langage d'implémentation de notre application. Avec ses multiples avantages en comparaison avec VB et java qui peut servir un début pour l'apprentissage des Langages Orientés Objets (LOO), C++ s'impose. Le fait que le projet soit en partie fait en C++ et l'autre partie en C a guidé notre choix car le principal souci était de continuer dans un langage compréhensible par toute la communauté des développeurs du projet et de continuer à associer le C++ et le C car l'ancienne version contient toujours une partie implémentée en C.

III.3.2. Choix de l'outil de développement (bibliothèque GUI) :

Vu la multidisciplinarité et sa maturité, plusieurs outils de développement de C++ ne cessent de voir le jour. On peut rencontrer pas mal de librairie de développement. Certains sont en open source et d'autres commerciaux.
Citons quelques uns :

wxWidgets, MFC, fltk, Gtk, wrapper C++, Qt etc. Ici nous présenterons uniquement les bibliothèques multiplates-formes.

Les bibliothèques multiplates-formes

Les avantages d'utiliser une bibliothèque multiplate-forme sont nombreux. Même si vous voulez créer des programmes pour Windows et que vous n'en avez rien à faire de Linux et Mac OS.

- Tout d'abord, elles simplifient grandement la création d'une fenêtre. Il faut beaucoup moins de lignes de code pour ouvrir une "simple" fenêtre.
- Ensuite, elles uniformisent le tout, elles forment un ensemble cohérent qui fait qu'il est facile de s'y retrouver. Les noms des fonctions et des classes sont choisis de manière logique, de manière à vous aider autant que possible.
- Enfin, elles font abstraction du système d'exploitation mais aussi de la version du système. Cela veut dire que si demain l'API Win32 cesse d'être utilisable sous Windows, votre application continuera à fonctionner car la bibliothèque multi-plateforme s'adaptera aux changements.

Bref, choisir une bibliothèque multi-plateforme, ce n'est pas seulement pour que le programme marche partout, mais aussi pour être sûr qu'il marchera tout le temps et pour avoir un certain confort en programmant.
Voici quelques-unes des principales bibliothèques multiplates-formes à connaître, au moins de nom :

- **.NET** (prononcez "Dot Net") : c'est en quelque sorte le successeur de l'API Win32. On l'utilise souvent en langage C#, un langage créé par Microsoft qui ressemble à Java (il ressemble plus à Java qu'au C++). On peut néanmoins utiliser .NET dans une multitude d'autres langages dont le C++.
  .NET est portable car Microsoft a expliqué son fonctionnement. Ainsi, on peut utiliser un programme écrit en .NET sous Linux avec Mono. Pour le moment néanmoins, .NET est principalement utilisé sous Windows.
- **GTK+** : une des plus importantes bibliothèques utilisées sous Linux. Elle est portable, c'est-à-dire utilisable sous Linux, Mac OS et Windows. GTK+ est utilisable en C.

Néanmoins, il existe une version C++ appelée GTKmm (on parle de wrapper, ou encore de surcouche).
GTK+ est la bibliothèque de prédilection pour ceux qui écrivent des applications pour Gnome sous Linux, mais elle fonctionne aussi sous KDE.
C'est la bibliothèque utilisée par Firefox par exemple, pour ne citer que lui.

- **Qt** : Sachez néanmoins que Qt est très utilisée sous Linux aussi, en particulier sous l'environnement de bureau KDE. Qt est une bibliothèque graphique portable, disponible pour Unix/Linux comme pour Windows. Elle est renommée pour avoir une architecture confortable. Qt est prévu pour être utilisé en C++ mais on peut aussi l'utiliser avec de nombreux autres langages (C/Python etc...).

Qt Development Frameworks (anciennement Trolltech) crée et commercialise Qt, framework d'applications multiplate-forme leader sur le marché. Nokia a fait l'acquisition de Trolltech en juin 2008 et rebaptisée Qt Development Frameworks. Grande facilité et la documentation de Qt est bien fournie c'est une librairie qui est en perpétuel mise à jour. Grace à la société Nokia ; dont les mérites ne sont plus à montrer, Qt possède beaucoup plus de crédibilité que ces concurrents. En effet, plusieurs applications embarquées des téléphones portables sont régulièrement développées sous Qt. Car nous savons en plus la forte croissance que connais le monde du mobile ces dernières années.

- **wxWidgets** : une bibliothèque objet très complète elle aussi, comparable en gros à Qt. Néanmoins, Qt est plus facile à prendre en main au début. Sachez qu'une fois qu'on l'a prise en main, wxWidgets n'est pas beaucoup plus compliquée que Qt. wxWidgets est la bibliothèque utilisée pour réaliser le GUI de l'IDE Code::Blocks. Pour la présenter, cette bibliothèque wxWidgets fournie un grand nombre d'API (Application Program Interface), utilisées pour le développement d'applications supportant des interfaces Homme-Machine multi plates-formes comme Linux, Windows ou Mac.
  Cette bibliothèque n'est pas nouvelle, elle a déjà bien plus de 10 ans d'existence et a considérablement évoluée depuis, d'ailleurs elle s'appelait auparavant xwindows, mais sous la pression de Microsoft à cause du "Windows" dedans, fut rebaptisée wxWidgets. wxWidgets est une bibliothèque multiplateformes, on écrit un seul code pour un OS et l'utiliser sous d'autres OS exemple le programmeur écrit une seule version de son code et peut normalement le compiler sous toutes les autres plateformes Linux, Windows ou Mac supportées par wxWidgets, via un compilateur

C++, et son application fonctionnera de la même façon sur chacune des plateformes, avec la même apparence mais avec le thème graphique de l'OS cible. Ce n'est pas un langage interprété cela garantie une certaine rapidité à l'exécution : en effet, à la compilation, les fonctions de la librairie wxWidgets utilisées par l'application ont été traduites en fonctions optimisées de l'OS, elles ne sont pas "traduites" à l'exécution comme des langages tel python. En plus, cette librairie est écrite en C++ et très orientée programmation objet et le fait de devoir compiler l'application avec de très nombreuses API et d'avoir ainsi la possibilité de rester un peu plus prêt du matériel de la machine que l'on utilise, ne désoriente pas le développeur C embarquée. Les défauts de cette librairie sont que l'installation n'est pas aisé notamment sous notre OS qui est mandriva 2008, en plus, elle ne possède pas son IDE et la documentation n'est pas si fournie que ça.

- **FLTK** : contrairement à toutes les bibliothèques "poids lourd" précédentes, FLTK se veut légère. C'est une petite bibliothèque dédiée uniquement à la création d'interfaces graphiques multi-plateforme.

Fltk a plusieurs avantages :

- Elle est orientée objet. Ses fonctions peuvent être facilement retenues, car la majorité des objets héritent de la même classe.
- Cela fait plusieurs années qu'elle est développée, et elle est toujours active (elle a été créé bien avant Qt et Gtk par exemple !).
- Elle est portable, votre programme multi-plateforme et son interface peuvent fonctionner sous GNU/Linux, Microsoft Windows ou Mac OS X (dans la majorité des cas, sans changer la moindre ligne de code !).
- À la différence d'autres bibliothèques, telles que QT ou Gtk, Fltk est très légère. Elle peut même être compilée statiquement avec votre programme sans se faire sentir (elle ajoute environ de 300Ko à votre exécutable ! Assez négligeable). La compiler statiquement a un deuxième avantage : éliminer les runtimes qui vous encombrent (les fichiers *.dll pour Microsoft Windows ou *.so pour GNU/Linux) pour avoir un fichier exécutable qui fonctionne sans imposer l'installation préalable d'une bibliothèque tierce.
- Elle supporte OpenGL. Vous pourrez intégrer une vue 3D dans vos interfaces !

- Le seul désavantage c'est que vous ne trouverez pas tout en standard (légèreté oblige), mais il y a l'essentiel pour faire des interfaces riches. Heureusement, quand une fonction disponible dans d'autres bibliothèques vous manque, vous avez la possibilité de télécharger des bibliothèques complémentaires à Fltk (par exemple, pour gérer l'impression, les effets spéciaux sur les widgets, un navigateur de fichiers plus complet, etc).

Comme vous pouvez le constater, le choix fait parmi toutes ces bibliothèques n'a pas été aussi facile car ce choix devrait être déterminant pour la suite de notre projet.

Le choix c'est finalement penché sur Qt, il est à noter que nous avons fait des compilations et des tests plus approfondis aussi bien avec Qt qu'avec wxWidgets, mais Qt a finalement gagné car elle est facile à prendre en main. C'est donc une bibliothèque plus « pédagogique » en quelque sorte.

Présentation de Qt

Qt est une **bibliothèque** multi-plateforme pour créer des GUI (programme sous forme de fenêtre).

Qt est écrite en C++ et est faite pour être utilisée à la base en C++, mais il est aujourd'hui possible de l'utiliser dans d'autres langages comme Java, Python, etc.

Plus fort qu'une bibliothèque : un framework

Qt est en fait... bien plus qu'une bibliothèque. C'est un ensemble de bibliothèques. Le tout est tellement énorme qu'on parle d'ailleurs plutôt de **framework** : cela signifie que vous avez à votre disposition un ensemble d'outils pour développer vos programmes plus efficacement. Qu'on ne s'y trompe pas : Qt est à la base faite pour créer des fenêtres, c'est en quelque sorte sa fonction centrale. Mais ce serait dommage de limiter Qt à ça.

Qt est donc constituée d'un ensemble de bibliothèques, appelées "modules". On peut y trouver entre autres ces fonctionnalités :

- **Module GUI** : c'est toute la partie création de fenêtres. Nous nous concentrerons surtout sur le module GUI dans ce cours.
- **Module OpenGL** : Qt peut ouvrir une fenêtre contenant de la 3D gérée par OpenGL.

- **Module de dessin** : pour tous ceux qui voudraient dessiner dans leur fenêtre (en 2D), le module de dessin est très complet !
- **Module réseau** : Qt fournit une batterie d'outils pour accéder au réseau, que ce soit pour créer un logiciel de Chat, un client FTP, un client Bittorent, un lecteur de flux RSS...
- **Module SVG** : possibilité de créer des images et animations vectorielles, à la manière de Flash.
- **Module de script** : Qt supporte le Javascript (ou ECMAScript), que vous pouvez réutiliser dans vos applications pour ajouter des fonctionnalités, sous forme de plugins par exemple.
- **Module XML** : pour ceux qui connaissent le XML, c'est un moyen très pratique d'échanger des données avec des fichiers formés à l'aide de balises, un peu comme le XHTML.
- **Module SQL** : permet un accès aux bases de données (MySQL, Oracle, PostgreSQL...).

Sachez d'ailleurs que le choix de Qt s'est fait en grande partie parce que sa documentation est très bien faite et facile à utiliser.

Qt est multiplateforme

Qt est un **framework multiplateforme**, voici un schéma qui illustre le fonctionnement de Qt :

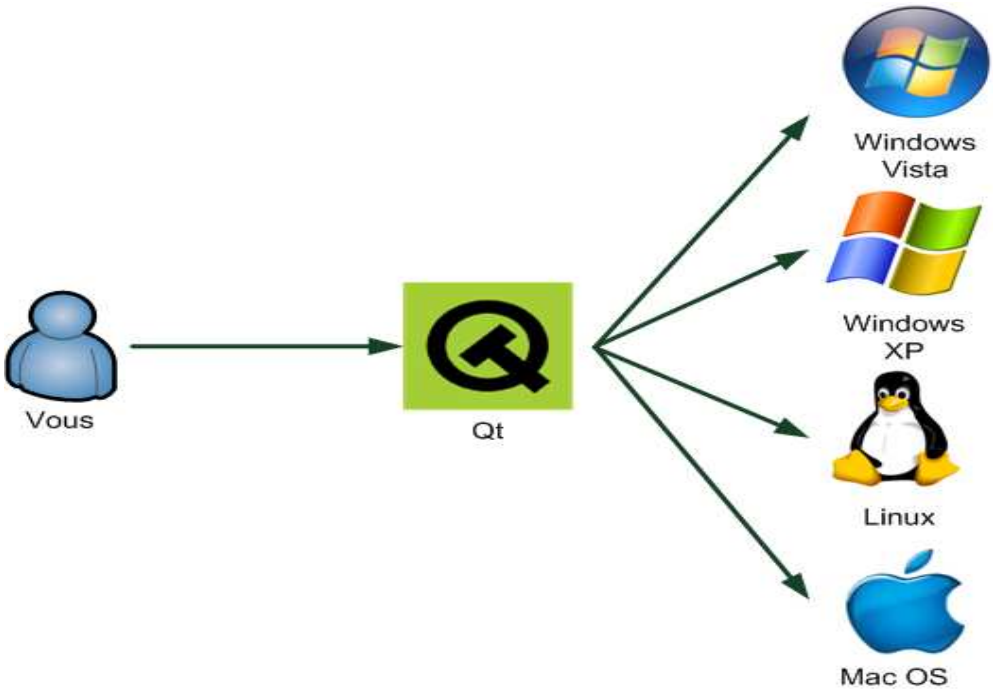


Figure II.b : représentation de Qt en multiplateformes

Oui, ça vous permet de créer des programmes binaires adaptés à chaque OS qui tournent à pleine vitesse.

Pour information, d'autres langages de programmation comme Java et Python ne nécessitent pas de recompilation car le terme "compilation" n'existe pas vraiment sous ces langages. Cela fait que les programmes sont un peu plus lents, mais ils s'adaptent automatiquement partout. L'avantage du C++ par rapport à ces langages est donc sa rapidité (bien que la différence se sente de moins en moins, sauf dans les jeux vidéo qui ont besoin de rapidité et qui sont donc majoritairement codés en C++).

IV. Etude de l'existant

L'étude de l'existant est la première phase du processus unifié. Elle a pour but ultime la clarification du champ de notre investigation.

Tout au long de cette phase, nous schématiserons l'expression préliminaire des besoins et nous présenterons une modélisation par des cas d'utilisation de fonctionnalités préliminaires de notre application.

Les activités principales qui vont se dérouler au cours de cette phase seront essentiellement la capture des besoins et l'analyse.

IV.1. Expression des besoins

Une plateforme de génération de code direct, de code adjoint et de code tangent linéaire est un logiciel qui assiste la conduite d'un projet d'assimilation variationnelle des données.

IV.1.1. Description du fonctionnement du logiciel

Compte tenu de la difficulté pour les non informaticiens de formation à pouvoir, installer, effectuer convenablement et efficacement les assimilations dans le logiciel Yao. La conception et le développement d'une interface graphique avec des outils de développement et de conception Open Source et adaptés à la plateforme de développement, permettra de générer au mieux le code adéquat pour la réalisation d'un projet d'assimilation, d'inversion de données variationnelle.

L'objectif de ce projet est de mettre en place une interface graphique aisée pour l'utilisateur et de faire certains contrôle pour interagir avec la grammaire de Yao, ainsi que de faciliter l'installation et la compilation d'un projet et de l'application elle-même, ce qui facilitera le travail de bon nombre d'utilisateur et de pouvoir élargir le nombre d'utilisateur du logiciel d'une part, et d'autre part, qui réduira de systématique le nombre d'erreurs du aux mauvaises manipulations de données, grâce à un premier contrôle en amont.

Le logiciel a pour tâches principales :

Structuration d'un projet sous Yao :

Les parties standards et spécifiques sont bien distinguées et séparées ce qui facilite la compréhension et la maintenance. Globalement, un projet se compose :

♠ Pour la partie spécifique :

- de la description du modèle (évoquée ci-dessus).

- du ou des sources chapeau : Intégrant des fonctions obligatoires et des données et/ou fonctions spécifiques et globales

- des sources des modules : Programmation des fonctions du schéma direct et de leur dérivée.

♠ Pour la partie standard :

- du code propre à Yao : L'interpréteur de commandes, les fonctions standards, la boite à outils, des variables et constantes globales.

- du code généré par Yao : Particulièrement pour les modules et selon leurs attributs : calcul du coût, ajustement, …, parcourt de l'espace en passe avant et arrière en référençant les fonctions des modules, puis leurs dérivées.

Avec ces éléments, Yao va générer un exécutable qui pourra interpréter les commandes saisies dans un fichier d'instructions.

schéma :

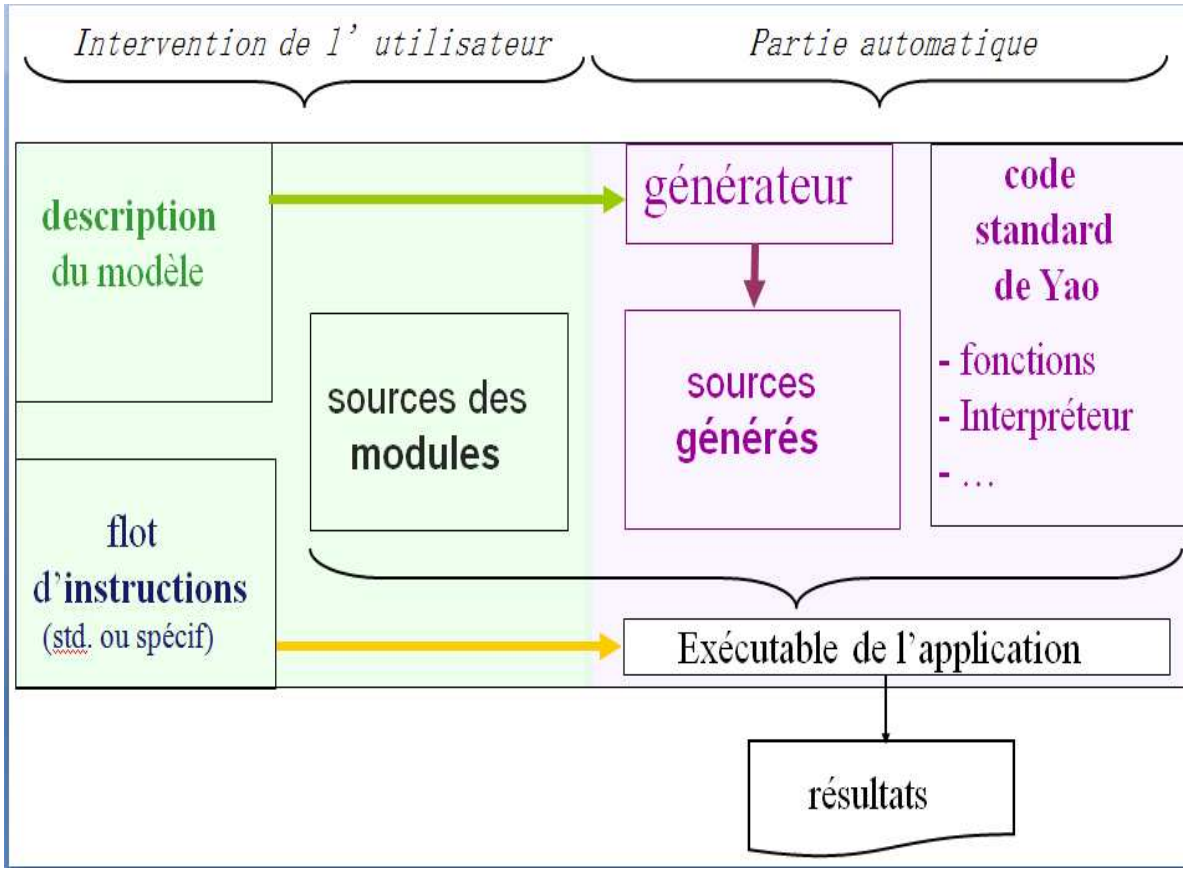


Figure II.d : Organisation d'une application YAO.

Il peut être utiles, pour fixer les idées, de distinguer plusieurs niveaux d'intervention :

⇒ Un niveau ***système*** : c'est la (ou les) personne qui développe et maintient le système Yao proprement dit.

⇒ Un niveau ***réalisateur*** : c'est la personne qui va programmer les parties spécifiques (hats et/ou modules).

⇒ Un niveau ***utilisateur*** : c'est celui qui va créer et utiliser un projet :
- par le biais du fichier de description, pour sa création
- à l'aide d'instructions pour son utilisation.

Les 2 derniers s'incarnent souvent dans la même personne, mais c'est ensuite une affaire d'organisation propre à chacun.

IV.1.2. Spécifications :

Cette partie concerne les développements spécifiques à effectuer par les réalisateurs de l'application et/ou des projets. Il est à noter que, pour se faire, il n'est pas nécessaire d'apprendre un langage dédié puisque le code généré par Yao (écrit en c) l'est en c/c++.

Yao permet un adressage aisé au ***voisinage*** espace-temps grâce à la définition de ***macros***.

Plusieurs ***points d'intervention*** ont été prévus (qui permettent de prendre la main à différents niveaux si cela s'avère nécessaire) :

- au lancement du projet.
- avant et après chaque itération
- avant et après chaque pas de temps (en passe avant puis arrière)

Pour certaines fonctions, l'implémentation de versions spécifiques qui se substituent à celles de Yao est possible pour :

- la fonction de coût
- la correction des paramètres

En prévision, une boite à outils (à enrichir au fur et à mesure) qui met à disposition les opérations courantes (opérations sur des scalaires, vecteurs, matrices, …)

Création d'un projet :

Pour finir, et en résumé, voici la liste des tâches qu'il convient d'entreprendre, et qui sont détaillées dans les prochains chapitres, pour créer un projet :

a) La première étape, qui incombe au réalisateur de l'application, consiste à développer (programmer), à partir d'une analyse fonctionnelle préalablement établie, les spécificités de L'application. Il s'agit en particulier de créer :

- Les sources hats, dont on rappelle qu'au moins un doit contenir les fonctions obligatoires qui permettent à Yao de passer la main au projet (points d'intervention évoqués ci-dessus), mais qui peuvent aussi contenir toutes autres données ou fonctions globales aux projets.
- Les modules sources. Chaque module est sensé contenir au moins :
  - une fonction (la fonction ***forward***) pour la passe avant (schéma direct), et
  - une fonction (la fonction ***backward***) pour le problème inverse (ou passe arrière) correspondant à la dérivée.

(nb : pour référencer ces 2 fonctions simultanément, on utilise parfois le vocable ****ward***)

b) L'utilisateur devra par ailleurs décrire le modèle et générer le projet. C'est l'objet de cette partie.

c) Enfin, des commandes (ou instructions) pourront être saisies pour être interprétées et exécutées et produire un résultat.

Identification des acteurs

Pour ce projet, nous avons identifié deux types d'acteurs et un nombre très réduit des cas d'utilisations répondants aux besoins de ces acteurs.

Les principaux acteurs sont :

L'utilisateur de l'application

Il a pour rôle de définir la description du projet en entrant les directives, d'insérer les modules et en têtes « hat ».

Le code standard de Yao

Il permet d'interpréter et de générer les commandes, il définit ses propres fonctions, il fournit la boite à outils. Il possède un générateur de code.

IV.2. Etude Comparative des analyseurs syntaxique

IV.2.1. Pourquoi faire une étude sur les analyseurs syntaxique ?

En effet, il est impossible de comprendre ce logiciel sans pour autant comprendre peu ou la totalité de sa grammaire qui elle-même se fait grâce à l'analyseur syntaxique. Compte ténu du temps pour la réalisation de ce projet on n'a pas pu changer l'analyseur

syntaxique, on s'est juste contenté de celui qui existe dans l'application étant donné qu'il a été testé avec succès, mais également de comprendre sont fonctionnement, tout en faisant une légère comparaison avec d'autres analyseurs syntaxique existant.

Cette étude permet d'explorer les possibilités futures du logiciel de changer ou non l'analyseur syntaxique ; car Yao se veut évolutif. En effet, cette étude s'est faite sur la base de certains critères qui seront énumérés ci-après.

IV.2.2. Critères de sélection pour le choix d'un analyseur syntaxique

- Analyse une version standard de C++.
- Complexité du parser (pour le comprendre, l'utiliser).
- Utilisable par d'autres programmes.
- Status du parser (en développement, maintenue, abandonné).
- Temps d'analyse.
- Outil implémenté en C++ ou Java.
- Complexité de la grammaire (pour le compilateur de compilateurs).
- Retourne un ASA facilement utilisable.

IV.2.3. Le choix d'un analyseur syntaxique.

Cette partie du projet consiste en l'observation détaillée des analyseurs syntaxiques du langage C++ avec le but précis de trouver le plus performant pour le connecter ainsi à l'outil Yao si possible. Cette première partie se divise en deux étapes. En effet, la recherche de l'analyseur syntaxique se fait, d'un côté, parmi les parseurs et/ou compilateurs, et d'un autre côté, parmi les compilateurs de compilateurs.

IV.2.3.1. Les parseurs et/ou les compilateurs.

Les parseurs disponibles pour le langage C++ sont très nombreux sur le web. Malheureusement, la majorité d'entre eux ont été conçue pour des besoins particuliers et donc, ils peuvent rarement être utilisés pour des programmes tiers. Par exemple, parmi les parseurs étudiés, on retrouve :

· **CINT**

Cet outil est un interpréteur qui peut analyser du code source C++. Dans l'implantation de CINT, on trouve des classes ERTTI (Extended Run Time Type

Identification) qui donnent information sur la structure des programmes analysés. Ces classes sont, par exemple : ClassInfo, BaseClassInfo, DataMemberInfo, MethodInfo, MethodArgInfo, TypeInfo, etc. Malheureusement, cet outil ne donne aucune information sur la partie interne d'une méthode et en plus l'outil contient plusieurs limitations syntaxiques (détails confirmés par l'auteur).

· **John's PCCTS-based C++ Parser**

PCCTS (*Purdue Compiler Construction Tool Set*) est un *embedded* C++ Parser aussi populaire que le GCC Parser. Cet outil a une communauté active des usagers, ce qui a contribué au parser d'être amélioré plusieurs fois. Notamment, John Lilley a contribué avec ce parser pour qu'il soit complet avec un pre-processeur, une table de symboles, etc. Malheureusement, ce parser ne génère pas d'ASA. PCCTS a été écrit initialement par Terence J. Parr, mais il a décidé de faire une re-modélisation complète de PCCTS, ce qui a donné : un compilateur de compilateurs, l'ANTLR.

· **GCC**

GCC (*GNU Compiler Collection*) est l'un des parsers le plus utilisé dans la communauté scientifique. GCC est une collection de compilateurs qui gère plusieurs langages de programmation (C, C++, Java, Fortran, Ada, etc.). Le compilateur pour le langage C++ et G++. Bien que GCC (G++) produit un arbre de syntaxe abstraite pour un programme analysé, il ne donne aucun moyen d'accès à celle-ci, car l'ASA n'est géré qu'intérieurement.

· **PUMA**

PUMA (*Pure Manipulator*) est une librairie de classes pour faire l'analyse lexicale, syntaxique et sémantique tout comme la manipulation de C/C++ code sources. Cet outil peut aussi générer des ASA et est muni d'un pre-processeur intégré.

Lors du téléchargement, on a remarqué que cet outil avait été fusionné avec l'outil AspectC++ (une extension pour la programmation AOP, *Aspect-Oriented*), ce qui rendait plus complexe l'utilisation de PUMA. Toutefois, on conseille fortement d'étudier plus en détail ce parseur dans une future continuation de ce projet.

· **Sage++**

Sage++ n'est pas qu'un simple parseur, mais Il est également un compilateur pré-processeur orienté objet. C'est un outil qui peut restructurer un code source C++ en optimisant les boucles, les variables, etc. Cet outil est conçu pour travailler avec pC++, ce qui est une extension du langage C++ qui est supposément portable et qui permet les opérations parallèles des données. Donc, le problème avec cet outil est qu'il ne supporte pas encore le travail direct avec C++, il faut toujours passer par l'intermédiaire de l'extension pC++.

IV.2.3.1. Les compilateurs de compilateurs et/ou les générateurs de parseurs.

De la même façon que les parseurs, on retrouve de nombreux compilateur de compilateurs et/ou de générateurs de parseurs sur le web. Certains supportent plusieurs langages en même temps (il suffit de se procurer une grammaire appropriée) et certains ont été implantés pour un langage en particulier. Le problème qui revenait le plus fréquemment avec ces outils était celui de se procurer une bonne grammaire (complète et fonctionnel) du langage C++, car sans la grammaire ces outils étaient inutiles pour nous.

· **SLK**

Ce générateur de parseurs se vante d'être le seul « vrai » analyseur LL(k). Les auteurs affirment que cet outil est doté des puissants algorithmes LL(k) que permettent de créer des parseurs plus rapides et plus compacts. Ce générateur de parseurs supporte C, C++, Java et C#. Malheureusement, il faut modifier manuellement SLK pour permettre l'outil de générer des ASA. Toutefois, cet outil semble être très prometteur (si on se fie à la documentation de cet outil) par conséquent une étude plus détaillée sur l'outil en question serait convenable.

· **JavaCUP**

JavaCUP est un générateur de parseurs LALR. Il est écrit en Java est, pour le moment, cet outil ne supporte pas encore C++.

· **Flex/Bison, Lex/Yacc**

Flex/Bison et Lex/Yacc supportent le langage C++. Toutefois, ces outils n'ont pas été testés pour manque du temps et surtout parce qu'on a décidé de mettre en priorité les outils écrits en Java. Notamment, ANTLR et JavaCC.

· **ANTLR**

ANTLR (*ANother Tool for Language Recognition*) est un compilateur de compilateurs qui supporte plusieurs langages et qui est le successeur de l'outil PCCTS, ces deux outils ont été créés par Terence J. Parr. ANTLR est un outil *open source* et il pourvu des plusieurs grammaires déjà fonctionnels. Avec ces 5 000 téléchargements par mois 1, cet outil est muni d'une communauté large et active.

Bref explication du fonctionnement de l'outil ANTLR

ANTLR est un outil, écrit en Java, qui accepte la spécification d'une grammaire d'un des langages supportés. Avec cette grammaire, l'outil produit un parseur qui permet l'analyse du

code de ce langage tout en suivant les règles établies dans la grammaire. On peut ajouter des opérateurs et des actions à la grammaire pour pouvoir générer des ASA et des actions de sortie.

Pour faciliter l'explication du fonctionnement de l'ANTLR, on va supposer qu'on veut écrire une grammaire « Expr ». Cette grammaire doit avoir l'extension ".g" (p.e. Expr.g). Elle doit être constituée des subclasses de ***Lexer***, ***Parser*** et ***TreeParser*** (cette dernière est implémentée seulement si on veut travailler avec l'ASA). Puis, les règles de la grammaire à l'intérieur de ces classes doivent être écrites en notation **EBNF**.

Une fois la grammaire écrite (Expr.g), il faut générer l'analyseur lexical, syntaxique, etc. Cela se fait en utilisant la commande :

$ java antlr.Tool Expr.g

Puis, ANTLR générera les fichiers Java suivants :

ExprLexer.java : Ce fichier Java s'occupe de faire l'analyse lexicale suivant les règles décrites dans la grammaire. Si on regarde dans la classe, on peut remarquer qu'il y aura une méthode pour chaque règle définie dans la subclasse du *lexer*.

ExprParser.java : Ce fichier Java s'occupe de faire le *parsing* suivant les règles décrites dans la grammaire. De la même façon que pour l'analyseur lexicale, on peut noter que pour chaque règle définie dans la subclasse du *parser*, on trouvera leur méthode respective.

ExprParserTokenTypes.java : Ce fichier définira les chiffres à utiliser pour chaque symbole (paramètre constant) qui est utilisé dans le *parser* et/ou le *lexer*.

ExprParserTokenTypes.txt : Ce fichier texte nous montre la liste de symboles avec leurs chiffres respectifs qui ont été généré dans *ExprParserTokenTypes.cpp*.

Ensuite, pour pouvoir utiliser le parseur, il faut écrire une classe Main qui fera appelle aux classes générées.

Maintenant, si on veut travailler avec des ASA, il faut faire quelques modifications à la grammaire. En effet, il faut activer l'option *buildAST* dans la subclasse du *parser* et aussi ajouter de suffixes aux opérateurs pour indiquer lesquels doivent être considérés comme des sous-racines.

class ExprParser extends Parser;

options { buildAST=true; }

expr: mexpr ((**PLUS^**|**MINUS^**) mexpr)*;

…. suite

Notre subclasse du *lexer* ne change pas, mais par contre notre classe Main doit subir aussi quelques modifications pour récupérer l'ASA. Puis, si jamais on veut parcourir l'ASA généré pour pouvoir l'analyser et/ou le modifier, il faudrait ajouter dans la grammaire une subclasse qui hérite de TreeParser avec les règles à utiliser lors du parcours de l'ASA.

Utilisation d'ANTLR avec la grammaire C++

ANTLR fourni l'ancienne grammaire C++ de PCCTS modifiée pour l'utilisation avec ANTLR. On peut observer dans cette grammaire un changement important :

L'ajout d'un flag au début de la grammaire pour indiquer qu'on travaille avec le langage C++. Cela fera en sorte qu'on générera des fichiers ".cpp" au lieu de fichiers ".java".

header { … }

options { language = "Cpp";}

.... suite

class ExprParser extends Parser;

.... suite

class ExprLexer extends Lexer;

.... suite

Les fichiers générés par ANTLR pour notre grammaire C++ sont :

CPPLexer.cpp et **CPPLexer.hpp** : L'analyseur lexicale.

CPPParser.cpp et **CPPParser.hpp** : Le parseur.

STDCTokenTypes.hpp et **STDCTokenTypes.txt** : La liste de symboles.

En examinant bien la grammaire C++, on note que le flag pour générer l'ASA n'est pas actif. En plus, on voit que les opérateurs n'ont pas les suffixes indiquant s'ils doivent être des sous-racines ou non. Alors, on essaie de l'ajouter nous-mêmes pour pouvoir ainsi travailler avec l'ASA. Malheureusement, étant la grammaire très complexe, on se rend compte rapidement que cette tâche était beaucoup plus difficile qu'on l'imaginait. Mais, en faisant une recherche sur le Web, on arrive à trouver une autre grammaire de C++ ayant les spécifications nécessaires pour générer des ASA, mais le point faible de cette grammaire est qu'elle est dans un état incomplet et expérimental.

Bien que, à la limite, on puisse s'en passer de l'ASA pour intégrer le parseur à l'outil de rétro-conception des programmes, on trouve que, en ayant l'ASA, cela facilitera notre travail grandement.

Alors, on décide de continuer notre recherche avec le but d'en trouver un parseur qui permet cette action. Heureusement, l'outil JavaCC vient à notre rescousse.

· **JavaCC**

Le programme « JavaCC » (*Java Compiler Compiler*) est, comme son nom l'indique, un compilateur de compilateurs ou un générateur d'analyseur lexical et syntaxique. Il s'agit donc d'un outil qui lit les spécifications d'une grammaire et qui les convertis en un programme en « Java » qui peut ensuite analyser du code suivant la grammaire. Ce programme est gratuit, disponible pour tous et depuis 2003,

« JavaCC » est *Open Source*. Vous pouvez donc apporter vos propres modifications à ce programme comme bon vous semblerez.

Fonctionnement

En tout premier lieu, pour que « JavaCC » fonctionne correctement, nous avons besoin d'une grammaire qui spécifie le langage source que nous voulons analyser, dans notre cas, le langage C/C++. Cette grammaire est sous forme de fichier avec l'extension « .jj ».

Lorsque nous avons une grammaire correcte, sans erreur, et que nous exécutons « JavaCC » avec cette grammaire, il nous créera toujours une série de fichier de base qu'il crée pour chaque analyseur. Il s'agit de fichiers généraux qui sont identique peu importe la grammaire donnée.

• ***SimpleCharStream.java*** : Cette classe représente et gère l'enchaînement des caractères et des jetons (Token) du fichier contenant les sources du langage à analyser. Elle reçoit donc en paramètre pour son constructeur, un « Reader » sur le fichier de source.

• ***Token.java*** : Cette classe permet de représenter un jeton dans un code source d'un langage. Un jeton est la représentation d'un mot ou d'un symbole présent dans la grammaire.

• ***TokenMgrError.java*** : Cette classe, dérive directement de la classe « *Error* », représente une erreur retournée par le gestionnaire de jeton qui sera expliqué plus bas.

• ***ParseException.java*** : Cette classe, dérive directement de la classe « *Exception* », représente une exception retournée par l'analyseur syntaxique qui sera expliqué plus bas.

Ces quatre fichiers seront toujours créés peu importe la grammaire. Ils seront créés seulement s'ils ne sont pas présents dans le répertoire courant. Le seul moment où il sera nécessaire de les remplacer est si les options de « JavaCC » sont modifiées (si l'analyseur doit être statique ou non par exemple.). Donc il est fortement indiqué d'être certain des options de « JavaCC » pour la grammaire avant de faire des modifications dans ces quatre fichiers.

Le programme « JavaCC » produira aussi une série de fichiers qui seront directement dépendant de la grammaire. Il s'agit en fait de l'engin principal qui permet de faire les analyses lexicales et syntaxiques sur le fichier de code source d'après la grammaire.

• ***CPPParser.java*** : Ceci est la classe principale. Elle contient l'analyseur syntaxique. Ce fichier sera la conversion exacte du fichier de grammaire plus toutes sortes de méthodes générées automatiquement pour assurer le bon fonctionnement de l'analyseur. Elle contiendra aussi le « Main » du projet. Habituellement, ce « Main » recevra le nom du fichier source à analyser en paramètre.

• ***CPPParserTokenManager.java*** : Cette classe représente le gestionnaire de jeton, qui est aussi l'analyseur lexical du projet. Cette classe décortiquera donc le fichier source en jeton pour permettre l'analyse syntaxique par la suite. Cette classe implémente l'interface qui suit.

• ***CPPParserConstants.java*** : Une interface qui contient l'association entre les jetons et les noms symboliques. Donc, par exemple le jeton de type « 0 » représentera le symbole « EOF ».

• *Il est à noter que « CPPParser » au début de chaque fichier est différent, tout dépendant du nom du projet donné dans la grammaire. Dans notre exemple, il est « CPPParser », mais pourrait être tout autre.*

Ces trois fichiers seront générés automatiquement à chaque exécution de « JavaCC », peu importe s'ils sont présents dans le répertoire courant ou non. Il est donc fortement déconseillé de faire une quelconque modification dans ces fichiers.

De toute manière, les modifications doivent être faites dans le fichier de grammaire vu que ces trois fichiers en dépendent directement. Donc si une modification doit être faite dans un de ces trois fichiers, elle peut être faite dans le fichier de grammaire.

Sinon, il faudra refaire la modification à chaque exécution de « JavaCC ».

Spécification de la grammaire de « JavaCC »

La grammaire de « JavaCC » est un fichier texte suivant une syntaxe particulière.

La syntaxe peut s'apparenter beaucoup à des langages de programmation impérative ou concurrente comme C/C++ et Java. Comme par exemple :

```
void simple_type_specifier() :
{}
{
(
builtin_type_specifier()
|
qualified_type()
)
}
```

```
void builtin_type_specifier() :
{}
{
“void” | “char” | “short” | “int” | “long” | “float” |
“double” | “signed” | “unsigned”
}
```

Ici, on veut représenter un noeud de type « simple_type_specifier() » qui est soit un « builtin_type_specifier() » ou un « qualified_type() ». Et ensuite, on voit que « builtin_type_specifier() » peut être soit « void », « char », etc. Il est donc assez simple de lire ou modifier une grammaire de « JavaCC ».

Donc, chaque nœud ou élément de la grammaire est représenté par une méthode ou fonction dans la syntaxe. Dans le premier couple d’accolade, nous pouvons inscrire du code « Java » d’initialisation, comme des variables locales par exemple.

Ce code sera simplement copier dans l’analyseur résultant sans aucun changement.

Dans la deuxième paire d’accolade, il s’agit de code suivant la syntaxe de « JavaCC ». Comme exemple simple, la barre « | » représente un « ou » entre des choix possibles pour la grammaire. Si nous avons besoin d’inscrire du code source « Java » dans la deuxième paire d’accolade, comme ajouter un « return … » par exemple, nous pouvons le faire en ajoutant une autre paire d’accolade. Tout ce qui se retrouvera dans cette nouvelle paire sera automatiquement inscrit dans le résultant sans aucun changement. Il est à noter de bien faire attention où vous placez les accolades, car « JavaCC » pourrait interpréter autre chose que ce que vous voulez faire.

« JavaCC » construit des grammaires de type « LL(k) ». Donc dans la grammaire, nous pourrons inscrire des appels à une fonction spéciale « LOOKAHEAD() » qui permet de faire des tests et de suivre les spécifications.

Pour la représentation des jetons, le mot clé « TOKEN » est utilisé dans la grammaire, comme par exemple :

Nous pouvons aussi représenter les informations qui seront laissé de côté par l’analyseur, comme les commentaires ou les espaces inutiles par exemple.

Au début de chaque grammaire, nous pouvons aussi ajouter un bloc d’option qui nous permet d’exécuter « JavaCC » avec une grammaire sans en connaître les paramètres importants. Comme par exemple, si une grammaire construit un analyseur qui est statique ou dynamique.

```
TOKEN :
{
```

< LCURLYBRACE: "{" >
| < RCURLYBRACE: "}" >
| < LSQUAREBRACKET: "[" >
| < RSQUAREBRACKET: "]" >
| < LPARENTHESIS: "(" >
| < RPARENTHESIS: ")" >
...
}
options {
STATIC=true;
}

Pour ensuite définir le nom de la classe de l'analyseur, la méthode « main » de cette classe ou toutes autres méthodes qui ne seront pas créé explicitement par « JavaCC », nous devons les inscrire dans un bloc spécial. Ce bloc pourra contenir aussi toutes les importations nécessaires au code de l'analyseur, le nom du « package » auquel il va appartenir, etc.
Bien sur, les détails les plus importants se retrouvent sur le site de « JavaCC »,

JJTree

« JJTree » est un utilitaire qui fonctionne de concert avec « JavaCC ». Comme son nom l'indique, il permet de générer un arbre de syntaxe abstraite, ce que « JavaCC » ne fait pas explicitement. « JJTree » prend en paramètre une grammaire de « JavaCC », puis il retourne un fichier de type « .jj.jj » qui est exactement la même grammaire, mais avec du code en plus.
Voici sommairement, comment il fonctionne. Si nous reprenons l'exemple de grammaire précédente :
Ici, quand il est indiqué « Création d'un noeud », « JJTree » ajoute du code « Java » qui permet la création d'un noeud et lorsqu'il est indiqué « Ajout du nœud PARSER_BEGIN(CPPParser) package test.exemple;

public final class CPPParser {
...
}
PARSER_END(CPPParser)
void simple_type_specifier() :
{ **/* Création d'un noeud */** }

```
{
(
builtin_type_specifier()
|
qualified_type()
)
{ /* Ajout du noeud dans l'arbre */ }
}
void builtin_type_specifier() :
{ /* Création d'un noeud */ }
{
"void" | "char" | "short" | "int" | "long" | "float" |
"double" | "signed" | "unsigned"
{ /* Ajout du noeud dans l'arbre */ }
}
```

Dans l'arbre », « JJTree » ajoute simplement le code nécessaire pour ajouter ce nœud dans l'arbre de syntaxe abstraite qu'il est entrain de créer.

Donc, pour que le tout fonctionne, « JJTree » ajoute plusieurs classes au projet pour l'analyseur, donc la plus important, « node.java ». Cette classe permet de représenté un nœud dans l'arbre. Avec des options de « JJTree », nous pouvons faire qu'il nous crée une entité différente de « node » ayant le nom de chaque méthode dans la grammaire. Donc dans l'exemple, il y aurait deux entités,

« simple_type_specifier.java » et « builtin_type_specifier.java » qui hériteraient tous de « node.java ». Sinon, une entité nommée « simplenode.java » sera ajoutée au projet et représentera tous les nœuds. Une option permet aussi d'implanter le patron de conception visiteur pour chaque nœud de l'arbre. Les méthodes nécessaires seront implantées automatiquement à chaque nœud.

À chaque fois qu'un nœud est ajouté dans l'arbre, la référence à ce nœud est envoyée dans la liste des nœuds enfants du nœud qui est en cours de création. C'est ainsi que la hiérarchie dans l'arbre de syntaxe est créée.

Grammaire C++ pour « JavaCC »

Malheureusement, il ne suffit pas d'un bon compilateur de compilateur et d'un bon engin pour bâtir des arbres de syntaxe abstraite pour analyser du code source

C++. Il nous faut une bonne grammaire pour C++. Et pour cela, nous avons du rechercher ailleurs que dans la liste de grammaire du site de « JavaCC ».

Nous avons du utilisé une grammaire disponible dans un paquet de source nommée « PMD ». Cette grammaire, créée en 1997 par le concepteur de « JavaCC », est une grammaire qui fonctionne bien et qui analyse le code C++ standard à cette date. Il construit un analyseur statique qui contient un genre de base de données des symboles supplémentaire. En plus de toutes les classes créées automatiquement par « JavaCC », cette grammaire requiert plusieurs classes qui sont fournis dans le paquet « PMD ».

-*SymtabManager.java* : Cette classe permet de conserver chaque nom de classe ou type spécial. Ceci permet de faire des recherches simples pour les noms des constructeurs ou des destructeurs ou de savoir si un nom de classe est présent dans le code source analyser, etc. Il s'agit d'un engin qui est semblable avec les « IdiomLevelModel ». Il est pratique, mais cet engin nous a causé quelques problèmes que nous avons pu, heureusement, régler.

-*Scope.java* : Contient la définition d'un environnement de variable et de nom de méthode. Des recherches sont faites pour savoir si un nom est contenu dans un scope.

-*ClassScope.java* : Classe qui hérite de « Scope.java » pour définir une classe comme un environnement.

Ces trois classes sont pratiques et nécessaire dans l'implémentation de la grammaire, mais ils pourraient sûrement, avec une bonne analyse, être totalement remplacer par les « IdiomLevelModel ». Cette notion de table de symbole nous a aussi créé quelque problème. Vu qu'il cherchait si chaque symbole (nom de classe, de méthode, de type variable, etc.) était présent dans la table, si, par exemple, une classe faisait référence à une autre classe qui n'est pas présente dans le code source qui est présentement analyser, et bien il sortait une erreur d'analyse. Donc, pour analyser un projet, il fallait que tout soit dans un seul fichier. Ce qui n'était pas très pratique dans « Ptidej UI », alors que nous pouvons recevoir un projet en plusieurs fichiers et même une seule classe par fichier. Heureusement, il fut très facile de court-circuiter le test qui sortait l'erreur d'analyse. Donc maintenant, l'analyseur construit avec cette grammaire peut recevoir un projet en plusieurs fichiers sans erreur.

Pourquoi avons nous choisis « JavaCC » ?

Voici un petit tableau démontrant les principaux critères qui nous ont permis de faire un choix éclairé :

Critères ANTLR JavaCC

Tableau récapitulatif

Comparaisons entre ANTLR et JavaCC

Critères	ANTLR	JavaCC
Analyse une version standard de C++	Oui	Oui
Retourne un ASA facilement utilisable	ASA possible*	Oui
Utilisable par d'autre	Oui	Oui
Sans interface graphique	Oui	Oui
Outil implémenté en C++ ou Java	Java	Java
La grammaire C++produit des fichiers C++ ou Java	C++	Java
Grammaire facile à utiliser	Un peu	Oui

*ASA est possible, mais doit être implanté manuellement. Plus compliqué à faire.

Figure II.e : Comparaison entre ANTLR et JavaCC.

*ASA est possible, mais doit être implanté manuellement. Plus compliqué à faire.

Donc, comme on peut le remarquer, les deux analyseurs sont presque semblables sur tous les points. Ils sont tous deux en « Java ».

Ensuite, l'autre aspect très important est la fabrication d'un arbre de syntaxe abstraite. Donc il faut absolument avoir accès à l'ASA. « JavaCC » nous offre la meilleure possibilité avec l'utilitaire « JJTree » qui fait tout le travail. Il est bien sur possible de faire un ASA avec « ANTLR », mais il faut le faire manuellement. Une tâche ardue à comparer de « JavaCC ». En plus, « JavaCC » offrait une grammaire un peu plus facile à modifier que « ANTLR ».

Donc, en conclusion, on voit assez bien pourquoi nous avons choisit « JavaCC » au lieu de « ANTLR » pour analyser le code C++.

IV.2.4. Présentation détaillé de l'Outil Antlr

Présentation :

ANTR *Another Tool for Language Recognition* : un outil de reconnaissance de langage, est un générateur d'analyseur syntaxique et lexical permettant de générer du code Java ou C++.

Sa caractéristique est d'être LL(K).

ANTLR, est un outil qui accepte des descriptions grammaticales de langue et produit des programmes qui identifient des phrases dans ces langues. En tant qu'élément d'un traducteur, nous pouvons augmenter nos grammaires avec les opérateurs simples et les actions pour dire à ANTLR comment produire du rendement. ANTLR sait produire des systèmes de reconnaissance dans Java, C++, C #, et bientôt python.

ANTLR sait établir les systèmes de reconnaissance qui appliquent la structure grammaticale à trois genres différents d'entrée : (i) les caractères streams, (ii) token streams, et (iii) les structures arborescentes bidimensionnelles. Naturellement ceux-ci correspondent aux lexers, parsers, et tree walkers. La syntaxe pour indiquer ces grammaires, *le métalangage*, est presque identique dans tous les cas

Les programmes de génération ET les programmes utilisés lors de l'utilisation sont dans un package antlr.jar qui doit se trouver dans la variable CLASSPATH.
Vous pourrez utiliser l'une des commandes du genre

```
setenv CLASSPATH ~hpc/lib/Java/antlr.jar:$CLASSPATH
export CLASSPATH=~hpc/lib/Java/antlr.jar:$CLASSPATH
```

suivant le shell que vous utilisez. (Ces commandes sont valables sur les machines FreeBSD de l'UVSQ)

Besoin :

Analyser une expression utilisant un langage complexe, à des fins diverses (traduction, interprétation, compilation, formatage, etc.)

Analyse

ANTLR combine l'analyseur lexical et les spécifications de parseur dans un fichier. Il peut générer des arbres syntaxiques abstraits (AST) et des classes pour le parcourir. Il permet un contrôle très fin du *parsing* des prédicats.

ANTLR accepte trois types de spécification de grammaire :

- *lexer* : identification des éléments lexicaux (*tokens*) d'un langage à fournir au *parser*
- *parser* :
- *tree-parsers* (ou *tree-walkers*).

ANTLR utilise effectue des analyses de type *LL(k)*, ce qui signifie que vous pouvez examiner "en avant" (*Look Ahead* ou *LA*) des éléments lexicaux à un niveau k > 1.

Parce qu'ANTLR utilise effectue des analyses de type *LL(k)* pour l'ensemble de ces grammaires, leurs spécifications sont similaires, et les lexers et parsers générés (en Java ou C++) se comportent de la même manière.

Exemples :

Un exemple de *lexer* ANTLR est :

class MonLexer **extends Lexer**;

```
// Une ou plusieurs lettres suivie(s) d'un retour chariot
NOM: ( 'a'..'z'|'A'..'Z' )+ RETOUR_CHARIOT
;

RETOUR_CHARIOT
: '\r' '\n' // DOS
| '\n' // UNIX
;
```

Un exemple de *parser* ANTLR est :

```
class MonParser extends Parser;

maRegle
: unNom:NOM
{ System.out.println ("Bonjour, " + unNom.getText()); }
;
```

L'analyse lexicale :

Elle permet à partir d'une grammaire, de générer un analyseur reconnaissant des token. Par exemple le fichier suivant permet de reconnaître les tokens constitués de mots de fin de ligne:

La grammaire :

Dans cette grammaire, on souhaite définir une liste de mots séparés par des retours à la ligne :

// Exemple modifie a partir du tutorial "Getting Started With ANTLR"

```
// http://www.antlr.org/fieldguide/start/index.html

class ExempleLexer extends Lexer;

// Une suite de mots séparés par des retour chariot
NAME
        : ( 'a'..'z'|'A'..'Z'
        | '\350' // e grave
        | '\351' // e aigu
        | '\340'
        | '\347' )+ NEWLINE
        ;
// Une ligne peut-etre definie par 1 ou 2 caracteres
NEWLINE
        : '\r' '\n'     // DOS
        | '\n'          // UNIX
        ;
```

La génération :

La commande

java antlr.Tool ExempleLexer.g

va générer les fichiers

- ExempleLexer.java qui définit le lexeur lui même. Dans ce code définit une classe dont le constructeur prend un InputStream, un Reader ou un InputBuffer en entrée et sur lequel on peut utiliser les méthodes suivantes:
- public Token nextToken () qui renvoie le token suivant
- public final void mNAME(boolean _createToken) qui essaie de lire un token de type NAME.
- public final void mNEWLINE(boolean _createToken) qui essaie de lire un token de type NEWLINE.

  Cette classe hérite de Token.java qui fait partie de la distribution d'ANTLR.
- ExempleLexerTokenTypes.java qui définit les tokens utilisés
- ExempleLexerTokenTypes.txt qui définit la même chose mais sous forme textuelle

L'utilisation :

L'utilisation se fera en général avec un analyseur syntaxique (voir Annexe 4), mais on peut aussi l'utiliser directement dans un programme du genre :

```
// Définition d'un token par antlr
import antlr.Token;
import java.io.IOException;
// Programme simple permettant de lire un flot de token
class MainLexer
{
    public static void main(String argv[])
    {
        // Le lexer
        ExempleLexer l;
        // Le token courant
        Token tok;
        // Le lexer lit sur l'entree standard
        l = new ExempleLexer (System.in);
        // Lire tous les token jusqu'à la fin de fichier
        try
            {
                do
                    {
                        tok = l.nextToken ();
```

```
                        System.out.println(tok);
                    }
                while (l.EOF != tok.getType ());
            }
        catch (IOException e)
            {
                System.out.println("Erreur d'entrée sortie");
            }
    }
}
```

Il faudra alors pour tout utiliser compiler le tout:

javac ExempleLexer.java ExempleLexerTokenTypes.java MainLexer.java

Pour résumer, voici un exemple d'utilisation:

```
% java antlr.Tool ExempleLexer.g
ANTLR Parser Generator   Version 2.5.0   1989-1998 MageLang Institute
% javac ExempleLexer.java ExempleLexerTokenTypes.java MainLexer.java
% cat IN
Bonjour
voici
les
tokens
qui
arrivent
les
uns
après
les
autres
% cat IN | java MainLexer
["Bonjour
",type:<4>,line:1]
["voici
",type:<4>,line:1]
["les
",type:<4>,line:1]
["tokens
",type:<4>,line:1]
["qui
",type:<4>,line:1]
["arrivent
",type:<4>,line:1]
["les
",type:<4>,line:1]
["uns
",type:<4>,line:1]
["après
",type:<4>,line:1]
["les
",type:<4>,line:1]
["autres
",type:<4>,line:1]
["null",type:<1>,line:1]
```

Un compteur de mots :

Ces deux fichiers CompterMots.java et LexerMots.g permettent d'implémenter un petit programme qui compte les mots présents dans un texte ou lu sur l'entrée standard.

L'analyse syntaxique

ANTLR permet de définir des parseurs qui utiliseront les tokens définis à l'aide du lexer. La syntaxe est relativement proche de la BNF, dans laquelle on peut insérer des actions à peu près n'importe où.

L'exemple de l'Annexe 4 est un parseur d'un sous ensemble du langage de description de scènes graphiques POVRAY. Il permet de lire un fichier constitué d'une liste de boites et de rotations.

Les fichiers suivants sont nécessaires pour utiliser ce parser :

- MainPovray.java est le programme principal qui instancie le lexer et le parser.
- Box.java et Point3D.java sont des objets qui sont manipulés par le parser lors de l'analyse syntaxique.

La compilation de ces fichiers pourra être réalisée par les commandes suivantes :

java antlr.Tool Povray.g
jikes +D MainPovray.java PovrayParser.java PovrayLexer.java Point3D.java Box.java

La première commande génère le lexeur et le parseur. La seconde compile sources pour obtenir les exécutables.

IV.3. Installation et prise en main de Yao (Ancienne version Yao9)

Pré-réquis :

- Avant de se lancer dans la compilation et l'installation de yao9, il faut,
- au préalable s'assurer de l'existence des paquetages suivants :
- Make, gcc, g++, gfortran, gcc-c++, java, gcc-cpp. S'ils n'existent il faut impérativement tous les installer. On procède comme suite :
- #urpmi make
- #urpmi gcc

- #urpmi gcc-gfortran
- #urpmi gcc-c++
- #urpmi gcc-cpp
- #urpmi java
- On peut obtenir satisfaction si on dispose des 3 cd de Mandriva 2008, mais nous vous conseillons d'allez prendre les mis à jour pour configurer les sources rpm sur le site http://easyurpmi.zarb.org.
- Après la mise à jour de la base des sources rpm nous avons installé le traceur de courbe gnuplot. Il est installé soit en utilisant les cd d'installation de notre distribution soit en ligne. Dans tous les cas on tape la commande urpmi gnuplot.

Installation

- L'installation du logiciel yao version 9 se fait en suivant les instructions indiquées dans le fichier /usr/local/yao/yao9/READMEinstall.txt. Notons que tout ceux-ci a été fait sous Mandiva Linux 2008 official/2008.0
- Ainsi, avions nous jugé utile de placer les variables d'environnements dans le répertoire de l'utilisateur courant, dans notre cas c'est l'utilisateur root, cette procédure nous permet en effet, de ne plus avoir à taper à chaque fois que l'on veut exécuter yao les lignes suivantes :
- YAODIR=/usr/local/yao/yao9/
- MQNDIR=/usr/local/yao/yao9/inria/
- PATH=$PATH:/usr/local/yao/yao9/etc/

- Alors, les lignes suivantes ont été ajoutées dans le fichier .bashrc de l'utilisateur root et de l'utilisateur mayombo.

```
#########################################################
### Environnement YAO
### Definition de la variable YAODIR contenant le chemin a YAO
#########################################################
export YAODIR=/usr/local/yao/yao9
export MQNDIR=/usr/local/yao/yao9/inria
### YAOARCHI permet de definir l'architecture du systeme (Verifiez avec 'uname -
s')
export YAOARCHI=lnx86_Mandriva2008
export PATH=${PATH}:/usr/local/yao/yao9/etc:.
############################ Par MAYOMBO Alexis
############################
```

Création d'un script d'installation pour la nouvelle version

En effet, pour faciliter l'installation de Yao9 un script a été créé afin que les utilisateurs puissent voir leurs taches allégées, ils n'auront plus ainsi besoins d'aller dans les répertoires /usr/local/yao/yao9/inria, /usr/local/yao/yao9/etc et /usr/local/yao/yao9/src pour faire les différentes compilations.

CHAPITRE 3

Conception de la plate-forme

I. Vision abstraite du contenu de la plate-forme

En effet pour que la plate-forme soit opérationnelle, elle doit au minimum intégrer les éléments suivants :

- Une description formelle de chaque niveau de modélisation. Donc au minimum un méta-modèle de PIM (qui fixe le cadre et explicite la sémantique de modélisation, la syntaxe étant généralement celle d'un sous-ensemble d'UML), et un méta-modèle de PSM (qui décrit la cible technique).
- L'implémentation du méta-modèle de PIM dans le modeleur sous forme de Profil UML ou de DSL.
- Un transformateur du PIM vers le PSM
- Des générateurs de code : un par composant du PSM, généralement sous forme de template. Dans ces éléments, deux sont difficiles à construire : le méta-modèle de PIM (appelé aussi "framework applicatif" car il définit le cadre de l'application) et le transformateur "model to model" (PIM vers PSM).

C'est dans ces éléments que réside "l'intelligence" du processus MDA et sa valeur ajoutée par rapport à un processus classique. C'est là aussi que réside sa nouveauté. En effet, un méta-modèle technique est, sinon simple, au moins sans surprise : les architectures techniques sont connues (on en trouve des instances dans toutes les applications réalisées à la main). Et le templating est une activité sans beaucoup de valeur ajoutée (et maîtrisée par à peu près tous les développeurs). Rappelons-nous, l'objectif du PIM est de compresser la complexité de l'application. Ca signifie que la "décompression" (la complexité ne s'est pas envolée comme par enchantement) doit se faire dans le processus de transformation et réapparaître dans le PSM et dans le code.

Le PIM doit donc être très "expressif" : chaque élément modélisé contient beaucoup d'informations qui seront restituées dans la transformation.

Le PIM doit être complet : on doit pouvoir produire tout (ou presque tout) le PSM à partir de lui.

Pour ces raisons, un bon framework applicatif et un bon transformateur PIM vers PSM sont des constructions difficiles donc coûteuses.

II. La modélisation par UML2 :

Comme on l'a dit UML2 possède treize diagrammes. Quant à la catégorie dynamique à elle seule est associée huit diagrammes, le reste est statique. On ne peut pas aller directement à la conception sans faire une petite description du fonctionnement de l'application.

III.Présentation des diagrammes utilisés.

- **Diagramme des cas d'utilisation**

Les diagrammes des cas d'utilisation identifient les fonctionnalités fournies par le système (cas d'utilisation), les utilisateurs qui interagissent avec le système (acteurs), et les interactions entre ces derniers. Les cas d'utilisation sont utilisés dans la phase d'analyse pour définir les besoins de "haut niveau" du système. Les objectifs principaux des diagrammes des cas d'utilisation sont:

- fournir une vue de haut-niveau de ce que fait le système
- Identifier les utilisateurs ("acteurs") du système
- Déterminer des secteurs nécessitant des interfaces homme-machine. (IHM)

Les cas d'utilisation se prolongent au delà des diagrammes imagés. En fait, des descriptions textuelles des cas d'utilisation sont souvent employées pour compléter ces derniers et représentent leurs fonctionnalités plus en détail.

- **Diagramme des séquences**

Les diagrammes des séquences documentent les interactions à mettre en œuvre entre les classes pour réaliser un résultat, tel qu'un cas d'utilisation. UML étant conçu pour la programmation orientée objet, ces communications entre les classes sont reconnues comme des messages. Le diagramme des séquences énumère des objets horizontalement, et le temps verticalement. Il modélise l'exécution des différents messages en fonction du temps.

- **Diagramme d'activité**

Les diagrammes d'activité sont utilisés pour documenter le déroulement des opérations dans un système, du niveau commercial au niveau opérationnel (de haut en bas). En regardant un diagramme d'activité, vous trouverez des éléments des diagrammes d'état. En fait, le diagramme d'activité est une variante du diagramme d'état où les "états" représentent des opérations, et les transitions représentent les activités qui se produisent quand l'opération est terminée. L'usage général des diagrammes d'activité permet de faire apparaître les flots de traitements induits par les processus internes par rapport aux évènements externes.

- **Diagramme des Classes**

Le diagramme des classes identifie la structure des classes d'un système, y compris les propriétés et les méthodes de chaque classe. Les diverses relations, telles que la relation d'héritage par exemple, qui peuvent exister entre les classes y sont également représentées. Le diagramme des classes est le diagramme le plus largement répandu dans les spécifications d'UML. Une partie de la popularité du diagramme des classes provient du fait qu'il existe des outils tels que Rational XDE, ClassBuilder, Omodo for Eclipse, Poseïdon ou Bouml permettant de produire directement du code source dans les principaux langages informatiques (Java, C++, et de C #, Phyton…) à partir de ces modèles (*foward engennering*). Ces outils peuvent synchroniser les modèles et le code, réduisant votre charge de travail. Ils peuvent également produire des diagrammes de classes à partir du code source orienté objet. (*reverse engennering*). C'est la partie la plus importante du modèle MDA, c'est la transformation du PSM au code.

- **Diagramme des Packages**

Très utilisés, indispensables pour les grands modèles, Indispensable pour définir l'architecture de développement d'un modèle, pour gérer un modèle et son développement par une équipe. Utile dès l'analyse est fondamental en conception. Il est difficile de définir la bonne structure d'un modèle (itération nécessaire sur la définition des packages).
UML2 nécessite de retravailler les sous-systèmes, de positionner les composants et la notion de modèle.

III.1 Diagramme des cas d'utilisation :

Le but de ces diagrammes est d'avoir une vision globale sur les interfaces du futur logiciel. Ces diagrammes sont constitués d'un ensemble d'acteurs qui agit sur des cas d'utilisation.

III.1.1. Les acteurs

UML n'emploi pas le terme d'Utilisateur mais d'acteur.

Les acteurs d'un système sont les entités externes à ce système qui interagissent avec lui.

Suivant les besoins de notre système on peut présenter deux acteurs. Il s'agit d'un utilisateur et du noyau de Yao.

III.1.2. Problématique

- Plusieurs problèmes nous sont soumis:
  - Comment permettre à l'utilisateur de générer facilement le fichier de description via l'interface graphique ?
  - Comment faire le contrôle d'erreur via l'interface graphique?
  - Comment insérer les fichiers d'instructions ainsi que les entête .h et les modules .h ?
  - Comment compiler et exécuter une application depuis l'interface graphique ?
- Ceci pour la réalisation de certaines expériences jumelles en vue de l'assimilation variationnelle des données.

III.1.3. PRESENTATION DU DOMAINE

- Notre système doit permettre les fonctionnalités suivantes:
  - Mise en place d'un environnement permettant à l'utilisateur de s'interfacer directement avec le logiciel Yao ;
  - Description du modèle: Fichier .d
  - Coder les Entête .h;
  - Sources des différents modules: Module .h;
  - Coder des instructions: Fichier .i;
  - Compiler l'application;
  - Exécuter l'application créée;
  - Enregistrer le fichier .d, les module .h, les entête .h et le fichier .i;
  - Faire un contrôle d'erreur au besoin.

Détermination des acteurs

Les acteurs de cette application sont divisés en deux groupes suivant leurs actions sur les processus de l'application. Il s'agit :

- De l'utilisateur
- Du noyau de Yao autrement dit du générateur

Diagramme des use case

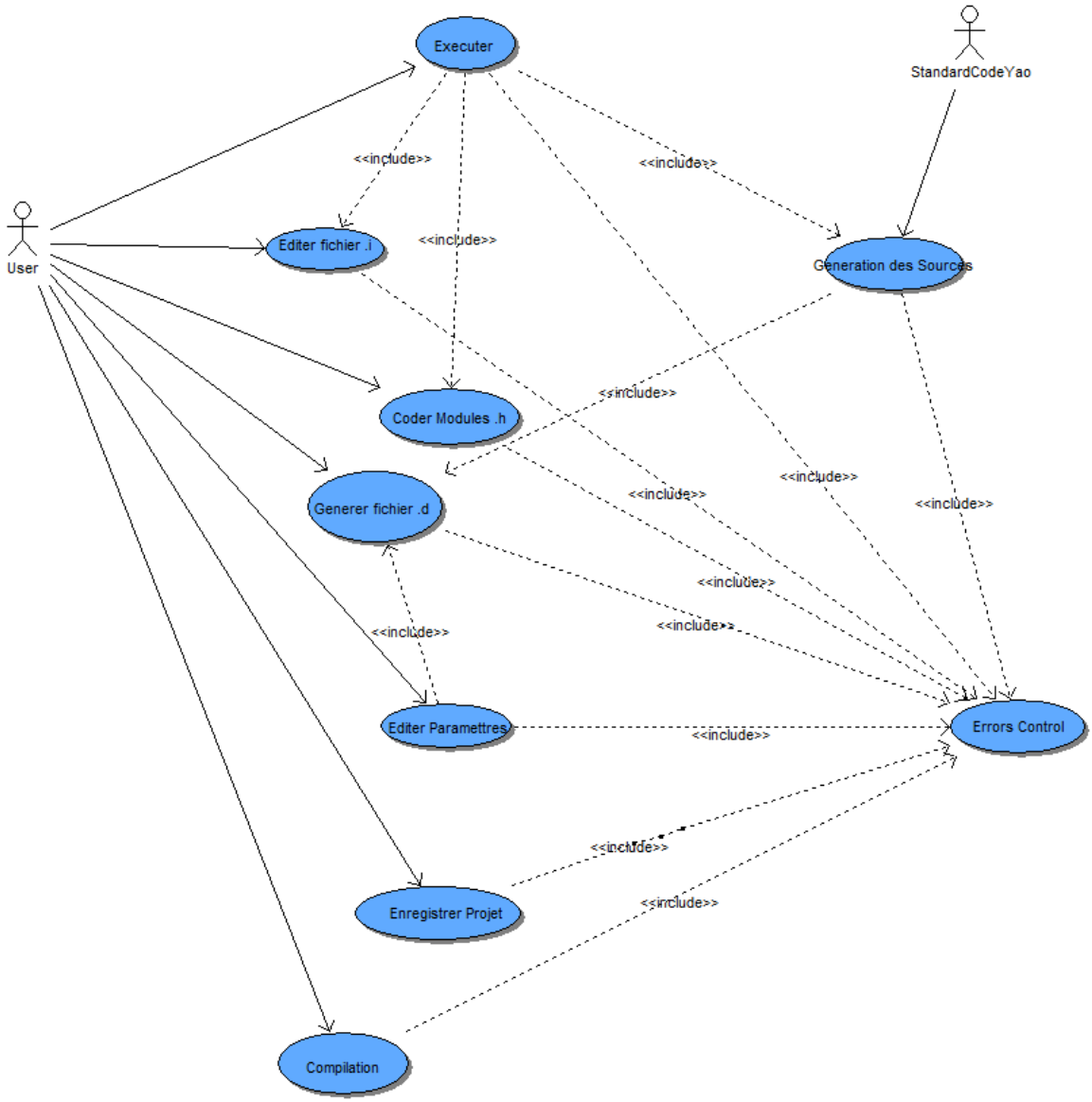


Figure III.a : Diagramme des use case.

Nota : Ce diagramme a été réduit au strict minimum pour une bonne compréhension de l'architecture et du fonctionnement global de Yao. Bien que le que le logiciel soit doté d'une architecture plus complexe, le but était d'amener tout personne ayant peut de connaissance sur la modélisation à avoir une idée générale de Yao.

III.2. Analyse des use case

Nous noterons que l'analyse des différents cas d'utilisation se fera par la présentation du scénario nominal.

Mais nous prendrons uniquement l'exemple de deux (2) use case tout en montrant que les autres se feront suivant le même processus.

III.3. Description des scénarios du use case « générer .d»

1. Créer un nouveau Projet.
2. Donner un nom du Projet
3. Activation de la barre des taches
4. Entrer les Directives de Yao
5. Enregistrer le Projet

III.4. Diagramme de Séquence du use case «Générer fichier .d»

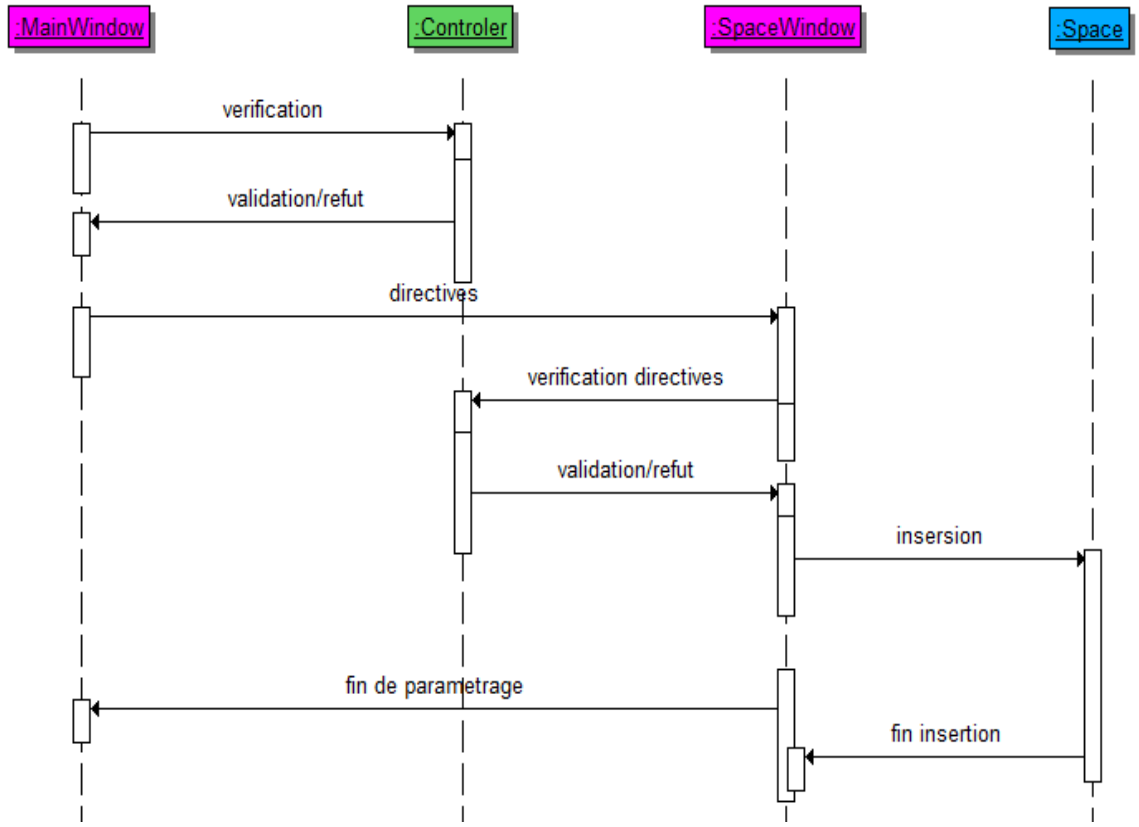


Figure III.b : Diagramme de séquence du use case «générer .d»

Notons, que dans le souci de clarté, seule la classe SpaceWindow a été représenté ici, car les autres classes qui permettent l'insertion des directives se font en suivant le même processus: C'est ainsi, que nous avons jugé utile de ne pas les mettre pour ne pas surcharger le diagramme.

III.5. Diagramme d'activités de la génération du fichier .d

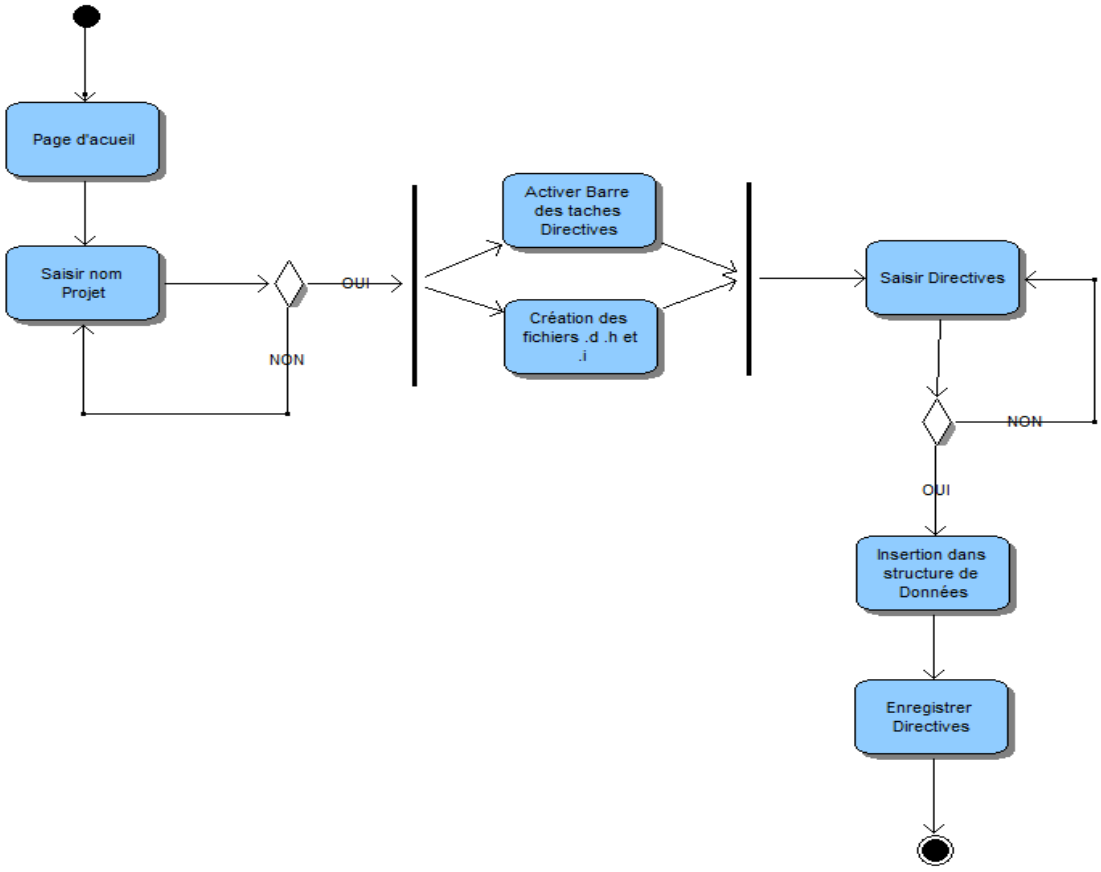


Figure III.c : diagramme d'activité de génération du fichier .d

III.6. Description des scénarios du use case « génération des sources»

Dans ce scénarios les séquences partent de l'instant où l'utilisateur presse sur le bouton Compiler, à et effet, On suppose que le fichier de description a déjà été paramétré au quel cas il y a un contrôleur pour se charger de vérifier tout le processus et d'envoyer un message à l'utilisateur que le fichier de description n'est soit pas totalement paramétré soit inexistant.

1. Lancer la compilation
2. Analyse syntaxique et lexicale
3. Transformation des directives en code C++
4. Génération des fichiers **Y1application_name** et **Y2application_name**

III.7. Diagramme de Séquence du use case «Génération des sources»

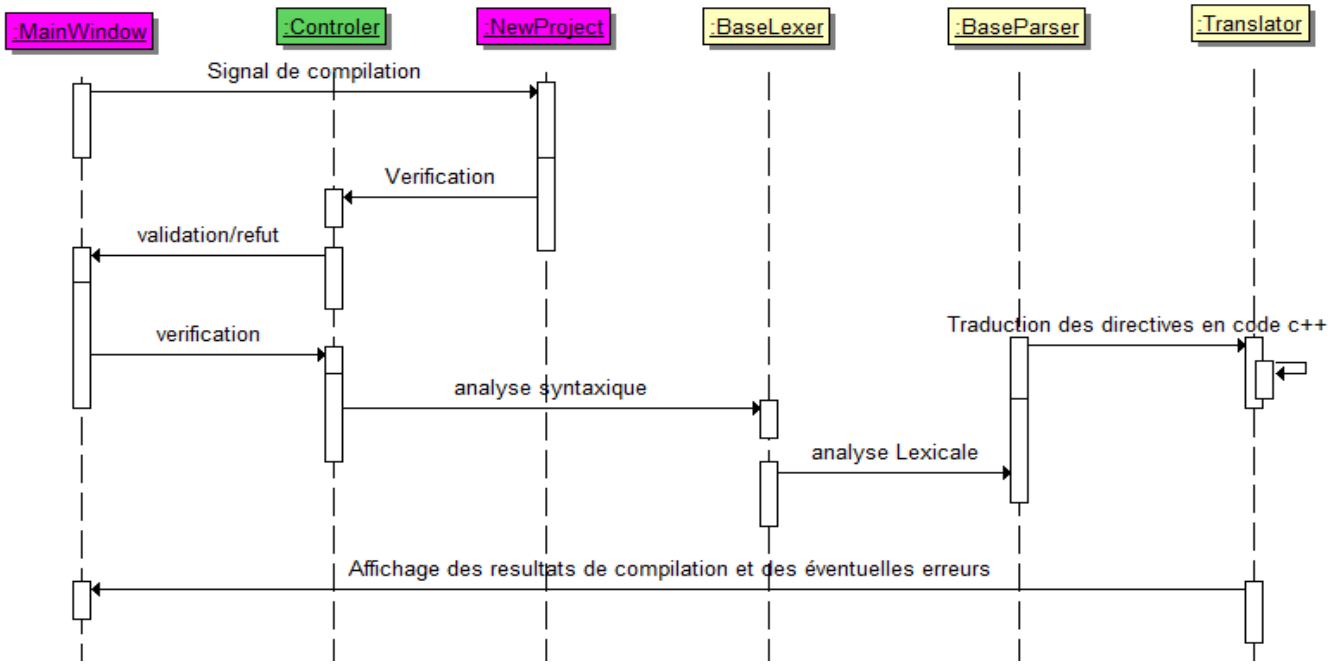


Figure III.d : Diagramme de séquence du use case « générer sources »

III.8. Diagramme d'activité de la génération des sources

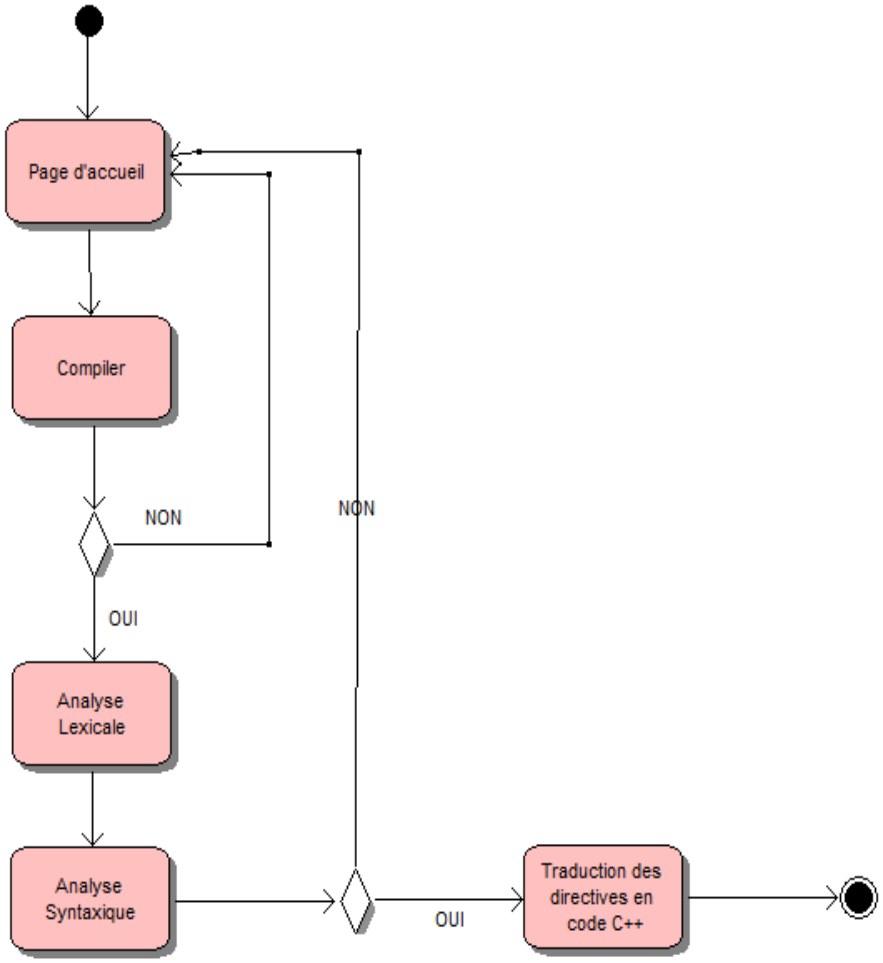


Figure III.e : Diagramme d'activité générer sources

III.9. Diagramme des classes Générale

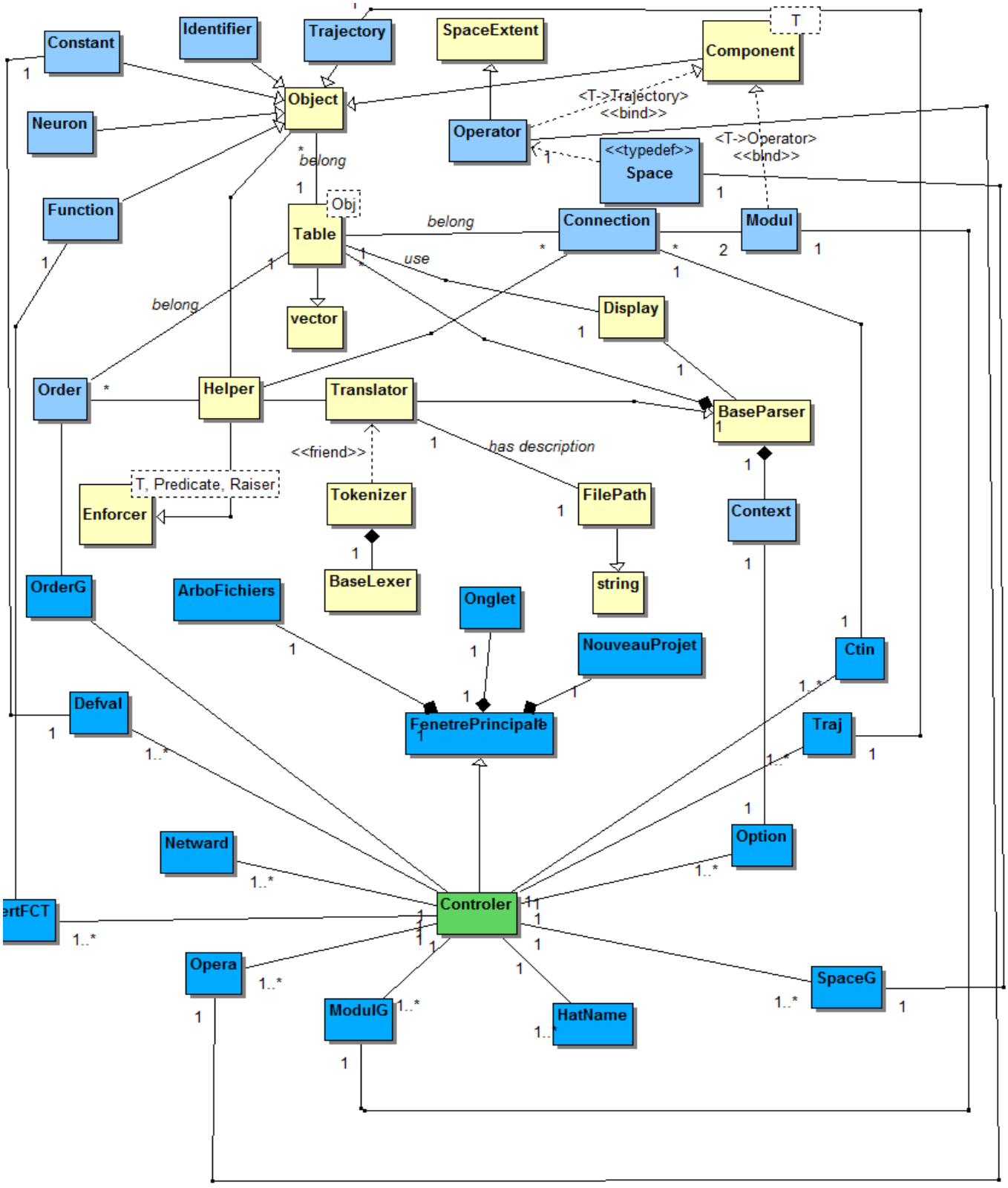


Figure III.f : Diagramme de classe général.

Description de certaines classes :

La classe « MainWindow »

Description

Comme son nom l'indique c'est la principale et la première de l'interface graphique, c'est elle qui fournit le premier contact entre l'utilisateur et la machine.

Méthodes

La méthode «**creerActions()** » permet de créer des actions, afin d'interagir avec certaines partie du projet tel que la compilation, la création du nouveau projet etc.

La méthode «**actionsBarreOutils**» permet de créer les actions qui vont créer les fenêtres pour l'implémentation des directives Yao.

La classe « NewProject»

Description

C'est dans cette classe que seront exécutés les implémentés les instructions de compilation et d'exécution du projet, ces instructions s'exécutent après que l'utilisateur est cliqué sur compiler (par exemple pour la compilation), ainsi la classe **NewProject** reçois des signaux émis par la classe **MainWindow**.

Méthodes

La méthode « **lanceCompilation()** » C'est la méthode qui est chargée d'aller chercher la commande YaoI et de l'associer avec le nom du projet pour lancer le processus de compilation. Cette méthode récupère le nom du projet défini par l'utilisateur lors de la création du projet. Tout ceci se fait bien sûr de manière transparente au vue de l'utilisateur.

La méthode «**creerRepertoireProjet()**» Cette méthodes permet comme son nom l'indique de créer le répertoire du projet ainsi, de définir le nom du projet**.**

Commentaires et approche MVC:

Le but de MVC est de séparer les couches d'une application (en 3 couches distinctes, au minimum, et le plus souvent). On va donc distinguer :

1. Le modèle qui encapsule la logique métier et la manipulation des sources de données. Ici, dans notre cas le modèle est représenté par les classes en bleu, ainsi que les différents fichiers intégrés tel que le fichier de description, les hat et le fichier d'instruction «les données ».

2. La vue : présente les données à l'utilisateur (interface homme/machine (IHM)). Ici aussi on peut introduire des motifs : Factory, composite, TemplateView; par exemple. Dans notre cas, c'est toutes les classes représentés en rose.
3. Le contrôleur : c'est la classe représenté en vert, c'est elle qui analyse les actions, les directives ainsi que les instructions (effectue le contrôle de tous ce que l'utilisateur entre comme information), l'utilisateur, accède aux données, formate le tout, et balance le tout à la partie vue, qui va afficher ça. Nous avons, choisi d'implémenter les contrôles dans une seule classe dans le souci d'une possible évolution de notre application, mais également pour ne pas surcharger le noyau de Yao, qui nous le savons n'est pas encore tout à fait en langage orienté objet, et ainsi cela nous permettra de gagner en terme de temps, en plus cela facilitera la compréhension d'autres développeurs qui aimeraient se mettre sur le projet.

III.10. Diagramme de Package

Un diagramme commun à tous les modèles UML des modules, est le diagramme "Package" qui décrit les liens entre les packages mais surtout qui permet de garder une trace de toutes les modifications appliquées au modèle du module avec les numéros de version.

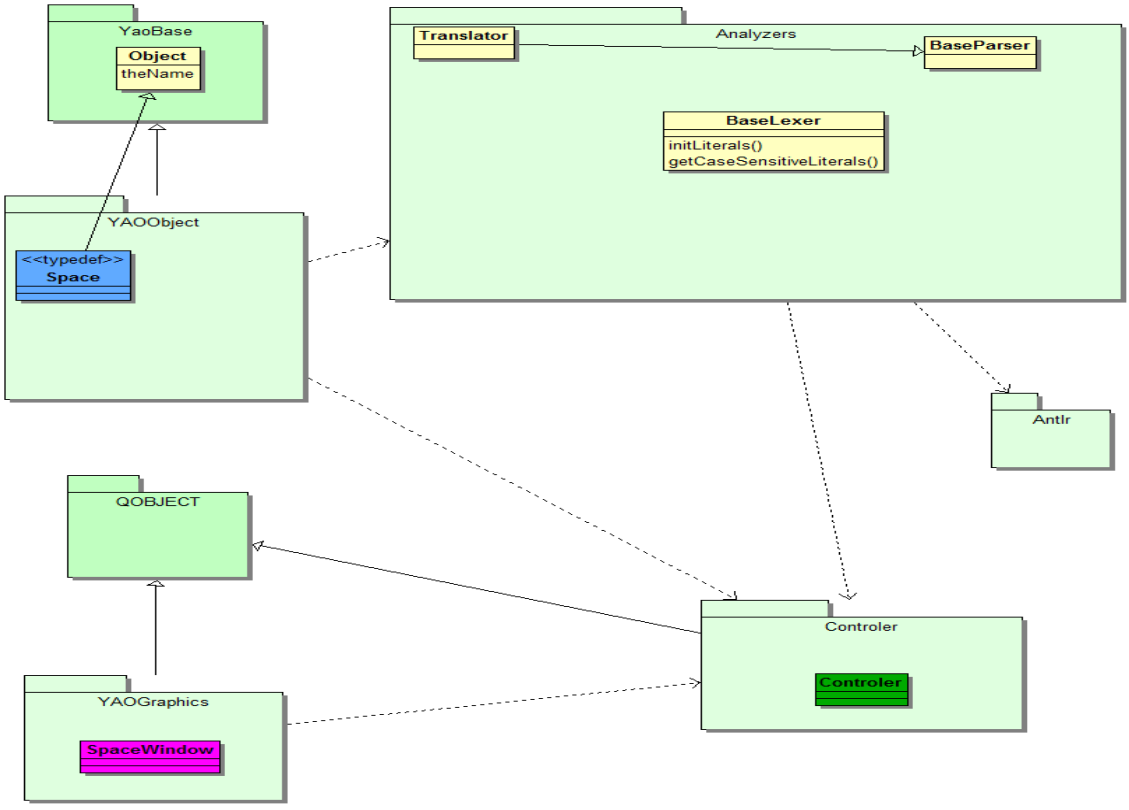


Figure III.g: Diagramme des packages

III.11. Du modèle au code

Comme il a été annoncé en approche de solution du chapitre I, le Modèle MDA permet de partir d'un cadre général pour aboutir à une solution adaptée à nos besoins. Grace à l'outil Bouml nous allons montrer comment passer des diagrammes UML au code C++ dans notre cas.

Les étapes de la manipulation sont :

1 La première consiste à créer un modèle UML.

2 Ce diagramme UML doit être traduit en C++. Nous verrons comment faire la traduction.

3 Ensuite, un modèle de type MOF est généré.

4 Le modèle existant peut être enrichi.

5 Le code C++ est généré à partir des objets enrichis.

6 Une fois les classes générées, il est souvent indispensable d'implémenter les fonctionnalités de ces classes. Les méthodes et variables peuvent être compétées mais le modèle lui-même ne peut pas être modifié (afin de permettre la régénération de code sans perte de données).

7 Si l'on a besoin de générer le code, souvent après avoir enrichi le modèle les ajouts manuels sont conservés.

Cette méthode permet donc un développement incrémental.

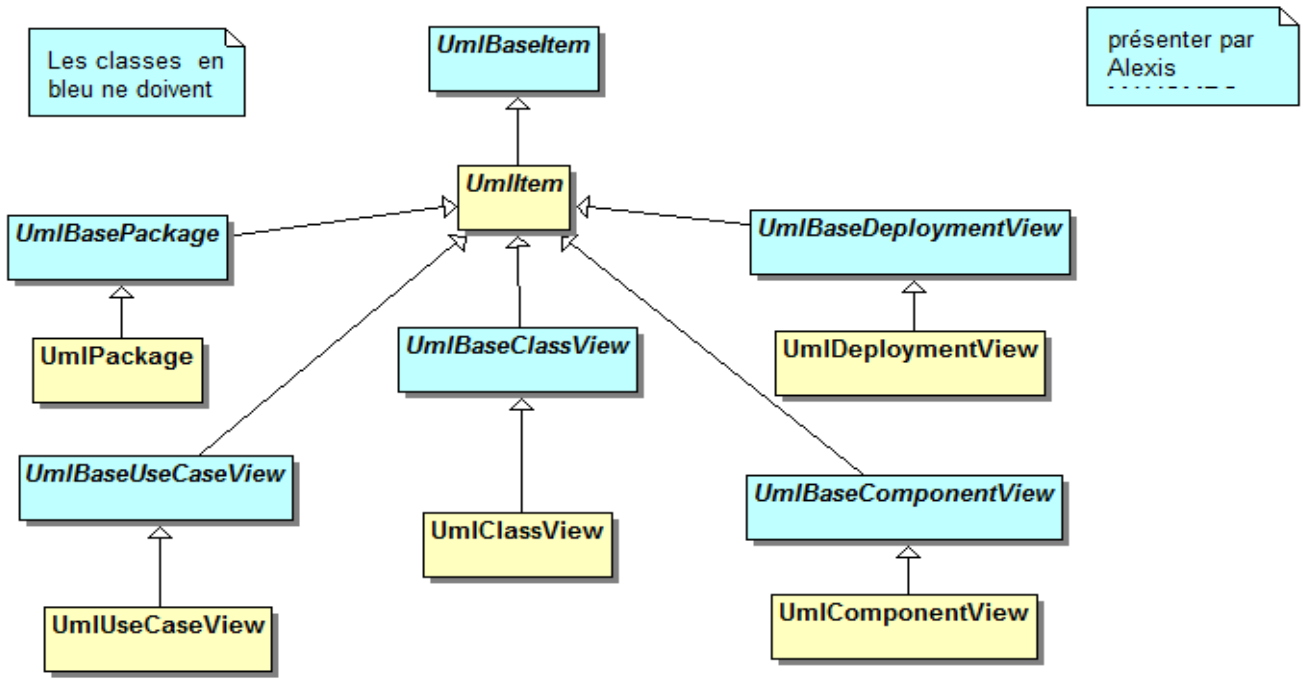


Figure III.h : exemple d'un diagramme des classes.

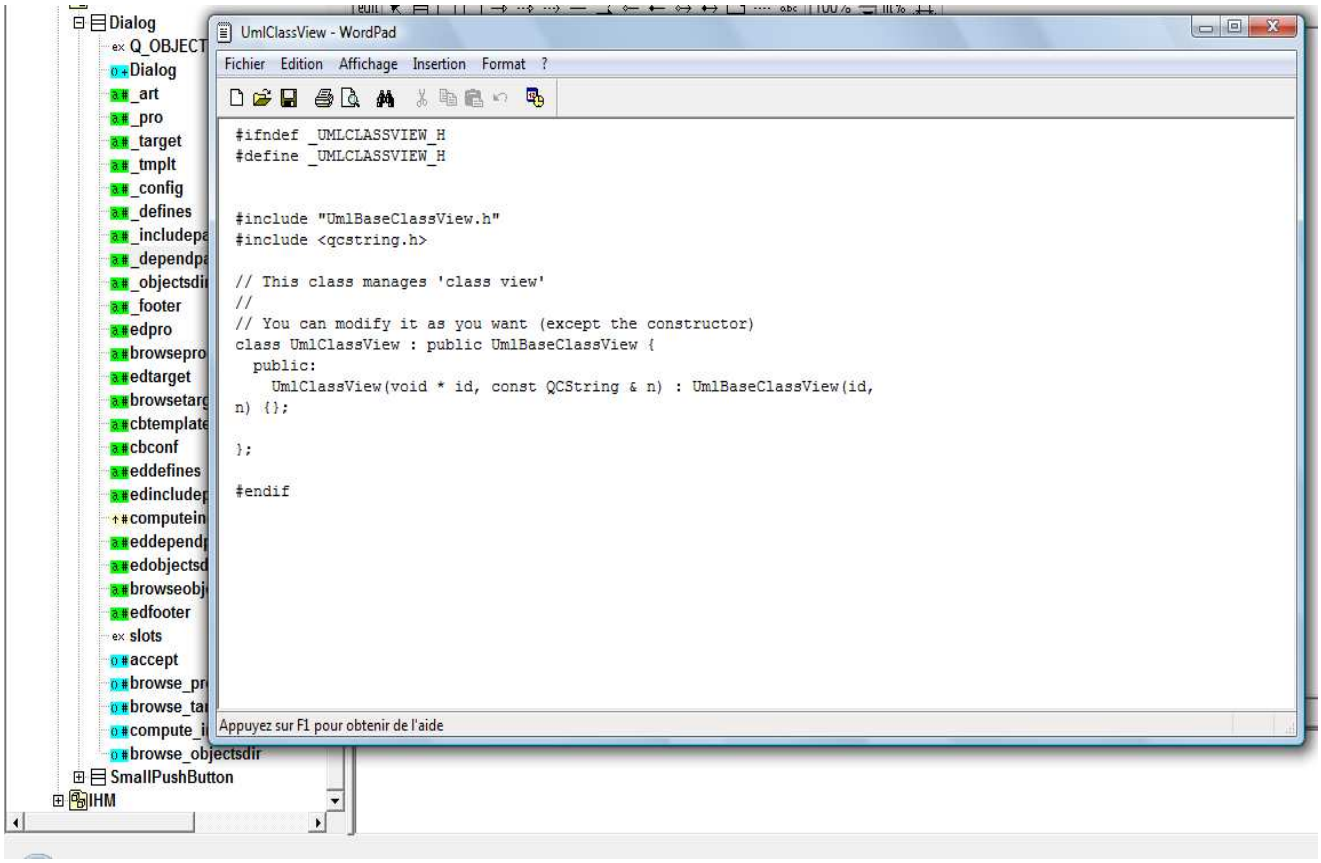


Figure III.i : exemple de génération de code d'une classe précédente du diagramme précédent

III.12. Les règles de gestion

Comment faire que les règles de gestion et, plus généralement, les méthodes et traitements, soient décrites par l'analyste et fassent partie du PIM ? Une idée naturelle serait de les modéliser. Après tout, n'y a-t-il pas tout ce qu'il faut dans UML pour modéliser un algorithme ? Diagramme d'activités, de séquence (depuis qu'on peut, en UML2, définir conditions et itérations), voire de vue d'ensemble des interactions.

Si on veut pouvoir produire le code cible à partir des diagrammes, ils doivent être équivalent à du code, c'est-à-dire être complets et extrêmement rigoureux.

Or les diagrammes dynamiques d'UML n'ont pas été faits pour être compilés : ils sont très difficile à maintenir et surtout, leur syntaxe, telle qu'implémentée dans les modeleurs est toujours incomplète et la plupart du temps beaucoup trop laxiste. Pourtant, derrière l'implémentation d'UML se cache un outil qui répondrait aux besoins : ces diagrammes dynamiques s'appuient sur une syntaxe abstraite, ASL (pour Action Semantics Language) qui permet de décrire toute la dynamique associée au modèle statique.

Une syntaxe concrète (i.e. un langage de programmation) associée à ASL suffirait pour non plus modéliser, mais programmer en UML ! C'est ce que certains outils (un peu confidentiels

malheureusement) comme D.OM ou Kermeta proposent : un langage de script très intégré au modèle UML, permettant d'écrire le corps des méthodes dans le PIM.

Avantage supplémentaire de l'approche : on peut simuler, et donc tester, ces méthodes avant même de disposer des transformateurs vers la plate-forme cible.

« Quels gains peut-on attendre de l'approche »

La mise en œuvre réelle des préceptes que j'ai décrits permettent de très gros gains sur :

- La qualité : le code produit est homogène et répond parfaitement aux normes et bonnes pratiques du développement.
- Les performances : toute optimisation est immédiatement utilisée par l'ensemble de la, ou des application(s).
- L'évolutivité fonctionnelle : l'impact d'une évolution fonctionnelle est immédiatement évaluable.
- La pérennité : via une certaine indépendance vis-à-vis de l'évolution de la technique.
- L'acceptabilité des applications : le cycle de développement très court, et la proximité de l'analyste facilitent le feedback utilisateur.
- La productivité : les charges de développement sont maîtrisées et les gains sont substantiels (-30% sur les coûts) une fois la plate-forme MDA opérationnelle (sans négliger le coût de mise en place de la plate-forme qui s'amortit généralement sur le premier projet, voire sur les premiers "Use Cases" si le projet est de taille suffisamment importante).

CHAPITRE 4

La réalisation

I. Modèles de cycles de vie d'un logiciel :

I.1 Modèle de cycle de vie en cascade

Le modèle de cycle de vie en cascade a été mis au point dès 1966, puis formalisé aux alentours de 1970. Dans ce modèle le principe est très simple : chaque phase se termine à une date précise par la production de certains documents ou logiciels. Les résultats sont définis sur la base des interactions entre étapes, ils sont soumis à une revue approfondie et on ne passe à la phase suivante que s'ils sont jugés satisfaisants. Le modèle original ne comportait pas de possibilité de retour en arrière. Celle-ci a été rajoutée ultérieurement sur la base qu'une étape ne remet en cause que l'étape précédente, ce qui est dans la pratique s'avère insuffisant. (Voir **Figure IV.11**).

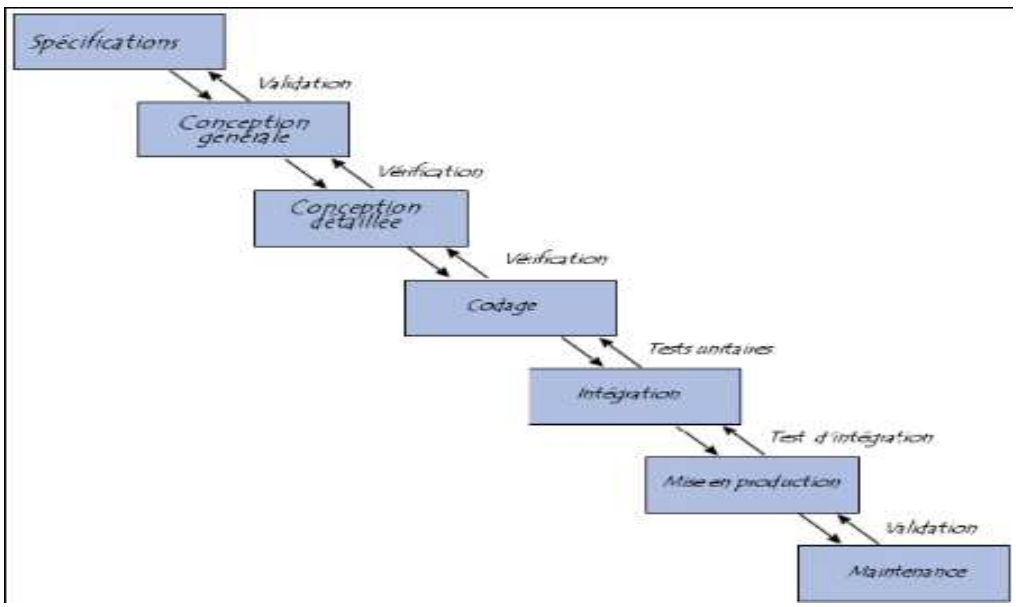


Figure IV.11. *Modèle du cycle de vie en cascade*

I.2 Modèle de cycle de vie en V

Le modèle en V demeure actuellement le cycle de vie le plus connu et certainement le plus utilisé. Le principe de ce modèle est qu'avec toute décomposition doit être décrite la recomposition, et que toute description d'un composant doit être accompagnée de tests qui permettront de s'assurer qu'il correspond à sa description.

Ceci rend explicite la préparation des dernières phases (validation-vérification) par les premières (construction du logiciel), et permet ainsi d'éviter un écueil bien connu de la spécification du logiciel : énoncer une propriété qu'il est impossible de vérifier objectivement après la réalisation. (Voir **Figure IV.12).**

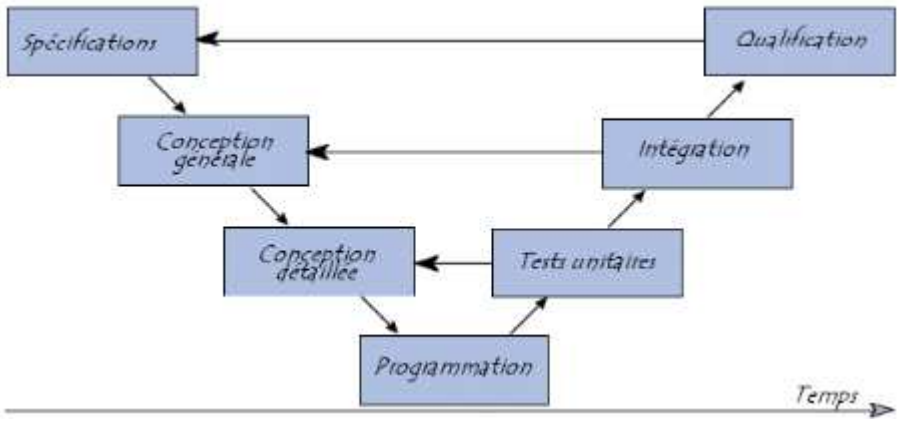


Figure IV.12. *Modèle du cycle de vie en V*

I.3 Modèle de cycle de vie en spirale

Proposé par B. Boehm en 1988, ce modèle est beaucoup plus général que le précédent. Il met l'accent sur l'activité d'analyse des risques : chaque cycle de la spirale se déroule en quatre phases :

· Détermination, à partir des résultats des cycles précédents, ou de l'analyse préliminaire des besoins, des objectifs du cycle, des alternatives pour les atteindre et des contraintes.

· Analyse des risques, évaluation des alternatives et, éventuellement maquettage.

· Développement et vérification de la solution retenue, un modèle « classique » (Cascade ou en V) peut être utilisé ici ;

· Revue des résultats et vérification du cycle suivant.

L'analyse préliminaire est affinée au cours des premiers cycles. Le modèle utilise des maquettes exploratoires pour guider la phase de conception du cycle suivant. Le dernier cycle se termine par un processus de développement classique.

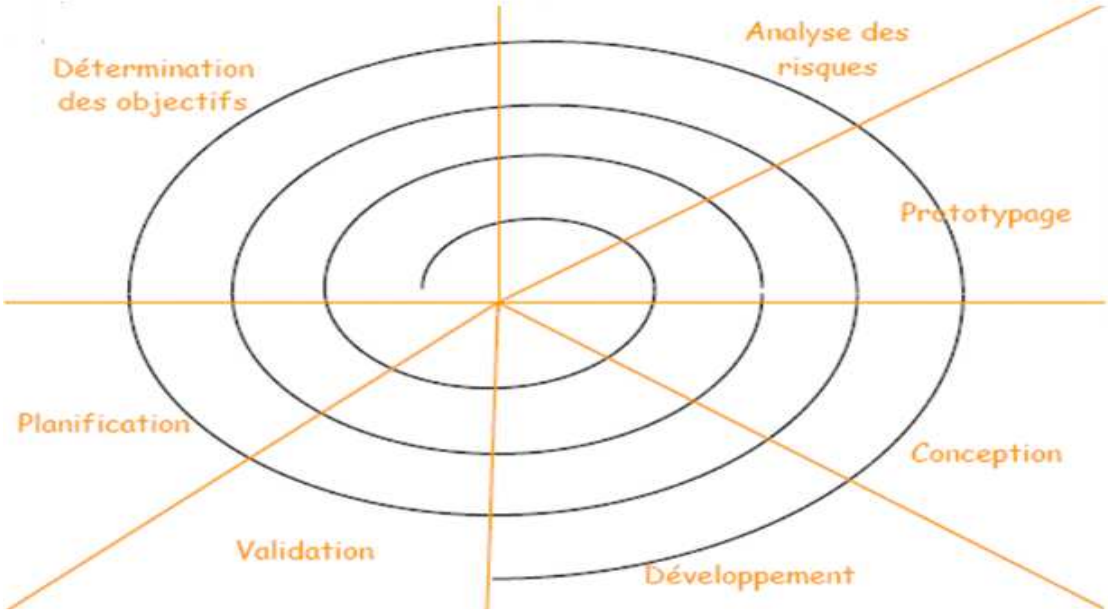


Figure IV.13 : *Modèle de cycle de vie en spirale*

I.4 Modèle par incrément :

Dans les modèles précédents un logiciel est décomposé en composants développés séparément et intégrés à la fin du processus.

Dans les modèles par incrément, un seul ensemble de composants est développé à la fois : des incréments viennent s'intégrer à un noyau de logiciel développé au préalable. Chaque incrément est développé selon l'un des modèles précédents.

Les avantages de ce type de modèle sont les suivants :

· Chaque développement est moins complexe.

· Les intégrations sont progressives.

· Il est ainsi possible de livrer et de mettre en service chaque incrément.

· Il permet un meilleur lissage du temps et de l'effort de développement grâce à la possibilité de recouvrement (parallélisation) des différentes phases.

Les risques de ce type de modèle sont les suivants :

· Remettre en cause les incréments précédents ou pire le noyau.

· Ne pas pouvoir intégrer de nouveaux incréments.

Les noyaux, les incréments ainsi que leurs interactions doivent donc être spécifiés globalement, au début du projet. Les incréments doivent être aussi indépendants que possibles, fonctionnellement mais aussi sur le plan du calendrier du développement.

I.5 Modèle de prototypage :

Un prototype : Un modèle exécutable d'un système logiciel, qui souligne des aspects spécifiques

· Caractéristiques :

Un degré élevé de participation de l'utilisateur.

Une représentation tangible des exigences de l'utilisateur.

Très utile quand les exigences sont instables ou incertaines.

· Avantages: participation de l'utilisateur :

L'utilisateur participe activement dans le développement du produit.

L'utilisateur reçoit des résultats tangibles rapidement.

Le produit résultant est plus facile à utiliser et à apprendre.

· Applicabilité :

Pour des systèmes interactifs de petite et moyenne taille.

Pour des parties de grands systèmes (par exemple l'interface utilisateur).

Pour des systèmes avec une vie courte.

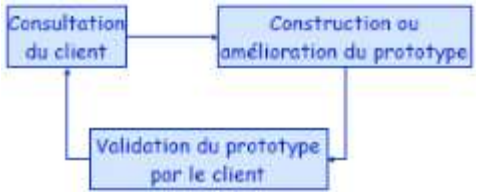


Figure IV.14. *Modèle de prototypage*

Il existe pas mal de modèles, mais ici on a essayé d'expliquer les plus connus et les plus utilisés actuellement. Le choix du modèle est à prendre au sérieux puisqu'il n'y a pas un modèle parfait et c'est difficile de se baser sur un seul modèle, n'empêche d'avoir un modèle de référence. Cependant, un ou plusieurs modèles peuvent bien s'adapter à un cas donné par rapport à d'autres. En ce qui concerne notre application, il s'agit d'un logiciel destiné à un public composé des non informaticiens de formation donc on doit être en relations étroites avec les futures utilisateurs c'est ce qui a été fait tout au long de ce stage, il fallait présenter un modèle et un aspect graphique répondant aux exigences d'un public cosmopolites qui est issus d'horizons divers. Il lui faut de temps en temps des maquettes d'essai. Raison pour laquelle on a choisi le modèle de prototypage qui nous permet de présenter aux utilisateurs un

prototype et l'améliorer jusqu'à avoir un produit fini satisfaisant. On peut faire ici des feed-back.

II. Présentation de l'application développée :

L'application est un logiciel d'assimilation variationnelle des données. La multitude des taches que l'application est capable de faire engendre un grand nombre de fenêtres. Pour des effets esthétiques on a essayé d'utiliser au maximum les avantages qu'offrent Qt selon les informations à afficher. On va essayer de sélectionner quelques unes qui nous paraissent important pour les intégrer dans ce présent mémoire.

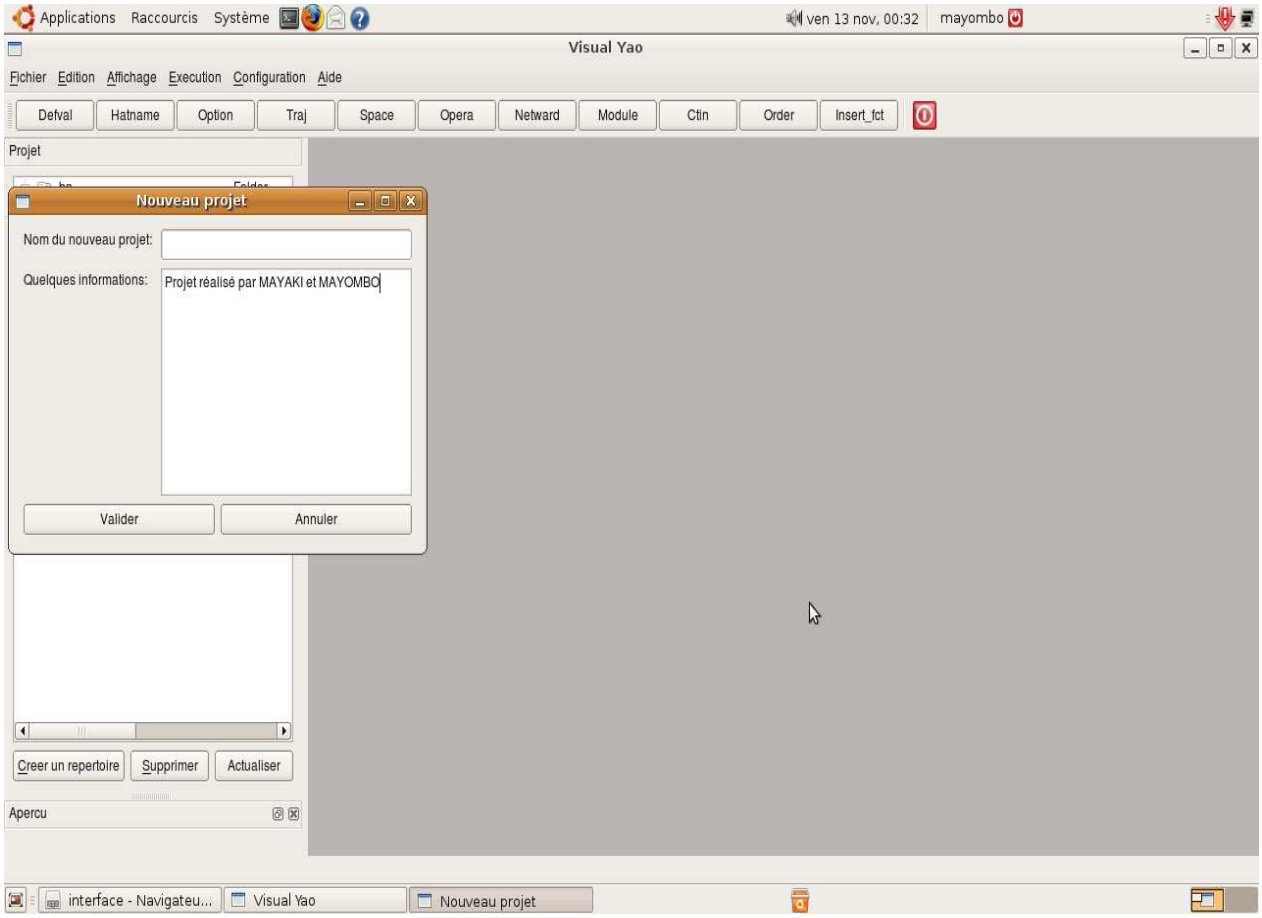


Figure IV.15: fenêtre d'accueil

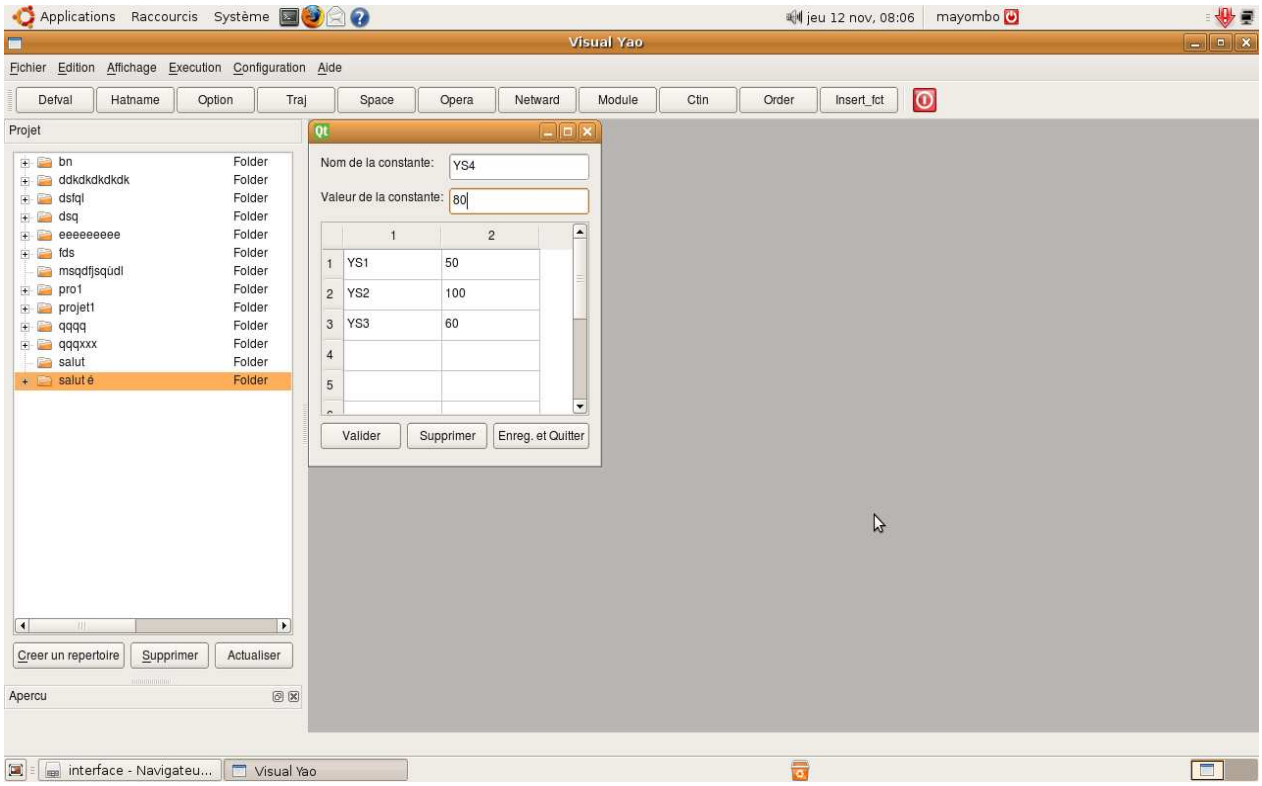


Figure IV.16: Insertion de la directive Defval.

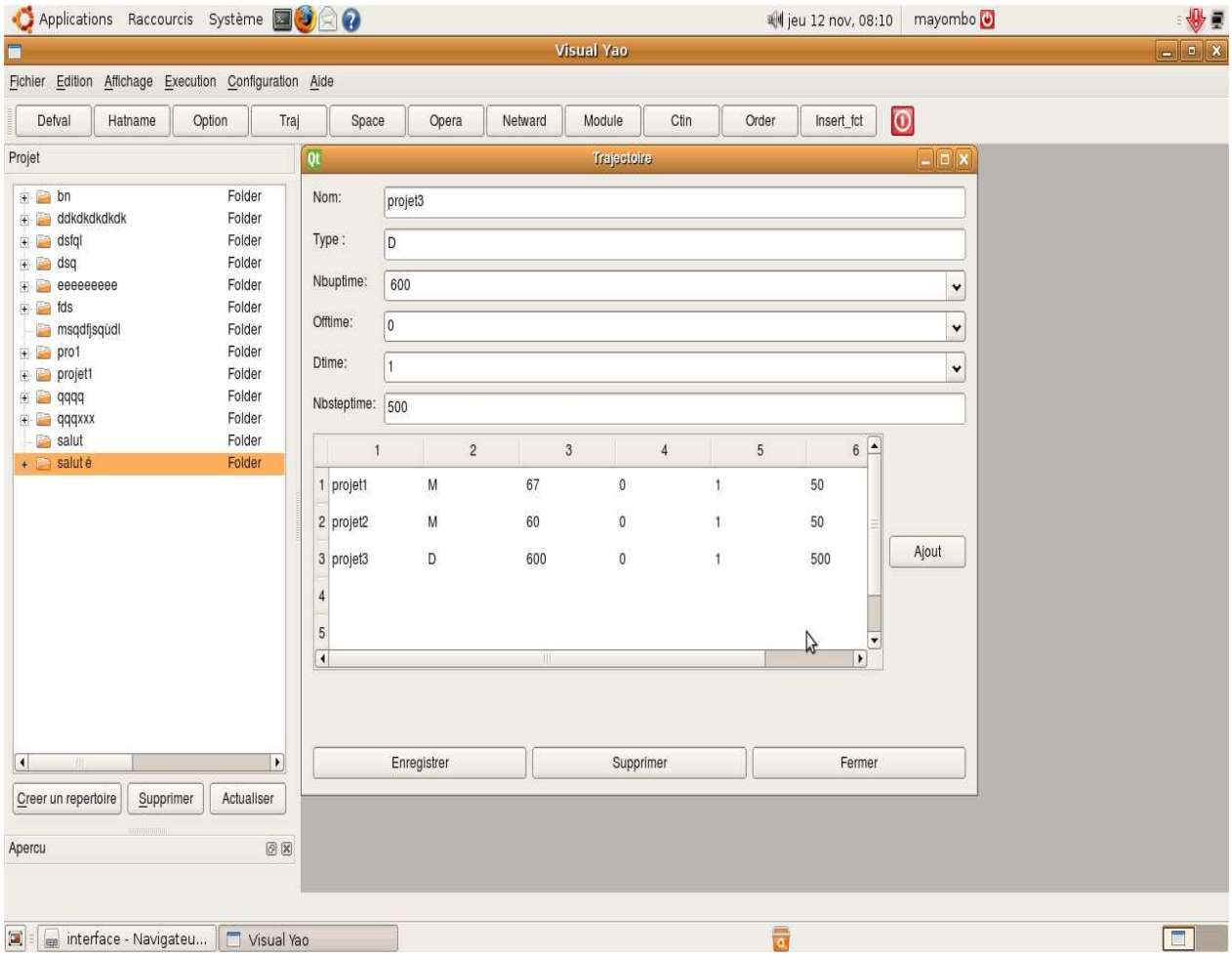


Figure IV.17: Insertion des directives Traj.

III. Intégration de l'outil Gnuplot :

Présentation

Gnuplot est un logiciel **libre,** permettant de **tracer** très rapidement **des courbes** à partir d'une **équation,** ou bien de **relevés des mesures**.

Le maniement de gnuplot nécessite de maîtriser deux commandes, la commande **plot** et la commande **set**. Si on y ajoute la commande **help** (qui fournit des informations sur les options possibles), on a un tour d'horizon assez complet, qui satisfera pour longtemps les besoins de l'utilisateur moyen. Gnuplot est un interpréteur de commande, une fois lancé, Gnuplot interprète un fichier programme ou des commandes en lignes pour produire un graphique 2-D ou 3-D, qui peut être affiché à l'écran, l'intérêt est alors de produire une simulation visuelle et/ou d'avoir un support visuel pour l'intégrer dans un document voir un exemple d'utilisation

en annexe 2. Il est à noter que d'autres logiciels utilisent également gnuplot tel qu'Octave ou Maxima.

III.1. Installation

L'installation se fait soit en ligne pour mandriva 2009 on tape la ligne de commande urpmi gnuplot, pour ubuntu on utilise l'utilitaire apt-get. On peut également l'installer à partir du CD d'installation, car la plus part des distributions linux possèdent les packages gnuplot.

III.2. Utilisation

Dans un shell Linux, taper simplement 'gnuplot'. Une invite permet alors d'entrer ses commandes à la suite. On peut alors entrer des commandes en ligne, visualiser, corriger certaines options et relancer le dessin. On peut aussi entrer ces **commandes dans un fichier** et lancer par **gnuplot** "Nom_de_Fichier" ce qui permet de corriger plus simplement, sans avoir à retaper toute la suite des instructions. Pour Yao il faut utiliser le fichier d'instruction pour préciser que gnuplot sera utilisé, et définir deux fichiers voir l'annexe 2.

Commandes de bases :

1. # : commentaire
   Tout ce qui suit un # est un commentaire. Cela n'est pas pris en compte par gnuplot
2. help : pour demander de l'aide
   par exemple: help plot permet de savoir comment se servir de la commande plot (voir ci-dessous)
3. exit: pour quitter GNUplot
4. cd: change directory
   Permet de naviguer dans l'arborescence, comme sous tous les terminaux linux, unix ou ms-dos
5. pwd: pour savoir dans quel dossier on est.

Fonctions de bases :

1. + : Addition
   Exemple : 2 + 5
2. - : Soustraction
   Exemple : 3 - 1
3. * : Multiplication
   Exemple : 2*x
4. / : Division
   Exemple : 2/x

5. ** : Puissance
   Exemple : 2*x**2
6. sqrt() : Racine carré
   Exemple : 2*sqrt(x)
7. pi : Le nombre Π ("pi" utilisé en trigonométrie)
   Exemple : 2*x+pi
8. cos() : Cosinus
   Exemple : cos(3*x+2*pi)
9. sin() : Sinus
   Exemple : sqrt(2)*sin(x)

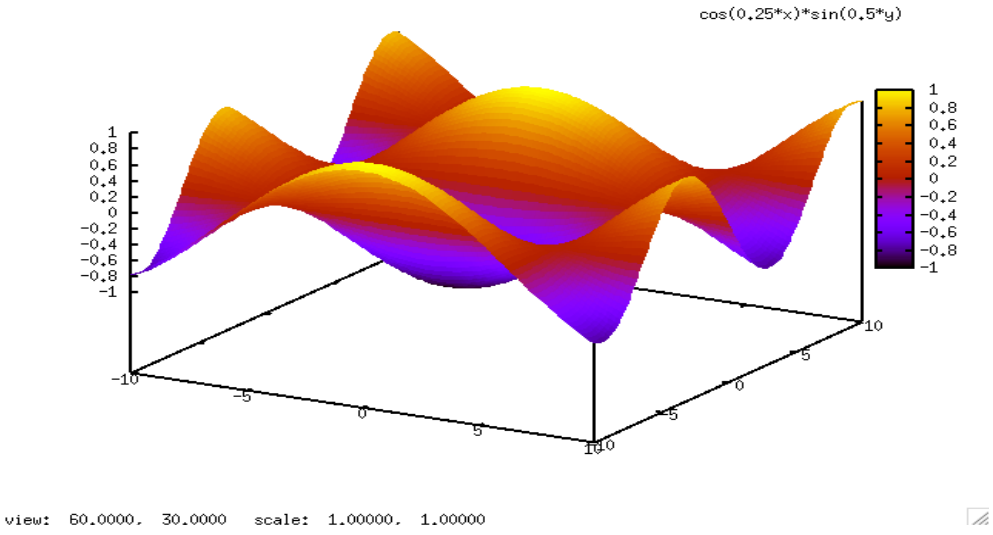


Figure IV.18 : Surfaces de couleur avec pm3d

CONCLUSION

Conclusions :

En compendium, le problème posé dans ce mémoire, était celui de la conception d'une version amélioré de Yao, passant de Yao9 à Visual_Yao. Après avoir posé le problème tout en faisant une brève approche de solution à la problématique liée à notre sujet, nous avons jugé utile d'établir une première approche plus globale afin de mesurer la portée de notre application dans le monde du génie logiciel permettant ainsi, de faire une première analyse sur les problèmes qui devraient être résolus.

Aussi, avons-nous jugé utile de faire une étude générale des outils et méthodes pour élaborer une bonne conception de l'application car rappelons-le l'outil n'était pas construit sur une base conceptuelle rigoureuse, aussi une partie de l'application reste néanmoins fonctionnelle. Ce faisant par la même occasion nous avons fait une étude sur l'existant et évalué l'ensemble des outils et méthodes pour sa mise en œuvre, afin de choisir des outils et méthodes qui devraient nous guider dans la suite.

Par la suite, nous avons élaboré la conception du produit tout en respectant rationnellement les nouvelles démarches dans le souci de pérenniser notre application. Puis, s'en est suivi le développement de notre application.

Enfin, nous avons fait une présentation succincte de l'application développée ainsi que les améliorations apportées.

Perspectives :

Un projet comme celui-ci se construit avec le temps. Les évolutions apportées depuis la première version pour aboutir à la version Yao9 et par la suite amélioré en Visual_Yao version graphique, en sont la preuve.

L'analyse du code existant a pris plus de temps que prévu. Cette étape importante pour la suite du déroulement du projet a empiété sur les délais de réalisation de chacun de nos objectifs. Nous avons, ainsi pu recadrer certains de nos travaux afin de remplir la plus grande partie de notre cahier des charges. Nous avons ainsi entrevu l'importance d'une définition réaliste et pertinente de ce dernier.

Aussi, comme perspectives à venir nous pouvons citer :

- Amélioration des algorithmes présentés.
- Possibilité de définir l'application par l'usage de graphe modulaire sur l'interface.
- Automatisation de l'installation grâce aux outils autoconf, automake et autotool.

Bibliographie

⇒ **Bibliographie**

- C et C++, Sébastien Lecomte, édition First Interactive. 2004.
- Qt4 et C++ Programmation d'interfaces GUI, Jasmin Blanchette et Mark Summerfield, éditeur CampusPress avril 2008

⇒ **Wébographie [Dernières consultations le 15/08/2009]**

- ANTLR http://www.antlr.org/
- CINT http://root.cern.ch/root/Cint.html
- CoCoLab http://www.cocolab.de/
- CppCC http://cppcc.sourceforge.net/
- FOG http://www.computing.surrey.ac.uk/research/dsrg/fog/
- GCC http://gcc.gnu.org/
- JavaCC https://javacc.dev.java.net/
- LEX/YACC http://dinosaur.compilertools.net/
- http://www.bykeyword.com/pages/detail8/download-8937.html
- http://howtos.linuxbroker.com/Lex-YACC-HOWTO.html
- PCCTS http://www.empathy.com/pccts/
- http://www.polhode.com/pccts.html
- PMD http://pmd.sourceforge.net/
- PUMA http://ivs.cs.uni-magdeburg.de/~puma/
- SAGE++ http://www.extreme.indiana.edu/sage/
- SLK http://home.earthlink.net/~slkpg/
- SPIRIT http://spirit.sourceforge.net
- http://www.imada.sdu.dk/~morling/
- http://standartux.fr/index.php?post/2008/07/21/wxWigets-ou-lart-de-programmer-en-C-multiplateforme
- https://javacc.dev.java.net/doc/.
- http://www.gnuplot.info
- http://easyurpmi.zarb.org

Index

A

B

C

D

E

H

I

J

L

M

O

P

R

S

Annexe 1 : Généralité sur l'Assimilation des données variationnelle

Définitions et concepts

- L'assimilation de données est une technique qui vise à estimer l'état d'un système dynamique en combinant des informations d'origines diverses.
- Comment combiner de façon optimale" diverses informations imparfaites (modèles, observations, statistiques, images...) sur un même système complexe ?

a- Le processus d'assimilation

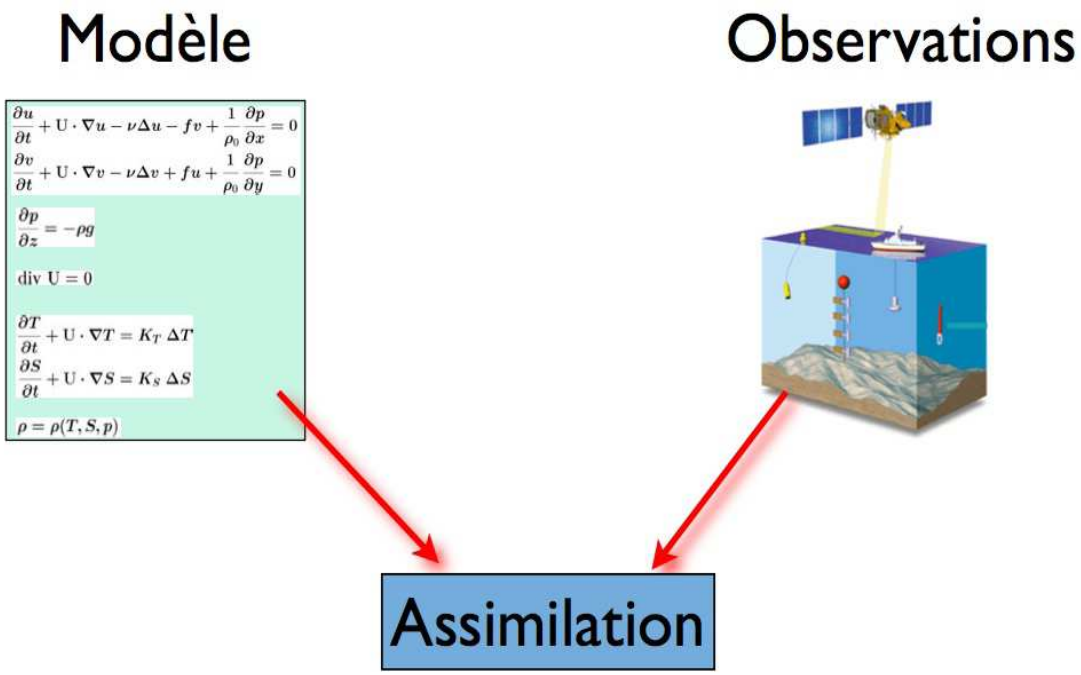


Figure a.19 : Exemple d'un processus d'assimilation

De quoi a-t-on besoin pour assimiler des données ?

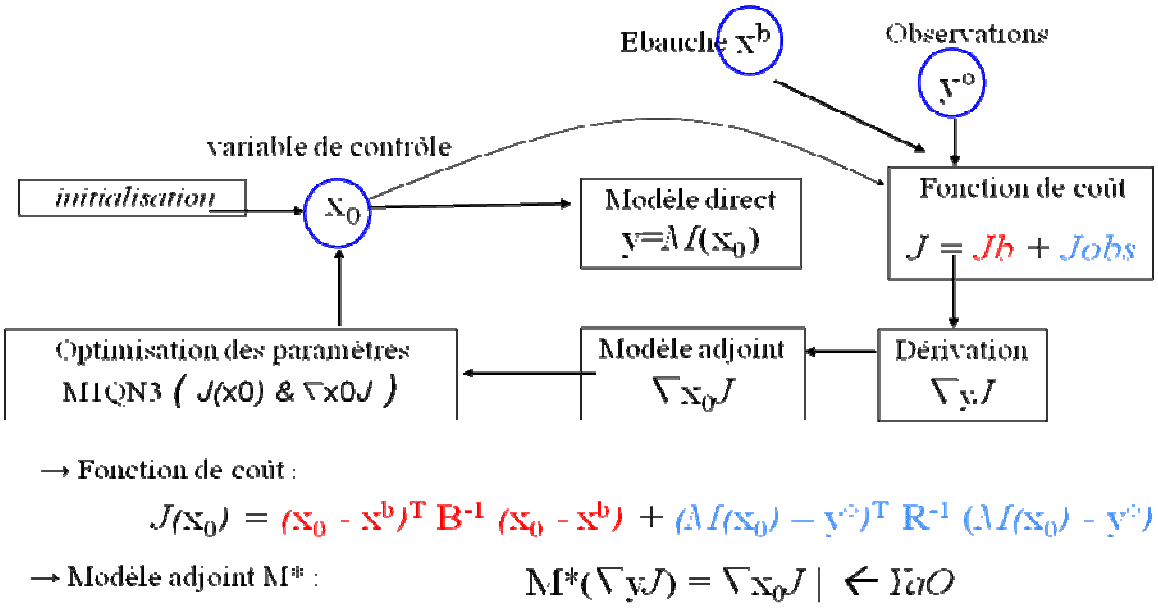
- Description de l'océan par
  - les observations
  - le modèle + C.I
- Techniques d'assimilation
  OI, 3D-Var, 4D-Var, Filtres de Kalman …
  Moyens de calcul et stockage
- Outils logiciels

Objectif : Utilisation au mieux des données disponibles pour optimiser les paramètres du modèle.
Assimilation : Logiciel Yao

Schéma d'assimilation adoptée

Objectif : Utilisation au mieux des données disponibles pour optimiser les paramètres du modèle.
Assimilation : Logiciel Yao

Schéma d'assimilation adoptée



→ Fonction de coût :

$$J(x_0) = (x_0 - x^b)^T B^{-1} (x_0 - x^b) + (M(x_0) - y^o)^T R^{-1} (M(x_0) - y^o)$$

→ Modèle adjoint M* :

$$M^*(\nabla_y J) = \nabla_{x_0} J \mid \leftarrow Yao$$

Figure a.20 : schéma d'assimilation adopté

Annexe 2 : Compilation d'un projet dans Yao

1. Cas particulier : YaoRapport.pdf

Créer un répertoire dans le répertoire /usr/local/yao/yao9/examplesTestYao9/ et copier le tp du projet YaoRapport.pdf un autre.

Dans notre cas nous avons créé le répertoire /usr/local/yao/yao9/examplesTestYao9/alexis/alexis2/

Dans ce TP, nous avons fait face à quelques erreurs de l'original, dont voici quelques modifications apportées :

Lors du lancement de shalw version 9, nous avons du faire face à plusieurs difficultés le logiciel n'étant pas stable, il a fallu changer un peu les fichiers Makefiles et bien d'autres.

[root@localhost ~]# cd /usr/local/yao/yao9/examplesTestYao9/alexis/alexis2/
[root@localhost alexis2]# YaoI shalw

Figure de compilation de shalw:

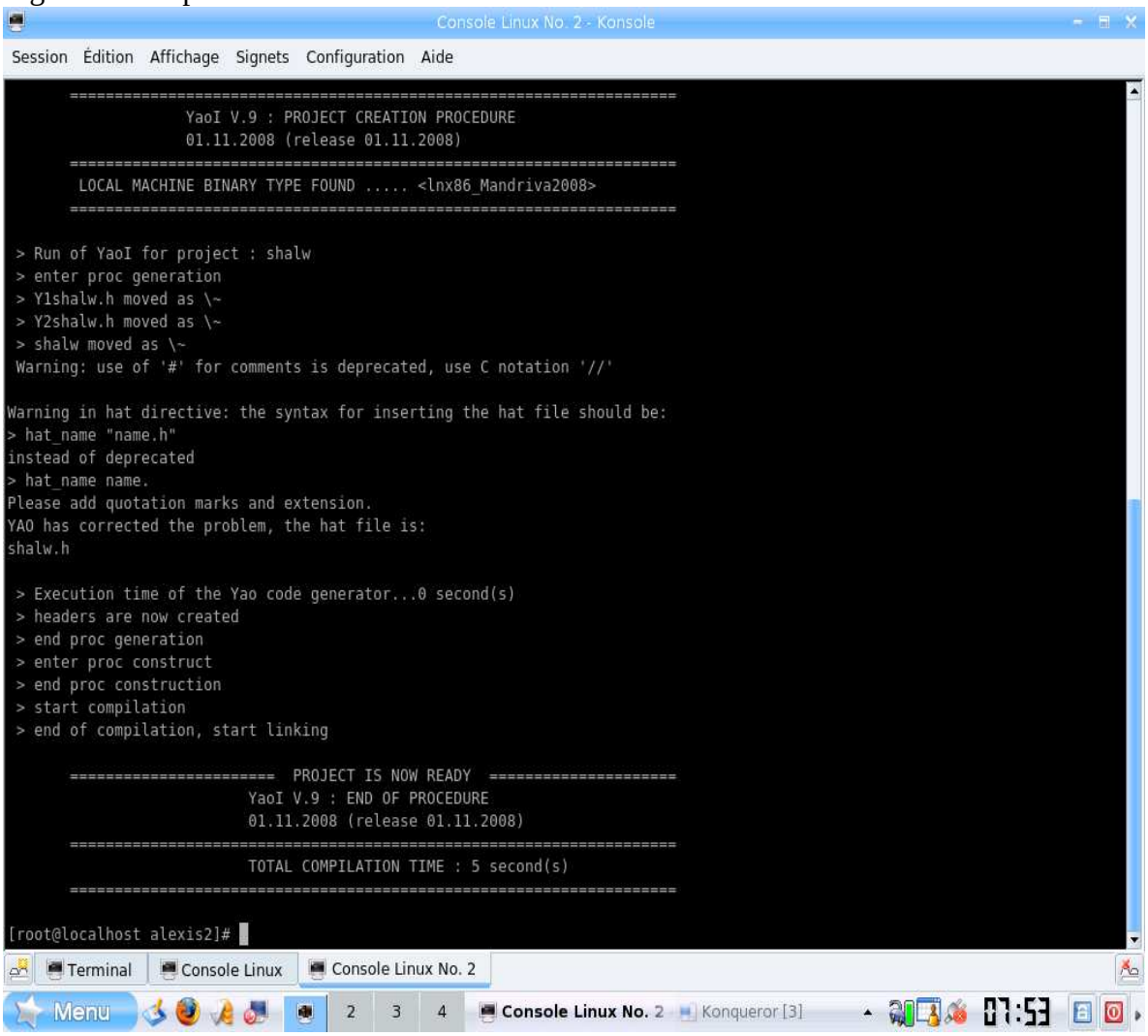


Figure a.24 : extrait de la compilation du projet shalw.

2. Exécution de shalw avec l'éditeur graphique gnuplot

Nous avons au préalable créé deux fichiers hsplot.gp et hanim.gp comme indiqué dans le tp du YaoRapport.pdf.

Le contenue du fichier hsplot.gp est :

```
set zrange [-5 ;15]      #coordonnée de l'axe z (qui correspond à la hauteur)
set pm3d                 #palette couleur 3d
set cbrange [-1 ;10]     #échelle des couleurs
imax=21                  # Nombre maximum de séquence à afficher
idx=0                    #index de début de séquence
load   "hanim.gp"        #script d'animation (splot des séquences de hauteurs)
reset
```

Le contenue du fichier hanim .gp est :

```
splot 'HfilT' index idx with pm3d    #affiche une séquence (selon l'index) du fichier HfilT
idx=idx+1                            #incrément de l'index pour passer à la séquence suivante
if(idx<=imax) reread                 #ré-exécute les instructions de ce fichier si on a pas
atteind
                                     #le nombre maximum de séquences à afficher
Pause  -1 "Hit return to finish"
```

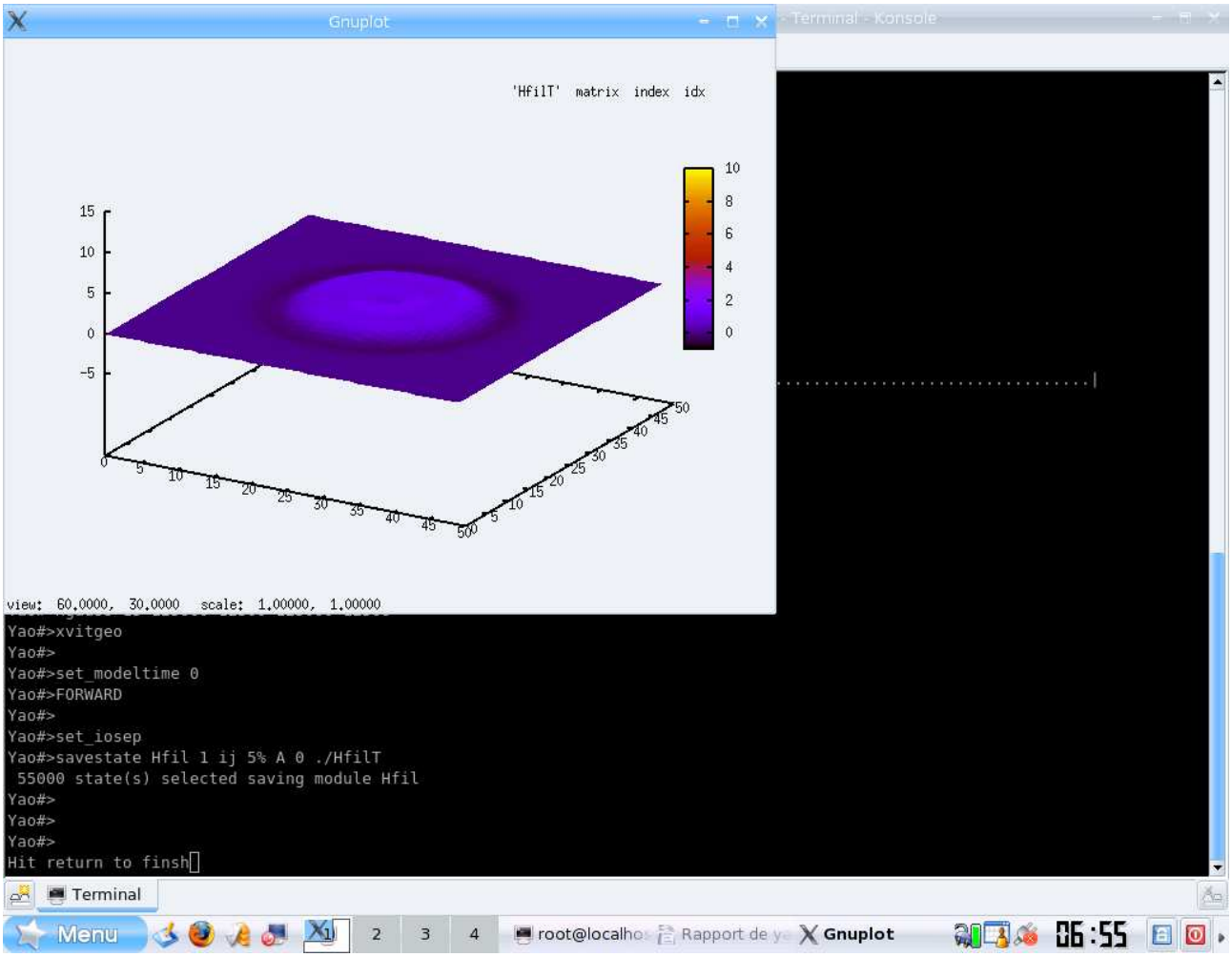


Figure a.25: exécution de shalw par la commande gnuplot hsplot.gp

REMARQUE : On a sauvegardé les résultats de l'exécution dans le fichier execas1.txt en tapant la commande shalw > execas1.txt.

Notons, qu'une fois la commande shalw est lancée, il apparait la figure de gnusplot. Pour avoir le reste de l'exécution fermer la fenêtre du graphe et appuyez sur Ctrl+C au clavier.

Nous pouvons également remarquer que le shalw du TP YaoRapport.pdf est différent du shalw du répertoire /usr/local/yao/yao9/examplesTestYao9/nardi/shalWaterRapportYAOV9/, car le constat a été fait au niveau des modules .h. Tel que ; Hfil.h, Uphy.h, Vphy.h, Hphy.h sont différents dans les positions des variables et les signes dans les fonctions backward, mais, la philosophie reste la même.

I. Exécution du shalw du répertoire /shalWaterRapportYaoV9:

/usr/local/yao/yao9/examplesTestYao9/nardi/shalWaterRapportYAOV9/

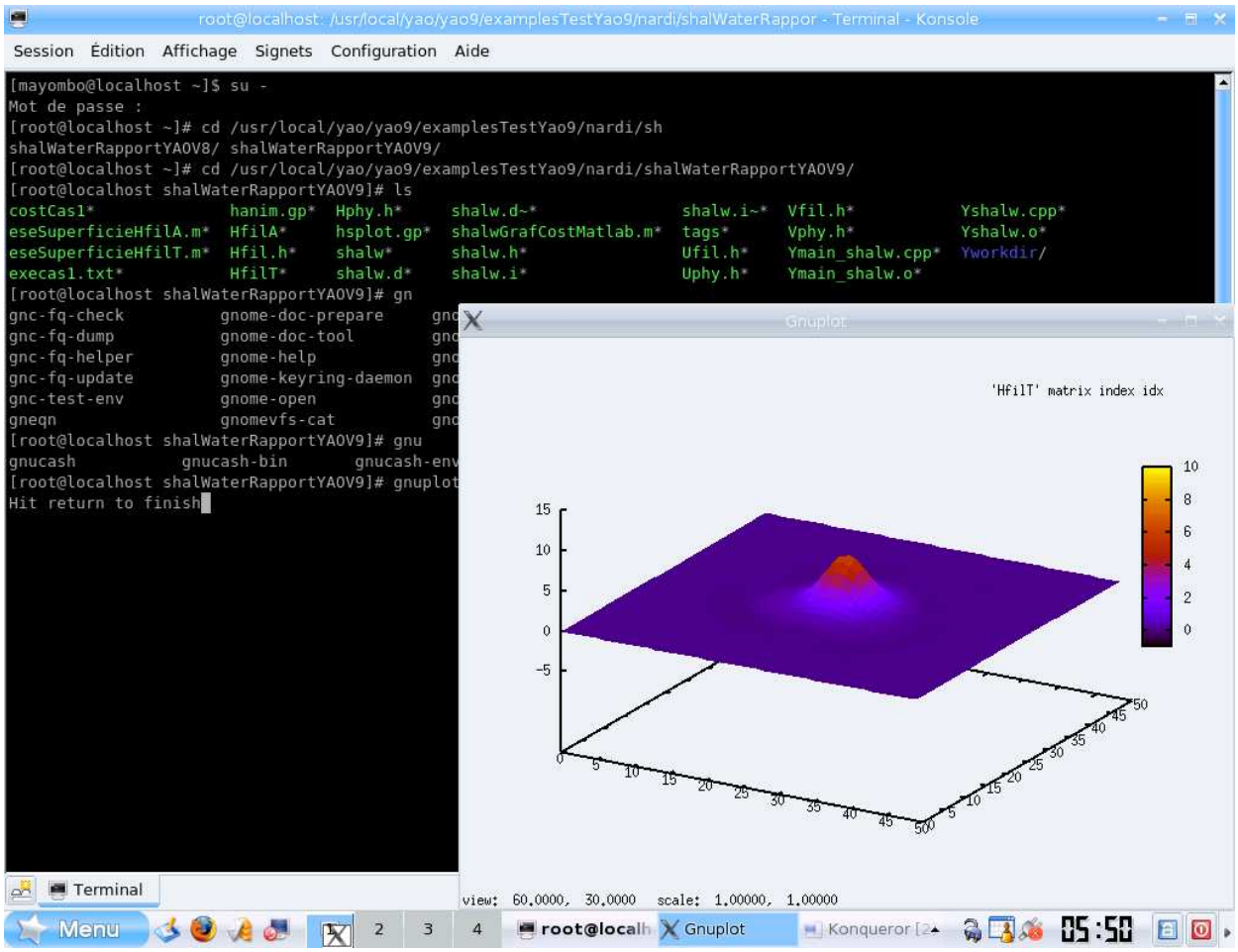


Figure a.26 : lacement avec la commande shalw.

REMARQUE : Nous constatons que la courbe shalw du TP YaoRapport.pdf s'aplatie à la fin de l'exécution de la séquence, alors que la courbe shalw dans le répertoire : /usr/local/yao/yao9/examplesTestYao9/nardi/shalWaterRapportYAOV9/ à un bout étroit lors de la fin de la séquence. Ceux-ci, peut trouver une explication dans le contenue des fichiers des modules qui sont programmés différemment.