UNIVERSITÉ DU QUÉBEC

MÉMOIRE PRÉSENTÉ À
L'UNIVERSITÉ DU QUÉBEC À TROIS-RIVIÈRES

COMME EXIGENCE PARTIELLE
DE LA MAÎTRISE EN MATHÉMATIQUES ET INFORMATIQUE
APPLIQUÉES

PAR
MARC-ANDRÉ ROCHETTE

VERS UNE INGÉNIERIE FLEXIBLE POUR LE TRAITEMENT DE L'INFORMATION

DÉCEMBRE 2008

REMERCIEMENTS

Je remercie particulièrement mon papa Mario Rochette et ma Maman Claire Rochette ainsi que ma sœur Mélanie Rochette pour m'avoir soutenu et encourager tout au long de ce gigantesque travail de maîtrise. Ils ont toujours été présents dans les moments difficiles. Je veux aussi remercier mon petit chien Maggy ☺. Je veux aussi remercier le petit bébé à ma sœur, Loïc, pour m'avoir changé les idées à la fin de ma maîtrise.

Je veux remercier toute ma famille du côté des Hébert et des Rochette pour m'avoir épaulé durant ce travail.

Je remercie tous mes amis(es) Caroline Rabouin, Annie Beauregard, Mylène Leclerc, Émilie Germain, Caroline Savard, ainsi que tous mes autres amis(es) pour leurs encouragements durant ce travail.

Je veux faire un petit remerciement à Marie Vaillancourt et Gilles Vaillancourt pour m'avoir donné des conseils.

Je veux aussi grandement remercier le Professeur Ismaïl Biskri pour avoir accepté de me diriger tout au long de ma maîtrise et de m'avoir encouragé.

Je tiens aussi à remercier le Professeur François Meunier ainsi que le Professeur Fathallah Nouboud pour avoir accepté de lire et d'évaluer mon mémoire.

Je veux aussi remercier le professeur Jean-Guy Meunier du département de Philosophie de l'Université du Québec à Montréal pour ses encouragements et conseils.

Je dédie ce mémoire à ma tante Micheline Rochette atteinte d'un grave cancer.

TABLE DES MATIÈRES

LISTE DES TABLEAUX

LISTE DES FIGURES

INTRODUCTION

L'informatique est rendue à un tournant où toutes les façons de faire pour la création des logiciels ne répondent plus à la demande. Les logiciels deviennent de plus en plus gros. Ils sont plus difficiles à maintenir ou à faire évoluer. Dans son article « Software Out There », JOHN MARKOFF dit clairement que des logiciels tels que Microsoft Word sont dépassés [Annexe D « Software Out There »]. Des armées de programmeur sont requises. Cette époque est révolue. Il faut commencer à penser les programmes non comme des grosses boîtes remplies dont les personnes n'utilisent que 10%, mais en termes de pièces détachées. Avec l'avènement d'Internet, la notion de pièces détachées est bien plus présente. JOHN MARKOFF parle plus de Légos [Annexe D « Software Out There »]. Ces petites pièces de notre enfance que nous emboitions pour construire ce dont nous voulons. Les Légos pratiquaient grandement notre créativité. Aujourd'hui, la créativité est plus laissée au programmeur qu'à l'utilisateur.

Comment peut-on arriver à faire travailler la créativité de l'utilisateur à la place du programmeur? Ce qui est sûr, c'est que l'utilisateur ne pourra jamais remplacer le programmeur. Il pourrait lui venir en aide et le compléter. Comment le compléter? Le programmeur construit différents blocs d'un système et l'utilisateur peut choisir entre les différents blocs ceux qui lui seront utiles. Il ne lui restera plus qu'à les intégrer à son système personnalisé. Bien sûr, rien n'empêcherait de faire des regroupements de blocs et de les vendre aux clients.

Google, comme JOHN MARKOFF le dit, a débuté dans cette idée [Annexe D « Software Out There »]. La compagnie offre, dans son inventaire, des dizaines de différents blocs. La gamme de produits offerts va du simple traitement de texte, en passant par une feuille de calcul, en faisant un détour par les blocs notes, de la recherche dans les livres, un agenda, etc. Les gens choisissent ce dont ils ont besoin et l'utilisent. Personne ne les oblige à acheter des blocs qu'ils n'utiliseront peut-être jamais dans la vie du logiciel.

Certaines personnes vont probablement commencer à penser à l'argent. Comment faire de l'argent avec ce type d'architecture en bloc? Pourquoi ne pas simplement vendre les blocs. On pourrait vendre les sources des blocs si les gens désirent modifier certaines choses. On peut s'inspirer du modèle de l'open-source.

Sûrement qu'un petit sourire vous vient au lèvre et que vous pensez automatiquement à l'Orienté Objet. L'Orienté Objet n'est en fait qu'une partie de la solution. Nous devons arriver à trouver un moyen de pouvoir offrir la possibilité à un utilisateur de pouvoir créer son application sur mesure sans toutefois l'obliger à apprendre la programmation.

Bien que l'Orienté Objet soit un des paradigmes les plus en vogue actuellement, il serait peut être intéressant de regarder ce qu'il se fait d'autre. Existe-t-il vraiment le paradigme parfait, je ne le crois pas. Alors quelle pourrait être la solution? Est-ce qu'utiliser plusieurs paradigmes combinés seraient une meilleure solution? Des anciens comme des nouveaux.

L'Orienté Objet est bien entendu important. Pourquoi on ne regarderait pas du côté du paradigme fonctionnel, du Multi-Agent et pourquoi ne pas regarder du côté des paradigmes descriptifs. Nous ne pouvons pas simplement donner à l'utilisateur inexpérimenté l'Orienté Objet et lui dire de connecter des classes entre elles. Nous devons arriver à trouver un moyen de donner à l'utilisateur la possibilité de connecter des blocs entre eux sans que ce dernier ait à acquérir des notions de programmations.

Au premier chapitre, nous allons voir les différents paradigmes que nous pourrions utiliser pour concevoir notre solution. Dans le second chapitre, nous verrons les différentes plates-formes présentent sur le marché. Nous allons voir si ces plates-formes pourraient nous servir pour le développement de notre plate-forme. Dans le troisième chapitre, nous entrons dans la théorie de notre approche. Nous allons voir la logique combinatoire qui est au centre de la solution, nous allons aussi voir

comment on construit une chaîne de traitements. Nous verrons aussi les différents aspects théoriques comme l'échange des messages entre les modules et ce qu'un module représente dans notre solution. Dans le dernier chapitre, nous allons faire une présentation de l'application de notre solution. Nous verrons en pratique ce que nous avons présenté au troisième chapitre. Nous ferons quelques comparaisons pour démontrer les preuves que la théorie fonctionne en pratique.

CHAPITRE 1
PARADIGMES DE PROGRAMMATION : UNE SYNTHÈSE

1 Les paradigmes

Depuis le début de l'ère de l'informatique, il nous a fallu des façons de penser. Dès le départ, lors de la création de l'ordinateur, les langages étaient présents. Un langage est une façon d'implanter des règles. Un programme est une suite d'instructions. Ces instructions doivent respecter des lignes directrices.

Les paradigmes de programmation que nous allons survoler dans ce chapitre vont nous être d'une grande aide pour la suite du mémoire. Ils vont nous donner des balises pour concevoir les différentes couches du projet. Pour construire notre solution, nous devons nous appuyer sur des bases existantes. Nous n'allons pas réinventer toute la roue de l'informatique. Pour cela, nous allons utiliser les paradigmes pour construire notre solution. Dans ce chapitre nous allons voir plusieurs paradigmes. Nous utiliserons la grande majorité des paradigmes. Le seul que nous n'utiliserons pas est le paradigme Aspect. Nous verrons pourquoi lorsque nous en parlerons. Nous allons devoir transposer les paradigmes de la théorie à la pratique par la suite.

Le paradigme de programmation nous énonce ces lignes directrices. Ces règles du paradigme nous dictent la façon dont les solutions doivent être formulées dans un langage de programmation implémentant ce paradigme. Les langages de programmation, quant à eux, peuvent implémenter un ou plusieurs paradigmes [Appleby 1991].

Les paradigmes sont divisés essentiellement en deux familles. Il y a une multitude de paradigmes mais on se limite à ces 2 familles. Chacune des deux familles à une multitude de paradigmes qui en découlent. Ces deux grands paradigmes sont : déclaratif et impératif. À partir de ces deux paradigmes, nous dérivons les autres. Il

se peut que certains paradigmes empruntent des concepts aux autres. La figure 1 présente les paradigmes que nous étudierons.

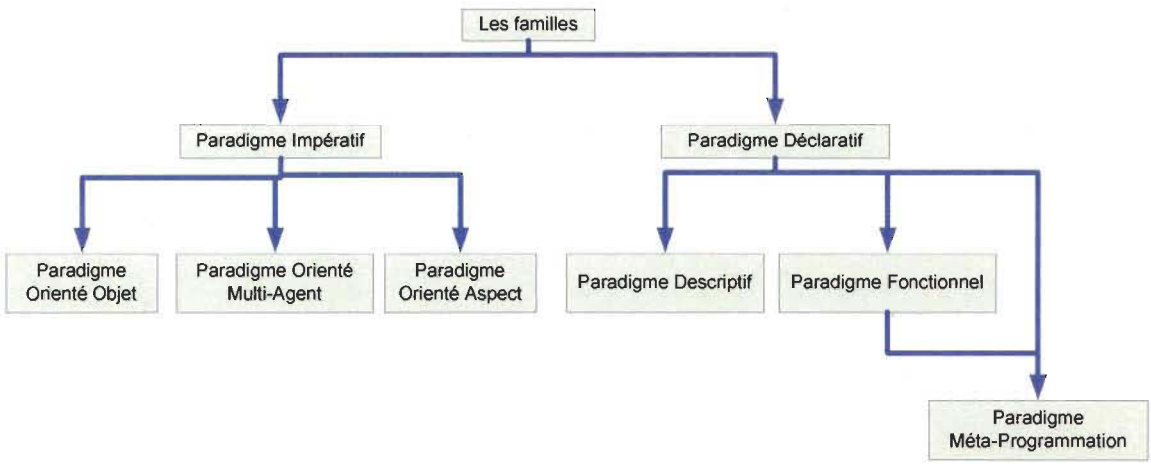


Figure 1 : Carte des familles de paradigmes.

1.1 Le Paradigme Impératif

Ce paradigme est supporté par la grande majorité des langages de programmation qui sont utilisés de nos jours. Ce paradigme est basé sur le modèle de la machine de Von Neumann (1903-1957). La figure 2 présente cette machine.

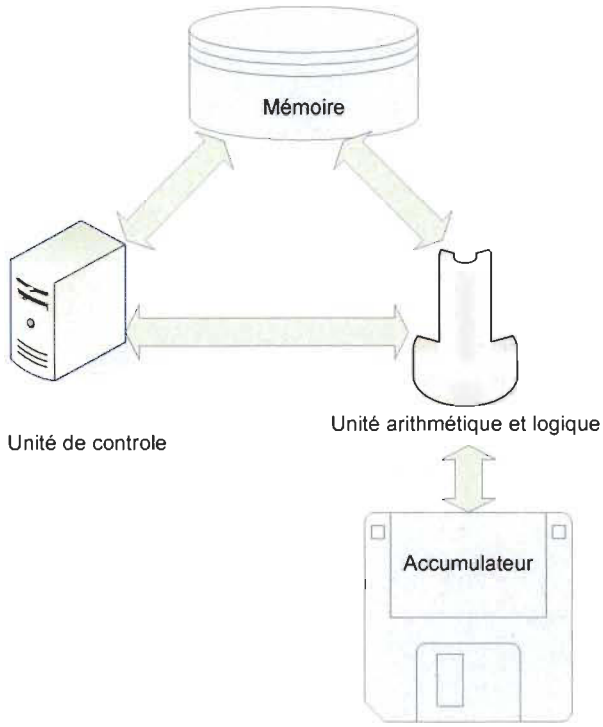


Figure 2 : Schématisation de l'architecture de Von Neumann.

Nous avons la mémoire qui contient le programme ainsi que ses données. Il y a l'unité de contrôle qui est chargé de procéder au séquençage des instructions. Nous avons l'unité arithmétique et logique (UAL) qui effectue les opérations et qui permet d'interagir avec l'extérieur grâce aux entrées et sorties. Cela introduit principalement la notion d'affectation ou d'effet de bord. Ce qui revient à dire que la mémoire est une zone temporaire qui est effacée chaque fois qu'on affecte une nouvelle donnée.

Une autre des caractéristiques est le séquençage des instructions qui se fait par l'intermédiaire de structures de contrôle. Les principales instructions de contrôle dans les langages sont : l'assignation, le bouclage, le branchement conditionnel ainsi que le branchement sans condition [McMillan et Al. 1992]. Le langage de programmation qui se rapproche le plus de la machine de Von Neumann est l'Assembleur (langage de bas niveau). On dit de bas niveau parce qu'il permet d'interagir directement avec la machine. Le langage Assembleur est de nos jours, très abandonné à cause de la lourdeur pour écrire de simples programmes. Pour résoudre ce problème, d'autres

langages ont été construits pour se rapprocher de plus en plus de l'être humain. Avant de parler des langages, voyons les paradigmes qui découlent de celui-ci.

1.1.1 Paradigme Orienté Objet

Ce paradigme a vu le jour avec le langage *SIMULA* dans les années 1960 [Ravi 1995]. Il était inspiré de la programmation modulaire. Ce langage n'a jamais eu de grand rebond dans le monde universitaire. Il était conçu pour des applications industrielles. C'est ce langage qui donna naissance à l'Orienté Objet et qui donna naissance à un très grand groupe de recherche appelé *Génie Logiciel*. Ce groupe commença à penser à des façons de faire de la programmation comme nous la connaissons aujourd'hui. Ce groupe continue toujours d'apporter de nouvelles stratégies pour la conception. L'orienté objet est assez vieux. Il a été abandonné un certain temps et a été repris plus tard pour résoudre des problèmes pour la conception des logicielles. Les principes de l'Orienté Objet ne s'appliquent pas seulement aux informaticiens. Nous pouvons dire que le paradigme Orienté Objet n'est aucunement l'invention des informaticiens. L'homme a souvent fait la résolution des problèmes de cette manière. Nous pouvons, par exemple, l'appliquer à l'industrie automobile. Une automobile est constituée d'objets. Ils sont assemblés de façon logique pour produire un véhicule qui se déplace. Il peut y avoir bien d'autres domaines dans lesquelles ce paradigme peut être appliqué. Nous avons juste à regarder le monde qui nous entoure. Ce paradigme est le plus populaire aujourd'hui.

1.1.1.1 Concept de base de l'Orienté Objet

En réalité, qu'est ce qu'un objet? Un objet est une représentation d'une entité du monde réel qui nous entoure : une maison, un livre, un chien, un avion,... Les objets peuvent aussi représenter nos idées. Ils sont définis par des classes. Un objet est une instance d'une classe qui est unique. La classe a été introduite afin de réduire la complexité des programmes. Avant, quand vous implémentiez un logiciel pour la gestion animal, vous deviez décrire chacun des animaux un à un. Vous vous rendiez

compte que les caractéristiques des animaux se répétaient. Un chien à un nombre de pattes tout comme l'araignée. Le chien émet un son spécial tout comme un chat. Cela veut dire que les deux animaux ont des caractéristiques communes.

Dans ces approches, nous parlons constamment de classes et d'objets. Ces deux entités sont différentes. Une classe est une description formelle des données. Elle contient les variables et les fonctions, elle est décrite dans un langage de programmation comme Java, C++, C#... Ces langages sont compréhensibles par l'humain. La classe peut être décrite dans un langage de modélisation de données comme l'UML *Unified Modeling Language* ou *Langage de modélisation objet unifié*. L'objet est une instance de la classe au moment du déroulement du programme. L'objet est une suite d'instructions non compréhensible par un humain. Les classes peuvent être lues par un humain mais pas les objets. L'objet ne peut être lu que par l'ordinateur. L'objet est représenté en langage machine lors de son exécution. Une classe contient deux ensembles d'entités bien à elle. Soit les variables et les méthodes. Une variable contient des données et les méthodes sont les actions qui sont faites. La figure 3 présente un schéma *UML*.

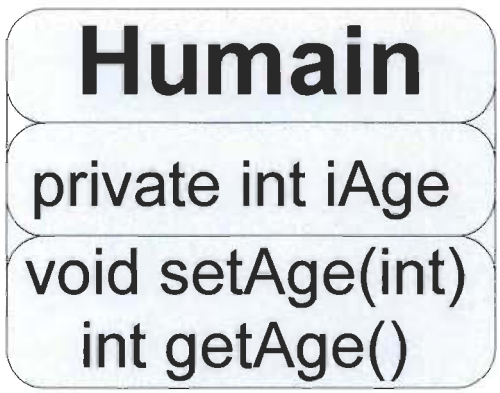


Figure 3 : Représentation d'une classe dans un schéma UML.

La classe de la figure 3 indique le nom de la classe qui est *Humain*. Il y a une variable nommée *iAge* qui va contenir l'âge de la personne. Nous avons une première méthode qui se nomme *getAge* qui va retourner l'âge de la personne. Une seconde méthode se nommant *setAge*. Elle a comme action d'attribuer l'âge à la personne.

1.1.1.2 Les propriétés fondamentales de l'Orienté Objet

Pour que le langage objet existe, ce dernier doit contenir certaines propriétés de base. Ces propriétés fondamentales sont : l'héritage, l'encapsulation ainsi que le polymorphisme. Si vous retrouvez ces trois concepts dans un langage, c'est que ce langage implémente l'orienté objet.

1.1.1.2.1 L'héritage

Cette caractéristique repose sur le principe de généralisation spécialisation. Une classe peut hériter d'une autre classe. Cela veut dire que la classe *B* hérite de la classe *A* et elle détiendra tous les mêmes comportements que la classe *A*. La classe *B* sera une spécialisation de la classe *A*. Elle pourra redéfinir des membres de la classe *A*.

L'héritage permet de construire des *graphes d'héritages*. Les classes les plus hautes vont être la généralisation. Plus on va descendre dans l'arbre et plus on va avoir de la spécialisation. La figure 4 présente un graphe d'héritage pour un parc de véhicules.

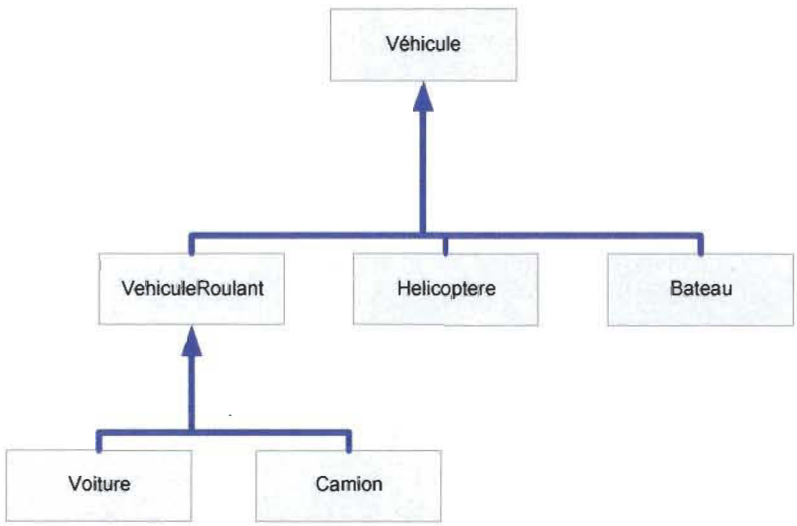


Figure 4 : Exemple d'un graphe d'héritage pour un parc de véhicules.

Dans cet exemple, nous avons notre classe de base qui généralise ce qui est un *véhicule*. Par la suite, nous allons plus vers la spécialisation. Nous avons l'*hélicoptère* et le *bateau* qui héritent uniquement de la classe *Véhicule*. La classe du *VéhiculeRoulant* donne une seconde spécialisation au *Véhicule*. Il donne comme caractéristique que les véhicules sont roulants. Finalement, nous avons la classe *Voiture* et *Camion* qui héritent de *VéhiculeRoulant*. Comme nous pouvons le constater, il peut y avoir plusieurs niveaux d'héritages.

1.1.1.2.2 L'encapsulation

Une classe est composée de variables et de méthodes. Un programmeur qui utiliserait notre classe peut voir toutes ses propriétés. S'il peut les voir, généralement, il pourra les modifier. S'il désire modifier une variable, cela pourrait avoir une grave conséquence sur la cohérence de la classe. La classe pourrait perdre sa raison d'être à la limite. Pour résoudre ce problème, le paradigme nous propose trois clauses importantes. Soit la clause privée, publique ainsi que protégée.

1.1.1.2.2.1 Privé (Private)

Cette clause va permette au programmeur de cacher les variables ou les méthodes à d'autres programmeurs qui pourraient utiliser la classe. Elle est plus utilisée pour cacher les variables. C'est même une façon de faire qui nous est enseignée. Pour accéder à ces variables cachées, nous utilisons des méthodes que nous appelons *get* (prendre) et *set* (définir). Les membres privés ne peuvent pas être vu quand la classe est héritée.

1.1.1.2.2.2 Publique (Public)

La notion de publique permet de rendre les propriétés visibles de la classe. Il est très déconseillé de rendre les variables publiques. Il est préférable de les transférer dans la

clause privée pour une plus grande protection et d'y accéder par des méthodes. Les membres de la section publique sont visibles quand la classe est héritée.

1.1.1.2.2.3 Protégé (Protected)

La notion n'existe pas dans tous les langages. Elle permet de cacher les membres d'une classe à la personne qui va l'utiliser. Par contre, lors de l'héritage, elle permet de voir les membres qui en font partie. On pourrait prendre cette clause comme un demi-moyen pour la protection des données.

1.1.1.2.3 Polymorphisme

Ce mécanisme permet, pour une même fonction nommée, plusieurs types de paramètres différents. Cette propriété de l'orienté objet est très pratique pour pouvoir opérer des traitements d'un même nom pour différents types de données.

Le fait d'utiliser ce principe évite au programmeur d'avoir à programmer un type très spécifique. Il peut rester dans la généralité. On veut définir, par exemple, une fonction qui fait de la conversion d'un type vers un autre. Nous savons qu'il existe une multitude de types. Avec le processus de polymorphe, à l'étape de la compilation, le compilateur fera le bon choix de méthodes pour le bon type de données. Voyons un exemple simple de ce concept.

Prenons comme exemple la conversion d'un type vers un autre. Vous voulez convertir un nombre entier ou réel en chaîne de caractères, vous n'avez qu'à saisir l'instruction *toString(valeur)*. Lors de la compilation, le compilateur fera le choix de la bonne méthode pour le bon type. On peut aussi appliquer le processus de polymorphisme aux opérations mathématiques.

1.1.1.3 Comment procéder sur papier

Pour soutenir le développement, des approches formelles sont nées. Comme bien des méthodes, elles sont venues après le langage de programmation. Pour l'Orienté Objet, plusieurs méthodes existent. La principale méthode se nomme *UML (Unified Modeling Language)* [Sommerville 2004]. C'est une méthode de modélisation des données qui est issue des travaux de Grady, Booch et James Rumbaugh [Sommerville 2004]. Il existe d'autres méthodes telles que *Merise* qui est l'ancêtre de l'UML. UML est composé d'une multitude d'outils permettant de représenter graphiquement nos besoins. UML est composé d'une série de 13 diagrammes [Larman 2002] pour définir des modèles. La figure 5 présente un petit schéma UML.

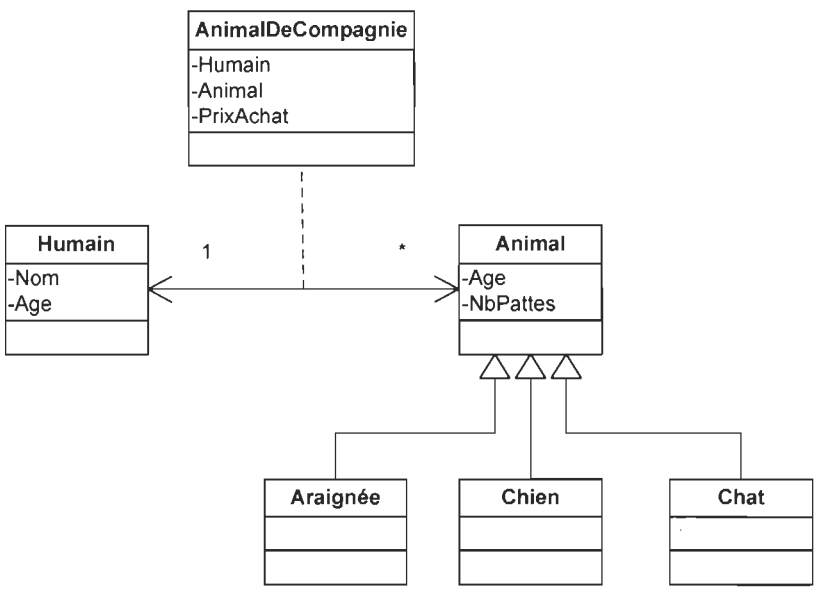


Figure 5 : Exemple d'un schéma UML.

1.1.1.4 Une discipline à part entière

Afin de bien supporter tout le développement, le génie logiciel est né. Il a proposé des approches afin de s'attaquer à la construction de gros logiciels. C'est une discipline à part entière. Le Génie Logicielle propose de multitudes d'approches, d'outils, de modèles conceptions,... L'explication de toute l'approche de cette

discipline dépasse largement le cadre de ce rapport. Plusieurs livres expliquent très bien cette discipline.

1.1.1.5 L'implémentation de ce paradigme

L'implémentation de ce paradigme n'est pas seulement réservée au langage purement objet. Ce paradigme peut être développé dans presque tous les langages. Si on conçoit cela dans un langage fonctionnel, comment allons-nous faire exister les concepts du paradigme Orienté Objet? La grosse différence entre les langages fonctionnels et Orientés Objets est que les langages Orientés Objets nous obligent à suivre les règles établies sur papier par le paradigme. On ne peut pas diverger de cela sinon cela ne passe pas à la compilation. Si vous prenez un langage comme l'Assembleur, vous devez faire respecter vous-même les concepts de ce paradigme. Cela vous demanderait sans doute une plus grande rigueur dans votre programmation afin de tout respecter.

N'oubliez pas que dans l'univers informatique, tous les programmes sont traduits en langage machine. Que nous soyons en Assembleur, Java, C++, C#, Pascal,..., vous devez vous rappeler que tous ces langages sont traduits en code machine à la fin.

1.1.1.6 Les langages de programmation

Tous les langages de programmation ne sont pas Objet. Une grande partie des langages ne le sont pas. Ce n'est pas parce qu'un langage n'est pas Orienté Objet qu'on ne peut pas le transformer. En transformation, je parle de changer notre façon de concevoir avec ce langage. Par exemple, le langage C n'est pas objet. Il est plutôt modulaire avec ses différents fichiers. Pourquoi ne pas prendre chacun des fichiers d'un programme et les penser comme étant des classes. Nous allons décrire quelques langages très utilisés dans le monde de la programmation. Certains sont purement Objet et d'autres non.

Bien que C# soit un langage entièrement Objet, une note reste à préciser. C#, C++.Net, ASP.NET, VB.NET (http://msdn.com) et toute la gamme des langages du .Net ne sont pas tout à fait des langages objets à 100%. Quand le programmeur conçoit un programme, il utilise l'orienté objet. Avant de passer au processus de la compilation machine, il doit passer par la transformation dans un langage intermédiaire qui se nomme *IL* ou *Intermediate Language* (*Langage Intermédiaire*). Java (http://java.sun.com) n'échappe pas à cela non plus. C'est un langage modulaire où l'objet disparaît complètement. Ce langage offre la possibilité à un programmeur de redéfinir lui-même son propre langage en utilisant le compilateur du Framework de Microsoft.

1.1.2 Paradigme Orienté Multi-Agent

Ce paradigme reprend l'idée de l'orienté objet. En effet, le paradigme agent reprend les caractéristiques de l'Orienté Objet. Il ajoute des couches d'intelligence et d'indépendance aux objets. Un objet va maintenant prendre le nom d'agent. Pourquoi ne pas distribuer les charges dans plusieurs programmes au lieu d'un seul. Diviser pour régner peut être un bon slogan pour ce paradigme. Ce paradigme est encore dans le domaine de la recherche. Il est très récent et utilisé presque uniquement dans le domaine universitaire.

1.1.2.1 Une définition

Le paradigme agent est un bon outil pour arriver à faire adéquatement la communication entre différents petits programmes. Pour commencer, essayons de faire une bonne définition de ce qu'est un agent. Dans l'univers des agents, les chercheurs ne s'entendent pas sur une définition unique. Nous pouvons cependant tirer certaines similitudes entre toutes ces définitions. Un agent peut être une entité autonome. Il doit évoluer dans un environnement qu'il reconnaît. Il doit être capable d'évoluer en société avec d'autres agents. Un agent doit aussi avoir une tâche qui est bien à lui. Il doit aussi avoir une tolérance aux fautes. Il ne doit pas planter parce

qu'il reçoit une communication erronée. Il devrait avoir un certain degré d'imprévisibilité. Il est important de dire qu'un agent doit avoir à exécuter qu'une seule tâche et non plusieurs.

Nous parlons de paradigme Agent. Ce paradigme admet une communauté d'agents. Ils vivent ensemble dans un même environnement. Dans bien des cas, un système sera considéré comme Multi-Agent s'il a des propriétés. Cette tâche, de définir les propriétés d'un tel système, revient au concepteur de l'environnement. Ce monde est encore très récent. La partie la plus développée dans ce paradigme est celle des protocoles de communication. Ces protocoles sont inspirés de ceux des réseaux informatiques.

Chaque agent observe le monde extérieur (l'environnement où il évolue) selon ses conditions. Il doit effectuer des actions dans l'environnement. On peut représenter cela par une sorte de vie en communauté qui est transposée dans le monde l'informatique.

1.1.2.2 Le système Multi-Agent

Il est important de dire qu'un agent peut évoluer seul. La plupart du temps, il va évoluer en groupe afin d'avoir un but commun. Un système Multi-Agent est comme une vraie société. Tous les membres de l'environnement évoluent vers un but commun. Cela représente bien notre société où tout le monde essaie de travailler en relation afin d'arriver à une solution. Nous apportons cette précision pour dire qu'un agent peut évoluer seul ou en groupe.

1.1.2.2.1 Comportement social

Les agents, qui évoluent dans un environnement social, doivent être capables de faire des opérations assez importantes :

1.1.2.2.1.1 Coopérer

Les agents doivent être capables de travailler ensemble dans l'environnement. C'est la base d'une société.

1.1.2.2.1.2 Compétitionner

Il doit être capable de prendre ses ressources et de les maximiser. Il doit toujours essayer d'être un peu meilleur que ses copains.

1.1.2.2.1.3 Interactivité

Il doit être capable de parler à ses confrères. Il s'agit d'avoir un échange entre un émetteur et un récepteur.

1.1.2.2.1.4 Adaptation

Il doit être en mesure de s'adapter dans les environnements tout en communiquant avec les autres. On voit mal un agent bouder parce qu'il n'aime pas un autre agent dans l'environnement.

1.1.2.2.1.5 Sociabilité

Un agent doit être capable de parler avec les autres. Il ne doit pas les bouder.

1.1.2.3 Les types d'agents

Nous distinguons deux grands types d'agents. Ces deux types d'agents découlent des deux grandes écoles de pensée. Nous avons les agents réactifs et cognitifs. Il y a aussi un dernier agent qui est moins connu mais qui est tout aussi important pour la bonne coopération, les réflexes.

1.1.2.3.1 Les agents réactifs

Comme son nom l'indique, cet agent a une réaction face à une situation. Il y a deux types d'agents réactifs. Soit les pulsionnels et les tropiques [Gerhard 1999]. Les

pulsionnels observent un évènement et réagissent en fonction de l'évènement. Les tropiques réagissent face à des actions qui se passent dans l'environnement. On peut dire que les tropiques sont les concierges de l'environnement.

1.1.2.3.2 Les agents cognitifs

Ils ont une tâche bien précise à accomplir. Ils peuvent servir d'agents à interroger. Si l'agent connait bien son environnement, il peut se servir de sa base de connaissances pour donner des informations aux autres agents. Les autres agents de l'environnement l'interrogeront pour avoir des réponses à leurs questions.

1.1.2.3.3 Les agents réflexes

Cet agent agit comme une fonction dans l'environnement. Il doit faire une réaction immédiatement face à une action déclenchée. Il ne peut pas l'ignorer ou la refuser. On pourrait dire qu'il est une sorte d'œil magique et lorsque quelqu'un passe devant, il déclenche un processus.

Comme vous pouvez le remarquer, vous devez voir qu'un agent a une certaine ressemblance avec l'humain. Ils doivent avoir certains comportements entre eux afin d'arriver à un but commun. Une grosse différence entre les agents et les humains est que les agents n'ont aucuns sentiments. Un humain va modifier sa façon d'agir selon ce qu'il ressent. L'agent va toujours agir de la même façon sans traitement de faveur.

1.1.2.4 Rôle d'un agent

Un agent évolue dans un groupe. Un seul agent ne pourrait pas effectuer un travail complet. L'agent effectue une tâche unique dans l'environnement. L'agent va agir à une action. Pour chaque action, il produira une réaction. Il s'agit de la troisième loi de Newton. L'agent va percevoir une chose qui est anormal dans son environnement

et il va réagir. Par anormal, on entend que cela va être un évènement. La figure 6 résume bien le rôle de l'agent.

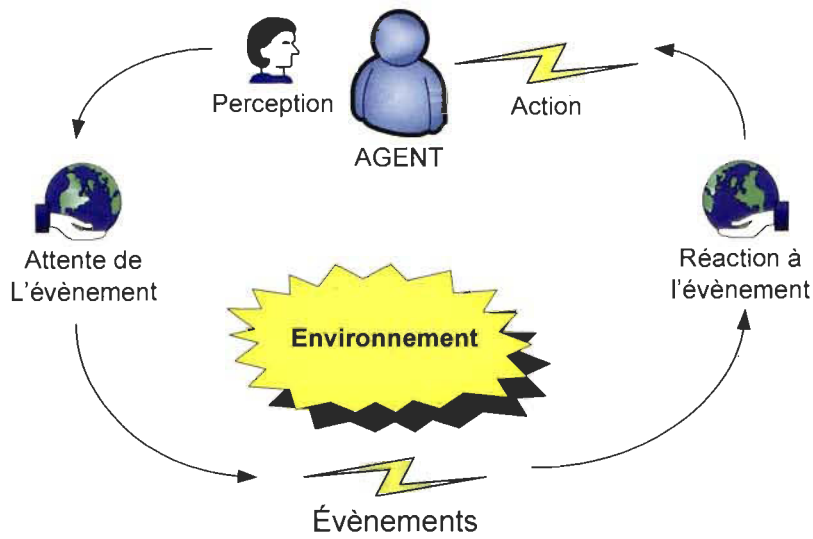


Figure 6 : Modèle d'un agent dans le système.

Un agent doit être intelligent. Il évolue dans un environnement. Il doit être en mesure d'interagir avec cet environnement.

1.1.2.5 Vous avez dit Orienté Objet

Il est important de dire que le paradigme agent n'est pas une évolution du paradigme Orienté Objet. Ce sont deux entités complètement distinctes. Un objet fait les choses sans rien dire. L'objet ne raisonne pas. Il exécute une tâche qui lui a été assigné à un moment bien précis. Un objet n'est pas autonome. Il a un état qui le définit.

L'agent va choisir s'il veut faire les choses. Normalement, il va exécuter une action parce qu'il a raisonné face à une situation. Les agents, contrairement aux objets, sont flexibles. Ils ont une certaine marge de manœuvre. Le paradigme agent est plus facile à mettre à jour. Nous n'avons pas à retoucher à tout le système. Si nous retirons un agent, la conséquence dans l'environnement sera que l'action que l'agent effectue ne

sera plus disponible sur le système. Nous avons qu'à prendre l'agent en défaut et l'enlever. On peut simplement rajouter un agent au système s'il en manque un. Nous n'avons pas à recompiler tout l'environnement parce que les agents sont tous indépendant dans l'environnement.

1.1.2.6 Les agents dans les jeux

Les agents se retrouvent beaucoup dans les jeux. Que viennent faire les agents dans les jeux? Les types de jeux où on les retrouve sont ceux de stratégies. Ces jeux sont généralement conçus comme une organisation sociale où tous les éléments du jeu doivent travailler ensemble pour arriver à gagner la partie.

Un jeu médiéval par exemple. Les acteurs du jeu, qui sont les personnages, doivent coopérer entre eux pour réussir à vaincre l'adversaire. Si nous voulons construire un chevalier que faut t-il? Il faut que nous ayons en premier des matières premières pour fabriquer l'épée et l'armure. Il nous faut des mineurs et une fonderie. Quand nous avons fait fondre les éléments, il faut envoyer le tout à l'atelier pour fabriquer l'épée et l'armure. Il ne faut pas oublier de nourrir tous les personnages. Il nous faut un fermier pour cultiver et entretenir les animaux. En parlant d'animaux, il ne faut pas oublier que notre chevalier a besoin d'un cheval. Il faut construire une écurie. Le fermier va produire du blé pour le cheval et pour le boulanger.

C'était un petit exemple pour une petite partie de la chaîne d'étapes pour fabriquer un chevalier. Comme nous pouvons le voir, tous ces éléments doivent communiquer entre eux pour arriver à un but commun. Chaque personnage a seulement une fonction dans l'univers du jeu. Un est fermier, un chevalier, un forgeron, un pêcheur, ...

Le paradigme agent n'est pas seulement et strictement fait pour construire des applications complexes. Comme nous venons de le voir, il peut être utilisé pour le divertissement.

1.1.2.7 Le couplage et les agents

Un agent, comparativement à l'Orienté Objet, n'est pas autant couplé. Normalement un agent va recevoir qu'une seule tâche à la fois. Dans certain cas, il peut avoir une file d'attente pour les tâches. Considérons qu'un agent n'effectue qu'une seule tâche. Il sera couplé qu'à l'agent qui va lui transmettre la tâche à un moment précis.

Il est important de faire une grande différence. La programmation d'un agent est différente de la programmation de l'environnement. À l'intérieur d'un agent, il peut y avoir autant de couplages que nous le désirons. La seule consigne est qu'il faut que l'agent accomplisse la tâche qui lui est affectées. À la limite, un agent pourrait être constitué lui-même d'agents pour accomplir sa tâche.

1.1.2.7 La communication

Quand nous évoluons dans une société, nous ne pouvons pas passer à côté de la communication. Nous utilisons ce moyen afin d'arriver à un but commun. Nous avons 4 grands questionnements : pourquoi, quand, avec qui et comment les agents communiquent-ils [Ferber 1997]?

1.1.2.7.1 Les raisons

1.1.2.7.1.1 Le pourquoi

Nous voulons que les agents communiquent entre eux afin qu'ils partagent des informations. Ils doivent arriver à accomplir un but commun. Ils doivent se parler entre eux. Ce comportement est le même que dans le monde des humains.

1.1.2.7.1.2 Le quand

L'agent va communiquer quand il en aura besoin. Si à un moment *t* l'agent à besoin de réponses, il enverra une requête à un autre agent.

1.1.2.7.1.3 Avec qui ils communiquent

Cela dépend beaucoup de qui connaît qui. Ils peuvent communiquer avec un ensemble ou avec un seul. Tout dépendant des connaissances qu'il a de ses confrères. On pourrait très bien imaginer fournir un bottin des agents avec leurs descriptions. Cela reviendrait à créer un bottin téléphonique d'humains.

1.1.2.7.2 Comment communiquer

Ils doivent communiquer en ayant tous un protocole commun. S'il n'y a pas de règles pour la communication, alors se sera un vrai chaos. Nous devons établir des règles et un protocole. Un protocole inventé pour ce type de communication est le KQML.

1.1.2.8 La communication entre les Agents (Modèle KQML)

Comment se déroule la communication entre les agents ? Les agents peuvent travailler en groupe. S'ils communiquent entre eux, ils doivent respecter un protocole. Le protocole qui est le plus utilisé est le KQML (Knowledge Query and Manipulation Language) (Langage de manipulation d'interrogation de la connaissance). Il est indépendant des protocoles de la communication qui existent sur le marché (TCP/IP, POP3, TELNET, etc.).

1.1.2.9 La programmation

Un agent peut être construit de différentes façons. Nous pouvons utiliser les vieilles techniques de programmation comme les nouvelles. Nous pouvons utiliser le

paradigme Orienté Objet pour la construction de nos agents. Il est possible d'utiliser un langage procédural. À cette étape, cela est du ressort du programmeur. Il doit définir un protocole de communication pour son petit univers. Un problème majeur est qu'il est difficile de faire coopérer une multitude de langages de programmation à l'intérieur d'un système agent. Pour cela, on peut utiliser l'architecture distribuée *CORBA (Common Objet Request Broker Architecture)* (Figure 7) qui permet à plusieurs langages de programmation de communiquer entre eux. Il n'est pas de tout repos que de concevoir un environnement avec de multiples langages de programmation qui coopèrent ensembles. On utilise un IDL (Interface de définition de Langage) pour permettre la communication entre les différents langages. La figure 7 présente bien l'architecture CORBA.

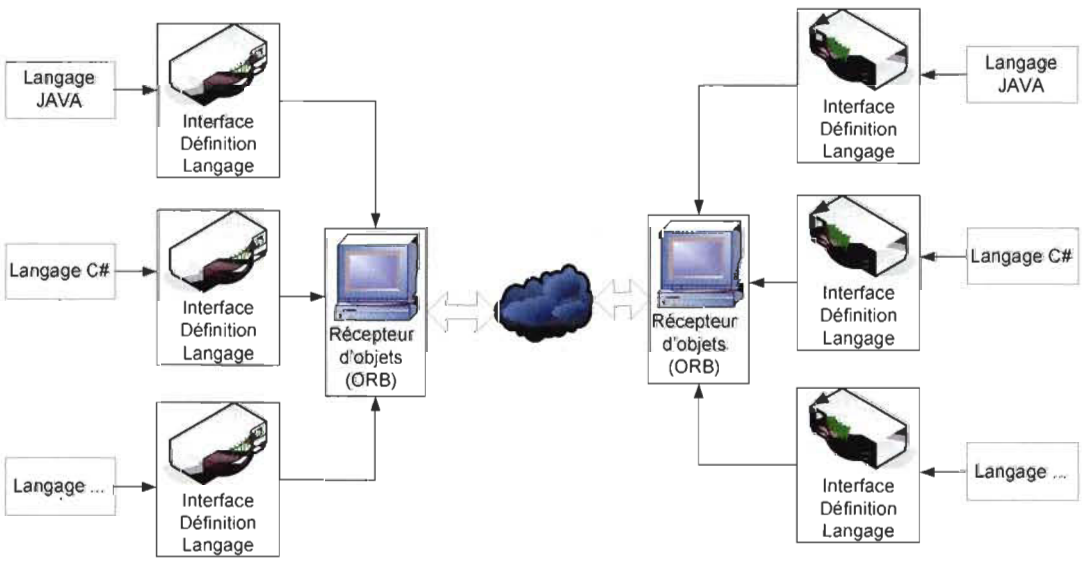


Figure 7 : Interface entre les différents langages dans les architectures distribuées.

1.1.2.10 La méta-programmation

La méta-programmation est un paradigme qui s'inspire de la programmation fonctionnelle. Un programme ayant une entrée et une sortie. Il peut évoluer seul dans un environnement. Il doit détenir certaines propriétés pour pouvoir évoluer dans

l'environnement. La grosse différence avec la Méta-Programmation est qu'avec le paradigme Agent les entités ont une certaine *autonomie* tandis que dans la Méta-Programmation, il doit y avoir un contrôleur qui supervise les programmes. Voici le schéma de la Méta-Programmation (figure 8) comparé à celui de l'environnement Multi-Agent (figure 9).

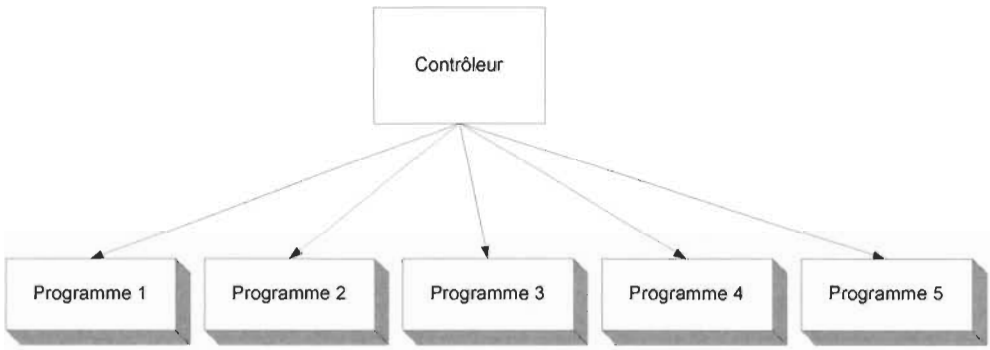


Figure 8 : Modèle général de la Méta-Programmation.

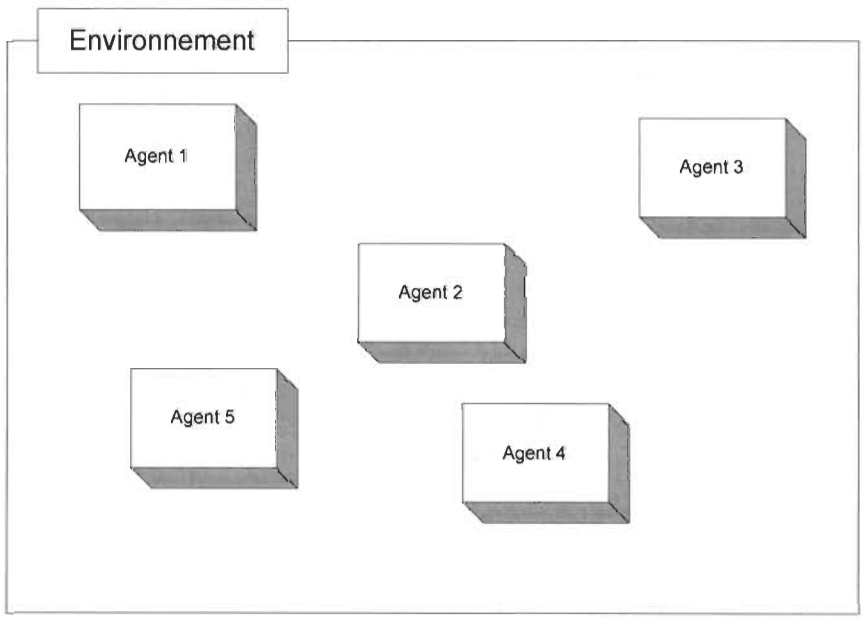


Figure 9 : Modèle général Multi-Agent.

1.1.2.11 Les environnements pour le développement des Systèmes Multis-Agents

Développer des agents n'est pas une tâche si simple. Aux premières apparences, les schémas semblent simples. Il en est tout autrement au moment de l'implantation. Pour faciliter la tâche des programmeurs, des environnements pour le développement sont développés pour venir en aide aux programmeurs. Nous allons décrire quelques environnements.

1.1.2.11.1 AgentBuilder

AgentBuilder est un environnement complet pour le développement des agents. Il est complètement Orienté Objet. Il utilise le langage KQML pour faire communiquer les agents entre eux.

http://www.limsi.fr/~jps/enseignement/examsma/2005/1_plateformes_2/MuhammadAfanSHAH/index.htm

1.1.2.11.2 Jade

Jade est un outil ouvert. Il n'impose pas de méthodologies. Il y a un annuaire qui enregistre un à un les agents qui sont créés. Cela permet de les répertorier plus facilement.

http://www.limsi.fr/~jps/enseignement/examsma/2005/1_plateformes_2/SAIDNA%20Said/Jade.htm

1.1.2.11.3 Zeus

Zeus est un environnement qui utilise la méthodologie appelée *Role Modeling* pour créer un système collaboratif. L'outil est complexe et demande une adaptation.

http://www.limsi.fr/~jps/enseignement/examsma/2005/1_plateformes_2/slimani/slimani.htm

1.1.3 Paradigme Orienté Aspect

Ce paradigme a été inventé pour pallier aux problèmes présents depuis longtemps dans le domaine de la programmation. Ce problème est la séparation des besoins fonctionnels des besoins non fonctionnels. Ce paradigme a son histoire bien à lui et n'a pas de liens avec aucuns autres langages. Il n'est pas un descendant de l'Orienté Objet qui l'a utilisé pour colmater des problèmes. On peut tout aussi bien l'utiliser en

Java qu'en langage C. Les concepts de base de ce paradigme on été jeté par Gregor Kiczales et son équipe dans le cadre du projet sur le Xerox Palo Alto Research Center [Gregor et Al. 1997].

1.1.3.1 Les bases

Une fonction utilisée dans ce paradigme est le tissage. Qu'entend-on par le tissage? Quand vous programmez, vous devez à la fois vous occuper des problèmes qui sont liés directement à votre étude ainsi que des problèmes techniques. Par exemple, le développement d'un système de gestion d'un parc pour les automobiles. Vous devez vous soucier des aspects tels que la sécurité de votre programme, la journalisation des évènements, etc. En programmation, tous ces besoins non fonctionnels devraient normalement se retrouver dans votre code des besoins fonctionnels. Soit au début, au milieu ou ailleurs dans votre code. Cela alourdit énormément votre code.

Pour résoudre ce problème, le paradigme Aspect introduit la notion d'aspect. Un aspect est un bout de code écrit par le programmeur. Il sera ajouté à votre code dans la phase de pré-compilation. Ce qui veut dire, qu'avant que votre code ne soit compilé, un prétraitement va introduire les aspects dans le code dans les endroits désirés. Ce procédé est appelé le tissage.

Pour permette l'ajout de l'aspect à l'endroit désiré dans votre code, nous utilisons des *jointpoints* ou des *points d'exécution*. Un *point d'exécution* permet de préciser les endroits dans le code où le prétraitement doit tisser les aspects. Le tissage veut dire que le code va être ajouté à des endroits précis.

1.1.3.2 Avantage

Les avantages de ce paradigme sont nombreux. Il permet au développeur d'avoir un code en s'occupant seulement de ses préoccupations sans se soucier des problèmes techniques qui peuvent être solutionnés après.

La réutilisation peut être accrue. Le code est pur, c'est-à-dire que le code ne contient pas de besoins non fonctionnels. Le code est plus facilement compréhensible.

1.1.3.3 Désavantage

Le paradigme Aspect n'est qu'un procédé que nous insérons par-dessus un autre paradigme. Il ne fait que colmater le problème.

Quand vient le temps du *débogage*, un problème qui est de taille se produit. Les aspects vont générer beaucoup de code automatique. Ce code ne sera pas nécessairement compréhensible lorsque des erreurs sont présentées.

Ce paradigme apporte des solutions aux langages existants. Il ne peut pas vivre seul. On appelle cela une béquille. Pour un projet simple, cette béquille peut très bien satisfaire les informaticiens lors des phases de tests et de *débogage*. Si nous prenons de gros projets, cela va être plus dur de tester et de *déboguer* le code. Ce paradigme est à utiliser avec modération.

1.2.1 Le Paradigme Descriptif (bases de données)

Ce paradigme permet de décrire des structures de données et de les manipuler. Aucune variable n'est admise dans ce paradigme. Les données seront toujours accessibles via les structures qui les entreposent sur un média. Le langage qui permet la création de la structure agira comme une fonction retournant une valeur. Cette valeur sera écrite sur le média. Le langage qui permet de faire l'interrogation de la structure agira comme une fonction en retournant les données faisant partie de l'interrogation. Quand on parle de ce paradigme, on pense immédiatement au langage pour les bases de données. Les structures de données comme le XML ainsi que le HTML en font aussi partie.

1.2.1.1 Paradigme XML

Ce paradigme permet de faire la description des structures de données. Une structure de données est une façon d'organiser les données afin de pouvoir les partager, de les utiliser ainsi que de les interroger. Nous allons nous concentrer sur le paradigme *SGML Standard Generalized Markup Language* ou *Langage normalisé de balisage généralisé* qui découle du paradigme descriptif.

Le paradigme *SGML* est né à la fin des années 1970 [Schwarte 1996]. Il fut développé par un chef de projet chez IBM Charles Goldfarb [McGrath 1997]. Après son départ d'IBM, il se concentra sur la deuxième génération du *SGML*. Le *SGML* est un langage à balisage permettant de structurer les données en oubliant le formatage comme le type de police, la grosseur du texte, etc. Il permet de séparer le texte dans une structure logique. Ce sont des structures telles que les titres, les chapitres, les paragraphes, les lignes, les illustrations, etc. Tout cela est balisé. Le langage est indépendant d'une application particulière. Si une personne désire lire un tel fichier, il doit détenir un lecteur de ce format. Cela normalise pour tout le monde. En 1990, il donne naissance au *HTML Hypertext Markup Language* ou *Langage de balisage d'hypertexte* [Koch et Al. 2000]. Le *HTML* a été introduit pour concevoir des pages Web sur Internet. On voulait pouvoir inclure des vidéos, des images, …

Bien que le *HTML* fût créé pour diffuser des informations, ce langage n'était pas assez structuré. Un nouveau projet débuta. Son objectif était de concevoir une façon claire et simple de structurer les données et de les afficher comme le *HTML*.

Ce langage fait partie des nouvelles générations de structures de données. C'est le *XML Extensible Markup Language* ou en français Langage de Balisage Extensible. Ce langage propose une façon ouverte d'échanger des informations entre les applications. Ce format vise à établir un standard. Les nouveaux langages

implémentent ce paradigme afin d'offrir un standard de communication qui était auparavant bien absent.

1.2.2 Paradigme fonctionnel

Nous devons remonter loin dans l'histoire de l'informatique pour retrouver les premiers langages fonctionnels. Le plus connu est le langage LISP qui est né en 1959 par *MacCarthy* [Sebesta 2001]. LISP est encore utilisé aujourd'hui. Il est installé sur les machines Linux. LISP, comme bien d'autres langages, a donné naissance à de petits enfants.

L'histoire du paradigme fonctionnel remonte bien avant les langages. Il faut remonter en 1893 pour retrouver chez les travaux de *Gottlob Frege* des traces du paradigme fonctionnel [Cardone et Al. 2006]. *Gottlob Frege* était un mathématicien, un logicien ainsi qu'un philosophe [Gyorgy 1988]. Ce qui le mit au monde officiellement, c'est la publication de l'article de *Moses Schönfinkel* sous l'article « On the Building Blocks of Mathematical Logic » [Cardone et Al. 2006].

Chaque fonction sera considérée comme un programme à part entière qui retourne une valeur. Ce paradigme n'admet pas non plus d'instructions. Il est purement mathématique. Ce modèle de programmation est peu utilisé dans l'industrie. Il est plus utilisé dans le cadre des universités et dans la recherche. Nous le retrouvons dans des langages de programmation Orienté Objet.

1.2.2.1 Les bases du paradigme

Le paradigme fonctionnel est inspiré des mathématiques. On est porté à parler de mathématiques quand on parle de ce paradigme. Il y a cependant quelques différences entre les deux concepts. Il est très facile d'implémenter des formules mathématiques dans ce paradigme. Les autres langages empruntent souvent ses

notions de fonctions. Le but d'un programme écrit dans un langage fonctionnel est de retourner une valeur.

Une particularité que nous devons aborder est la possibilité d'avoir des effets de bords dans le paradigme fonctionnel. Un langage fonctionnel pur ne devrait pas avoir d'effets de bords. Malheureusement, ce ne sont pas tous les langages de programmation fonctionnelle qui respectent cette caractéristique. Regardons un peu ce qu'est l'effet de bord.

1.2.2.2 Les machines à effets de bords.

Qu'est-ce qu'une machine à effet de bord. C'est une propriété à laquelle la programmation fonctionnelle n'échappe pas. Les effets de bords sont résumés par le bout de code suivant :

```
int n = 2;
int inc(int k)
{
        n = n + k;
        return n;
}
f(inc(1) + inc(1))
```

Nous déclarons une fonction qui permet de modifier les variables globales. Prenons les fonctions *inc(1)* plus *inc(1)*. La première fois, la valeur sera 2+1=3 et la fois suivante 3+1=4. La fonction ne retourne pas la même valeur pour un paramètre semblable. C'est ce que nous appelons l'effet de bord.

Le paradigme fonctionnel est dispensé des effets de bords. Nous devons dire qu'il en est dispensé seulement et seulement si les concepteurs le veulent. La fonction est représentée par une boîte noire, avec une entrée et une sortie comme à la figure 10.

Figure 10 : Schéma d'une boîte noire.

Chaque boîte noire est définie comme étant une fonction. L'application est construite autour des fonctions qui sont emboîtées les unes dans les autres. Pour chaque fonction, il y a seulement un ensemble unique. Il n'y a pas de variables globales. Chaque fonction retourne un résultat unique. On dit qu'un programme est une fonction qui retourne un résultat unique pour une entrée unique. Abordons le fondement de ce paradigme : la fonction.

1.2.2.3 Les fonctions

Une fonction est normalement représentée par une boîte noire comme le présente la figure 10. Qu'est ce qu'une fonction? C'est un mécanisme qui va prendre en entrée une série d'arguments et va retourner une seule valeur. Une fonction ne vie que pendant son exécution. Dès qu'elle a exécuté son traitement, elle meurt. Lors de sa prochaine vie, elle fera un nouveau traitement. Elle n'a aucune mémoire de ce qui est arrivé la dernière fois. L'exécution d'une fonction veut simplement dire que nous l'évaluons de façon bête et méchante. Comment peut-on évaluer une fonction?

1.2.2.4 Comment évaluer une fonction

Les concepts mathématiques nous viennent en aide pour cette partie. Le lambda-calcul est l'outil qui va nous venir en aide. C'est *Alonzo Church* qui introduisit son lambda-calcul afin de nous donner un moyen d'évaluer les fonctions. Ce concept est en lien avec la logique combinatoire de *Haskell Curry*. Nous substituons à l'aide des règles pour arriver à trouver la forme simplifiée. Après la dernière simplification, nous ne devrions pas pouvoir la simplifier plus.

1.2.2.5 Propriétés de la fonction

Sans les fonctions, le paradigme n'existe pas. Une fonction est une formule mathématique. Ce n'est pas un algorithme comme on le prétend souvent par erreur. Formellement, une fonction mathématique f a un ensemble E de départ et un ensemble M d'arrivé. L'ensemble E doit remplir les conditions d'admissions pour la fonction f. Un ensemble qui n'admet que les types de l'ensemble des nombres entiers, n'acceptera pas les nombres réels. La composition de la fonction est une séquence d'instructions. La séquence d'instructions sera bien entendu un algorithme. Les deux concepts sont liés. Il ne faut pas les mélanger.

Nous avons la composition de fonctions. La composition de fonctions veut dire que nous prenons deux fonctions, soit : $f : E \to F$ et $g : F \to G$. Comme nous pouvons le voir, ces deux fonctions sont identifiées comme $g \circ f$. Nous devons suivre un ordre précis pour l'évaluation de cette composition. Si nous appliquons la variable « y » à cette composition, nous aurons $g \circ f(y) = g[f(y)]$. Nous appliquons f à y et la sortie de la fonction f est reprise par g. Une fonction peut aussi être injective. Cette propriété stipule que pour chaque élément de l'ensemble de départ, un seul élément de l'ensemble de l'arrivé est à lui. Elle est injective si et seulement si : $f(x_1) = f(x_2) \Rightarrow x_1 = x_2$. Une fonction est surjective si pour un élément de l'ensemble de l'arrivé, au moins un élément de l'ensemble de départ existe. $f : I \to J$ est surjective si $f(I) = J$.

La fonction qui est transmise en paramètre doit satisfaire les exigences du paramètre. Par exemple, la fonction f prend en paramètre un nombre entier et que la fonction g que nous essayons de lui passer en paramètre retourne un réel, alors nous avons un problème.

1.2.2.6 Les langages existants

Plusieurs langages fonctionnels ont été développés dans les années 1970 et 1980. De nos jours, la programmation Orienté Objet a pris la première place faisant oublier ce paradigme de programmation. Ce paradigme ne pourra pas être oublié. Dans la programmation Orienté Objet, on retrouve bien entendu des éléments de la programmation fonctionnelle. Voici quelques langages qui existent pour ce paradigme avec une courte description.

Langage *ML (Meta Langage)* a été développé par *Robin Milner* et sont équipe dans les années 1980 [Sebesta 2004]. Le *ML* est un langage qui permet de produire des effets de bords. *ML* était surtout utilisé pour l'écriture de compilateur ainsi que de langages. Ce langage utilise le *lambda-calcul*.

http://caml.inria.fr
http://www.smlnj.org/

Le langage *Haskell* a été implémenté pour la première fois par Paul Hudak et Philip Wadler [Hudak et Al. 2007]. Certains dérivés de ce langage ont été développés afin de le rendre pédagogique. Ce langage utilise la notation de la logique combinatoire.

http://www.cs.chalmers.se/Cs/Grundutb/Kurser/d1pt/d1pta/external.html
http://www.haskell.org/haskellwiki/Haskell

Nous venons de voir quelques paradigmes. Ces paradigmes, pour la plupart, on de solide base et nous pouvons les utiliser afin d'avoir une solution complète et pratique. Le seul que nous n'utiliserons pas dans projet est le paradigme Aspect. Le modèle de paradigme ne nous viendra pas en aide pour la conception de la plate-forme. Par contre, comme nous le verrons plus loin, le paradigme pourrait nous aider pour le développement des modules. Avant de décrire notre solution dans ses moindres détails et de voir la théorie, il serait intéressant de jeter un petit coup d'œil sur ce qui se fait présentement sur le marché. Le prochain chapitre sera consacré exclusivement aux plates-formes existantes. Nous feront une brève description pour chacune des plates-formes.

CHAPITRE 2
LES PLATES-FORMES EXISTANTES

La programmation sous forme de bloques Lego fait son appariation petit à petit. Des solutions concrètes et graphiques ont commencé à faire leurs apparitions. Nous devons voir si une de ces plates-formes pourraient nous aider à répondre à nos exigences. Nous pourrions peut-être rajouter seulement une extension et notre projet serait complet. Certaines sont plus graphique que d'autres mais conceptuellement parlant, elle se rapproche de l'idée de la programmation sous la forme des bloques Lego.

L'idée des plates-formes modulaires date de longtemps et plusieurs approches existent dans ce domaine. Voyons les différentes approches qui ont été pensés. Nous pourrions peut-être utiliser la notion de la réutilisation si jamais une plate-forme se rapproche de ce que nous voulons faire.

2.1 Knime (Konstant Information Miner)

http://www.knime.org/

Cette solution est construite en utilisant la plate-forme de l'open-source Java : Eclipse. Elle est bâtie avec l'architecture Java. Les entités sont prénommées des Nœuds. Chaque Nœud représente une classe avec des fonctions spécifiques. Pour relier ces Nœuds, on relit les classes. Les classes sont reliées à l'aide des méthodes. Il est important de spécifier que chacun des Nœuds doit hériter d'une famille de classes (Source, Sink, Learner, Predictor, Manipulator, Visualizer, Meta, Other). Le processus de communication entre les nœuds n'est pas vraiment expliqué. Nous savons que nous faisons affaire avec la technologie objet. Nous pouvons supposer que la communication se fait à l'aide du couplage entre les différentes méthodes du langage Java. La figure 11 présente une capture de l'écran de l'application KNIME. On peut voir que l'application est basée sur la plate-forme Éclipse.

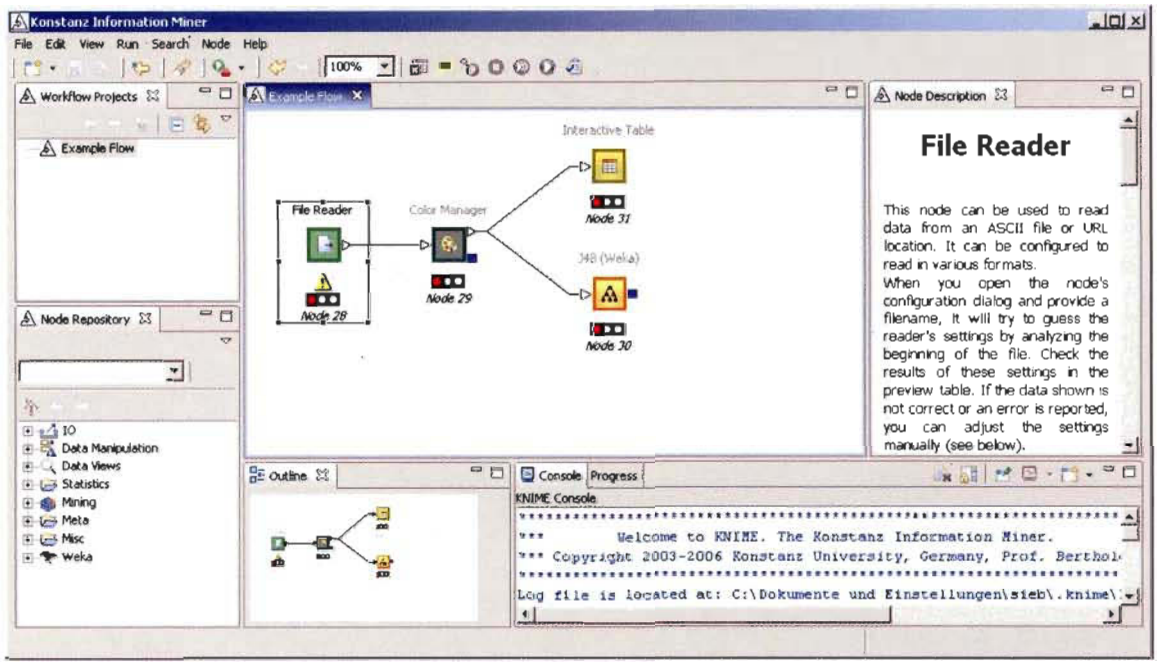


Figure 11 : Capture d'écran Knime.

Knime utilise la technologie Orienté Objet. La plate-forme *Eclipse* est construite sur l'orienté objet. Les modules sont tous construits en Java. Nous pouvons dire que tout est construit en Orienté Objet. L'environnement, de façon visuelle est construit sur le principe du paradigme agent. Les diverses entités communiquent entre elles. Cette plate-forme est disponible gratuitement sur Internet. On ne dit pas si un module peut être indépendant ou s'il peut être exécuté sur une autre plate-forme.

2.2 T2K (Data to Knowledge), D2K (Text to Knowledge)

http://alg.ncsa.uiuc.edu/do/index

Cette solution est très semblable à Knime. La grande différence est que la plate-forme est totalement indépendante. Elle est complètement construite en Java comme Knime. Les entités sont prénommées des modules. La création de ces modules se fait aussi à l'aide de l'héritage. 6 grandes familles existent : les entrées de l'utilisateur, les entrées provenant des sources (BD, fichier, ...), la sortie des données vers une source, une famille de préparation des données, une famille pour le traitement des données ainsi que finalement, une famille pour la visualisation des données. La

communication entre les nœuds n'est pas expliquée. Comme la technologie utilisée est l'objet, nous pouvons supposer que la communication se fait par le couplage des méthodes. De plus, l'application est capable de suivre la progression de chacun des modules. Ce qui veut dire qu'ils doivent passer par des objets pour pouvoir communiquer. La figure 12 présente la plate-forme D2K. Elle est très visuelle.

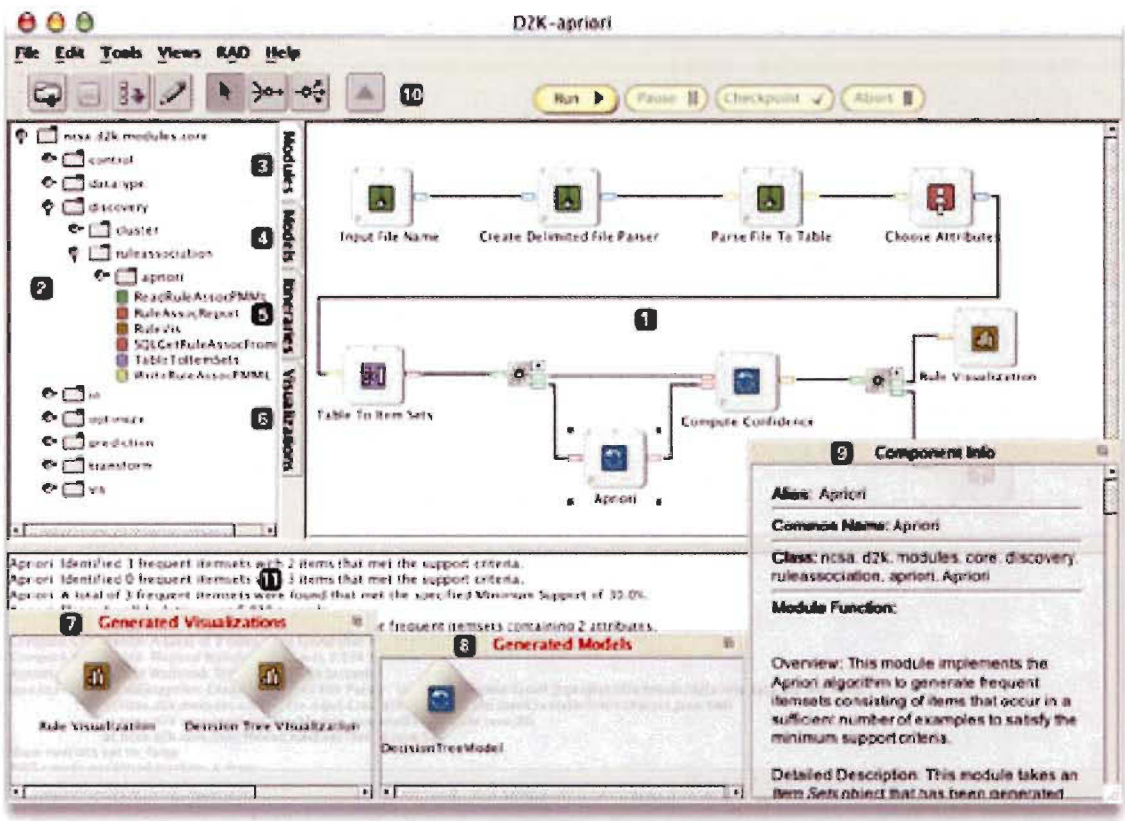


Figure 12 : Capture d'écran D2K.

La génération D2K / T2K utilise complètement la technologie Orienté Objet. Pourquoi je dis cela? Cette technologie est construite complètement en Java. La communication entre les modules semble aussi basée sur la méta-programmation. Il n'existe aucune documentation technique pour sa construction. Ce que nous pouvons déduire, c'est que tout est construit en Java : les modules et la plate-forme. Chaque module agit un peu comme des agents. On ne dit pas si le module peut être indépendant ou s'il peut être exécuté sur une autre plate-forme.

2.3 Gate (General Architecture for Text Engineering)

http://gate.ac.uk/

Gates est un outil pour le traitement linguistique. Ce n'est pas vraiment un outil à base de modules. Il y a un langage de programmation inclut dans l'outil. Le langage ressemble au langage Java. La plate-forme Gates est complètement développée en Java. Ce n'est pas un environnement très convivial pour une personne qui ne connaît pas les rouages des langages de programmation. Il est possible de créer de petits scripts. La figure 13 présente la capture d'écran de Gate. Nous pouvons remarquer que l'écran n'est pas si simple à prendre en main.

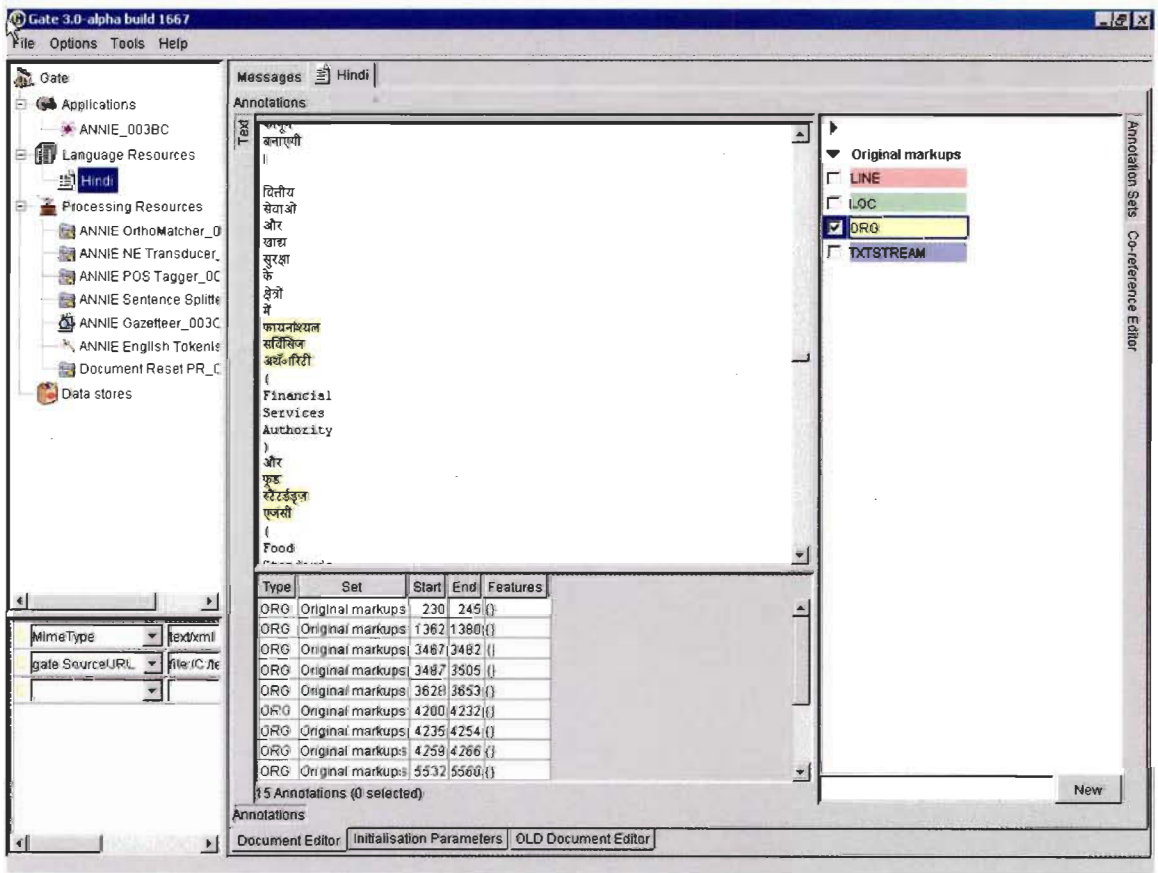


Figure 13 : Capture d'écran Gates.

Même si cette plate-forme est construite en Java, son principe est fonctionnel. Ce sont des fonctions qui s'emboitent entre elles. C'est une plate-forme qui est gratuite.

Il existe beaucoup de documentation pour construire des extensions de l'application. Cela dit, cette plate-forme n'est pas très modulaire. Si nous désirons présenter un traitement à quelqu'un, nous devons vraisemblablement lui envoyer tout le contenu de la plate-forme pour qu'il puisse l'utiliser. Cela peut devenir très lourd.

2.4 Monk Project

http://www.monkproject.org/

Le projet Monk est composé essentiellement de deux sous projets. Le premier est le projet NORA et le second projet est WordHoard. Ce projet vise le traitement des textes littéraires. C'est de faire l'étude des textes assistés par un ordinateur. On parle de très grands corpus.

2.5 WordHoard

(http://wordhoard.northwestern.edu)

WordHoard est un projet visant les grands corpus qui ont été écrits voilà plusieurs années. Le projet est fait pour trouver des classes de mots. On analyse et on fait de la recherche. Aucune explication n'est donnée sur les méthodes du traitement. Ce projet permet de créer des scripts afin d'utiliser nos propres méthodes. Le langage de script ressemble de près au langage C. L'application elle-même est créée dans le langage Java. Il ne sert qu'à l'analyse de textes. Ce n'est pas une plate-forme modulaire comme Knime ou D2K / T2K. La figure 14 présente une capture d''écran de l'application WordHoard. Les utilisateurs doivent se familiariser avec l'application avant de l'utiliser.

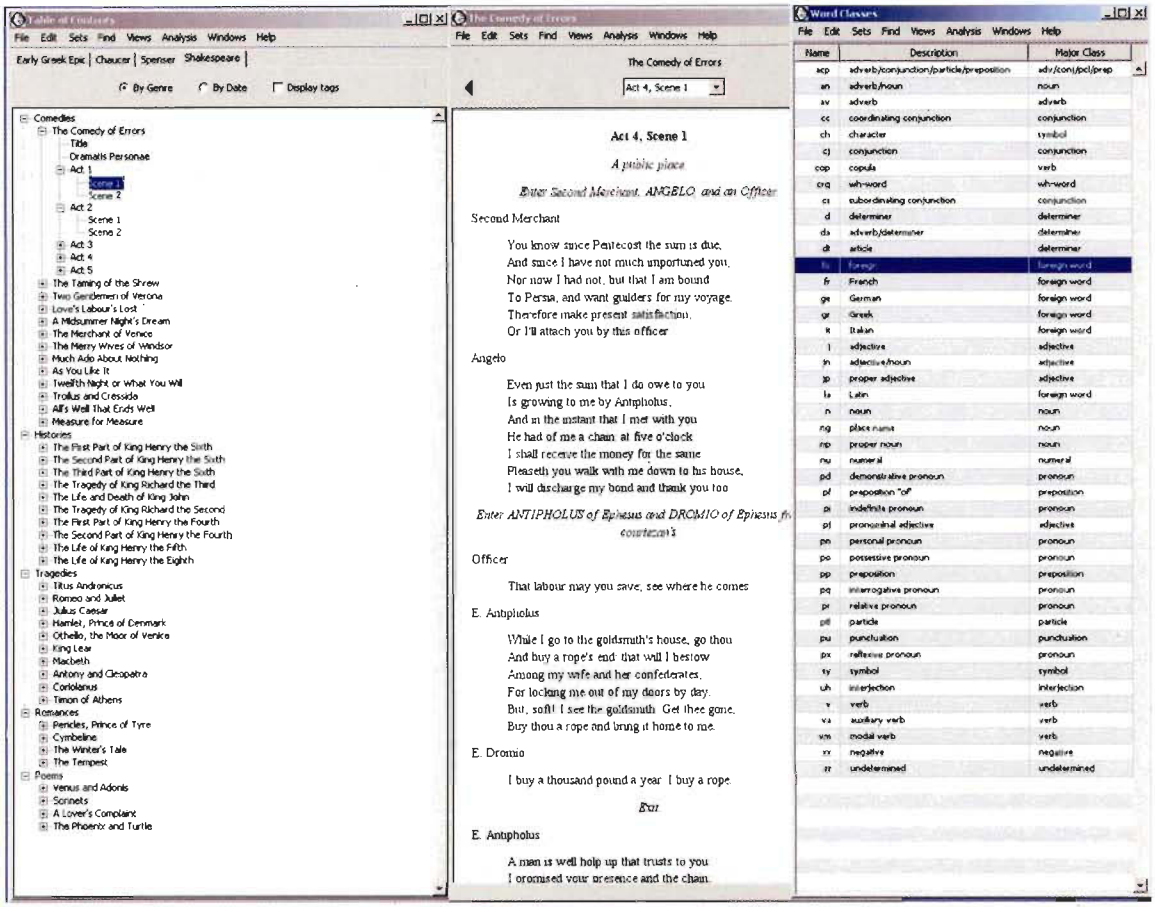


Figure 14 : Capture d'écran WordHoard.

2.6 NORA

http://www.noraproject.org/

Le projet NORA a été conçu pour forer les textes afin de trouver des modèles. Ce projet vise plus précisément les textes numérisés. Ils utilisent des techniques de data-mining (Forage de données). Les concepteurs de cette application on aussi contribués au développement de D2K. Leurs techniques pour le développement est qu'ils ont décidé de réunir des développeurs et des chercheurs. À partir de là, les deux groupes ont recherchés une manière de bien faire l'analyse des textes. Cette application est une plate-forme fermée qui prend des textes et les analyses selon certains critères. Nora utilise quelques ressources de D2K. Ce qui nous donne l'explication du forage des données (data-mining). La plate-forme D2K a été principalement conçue pour le forage de données.

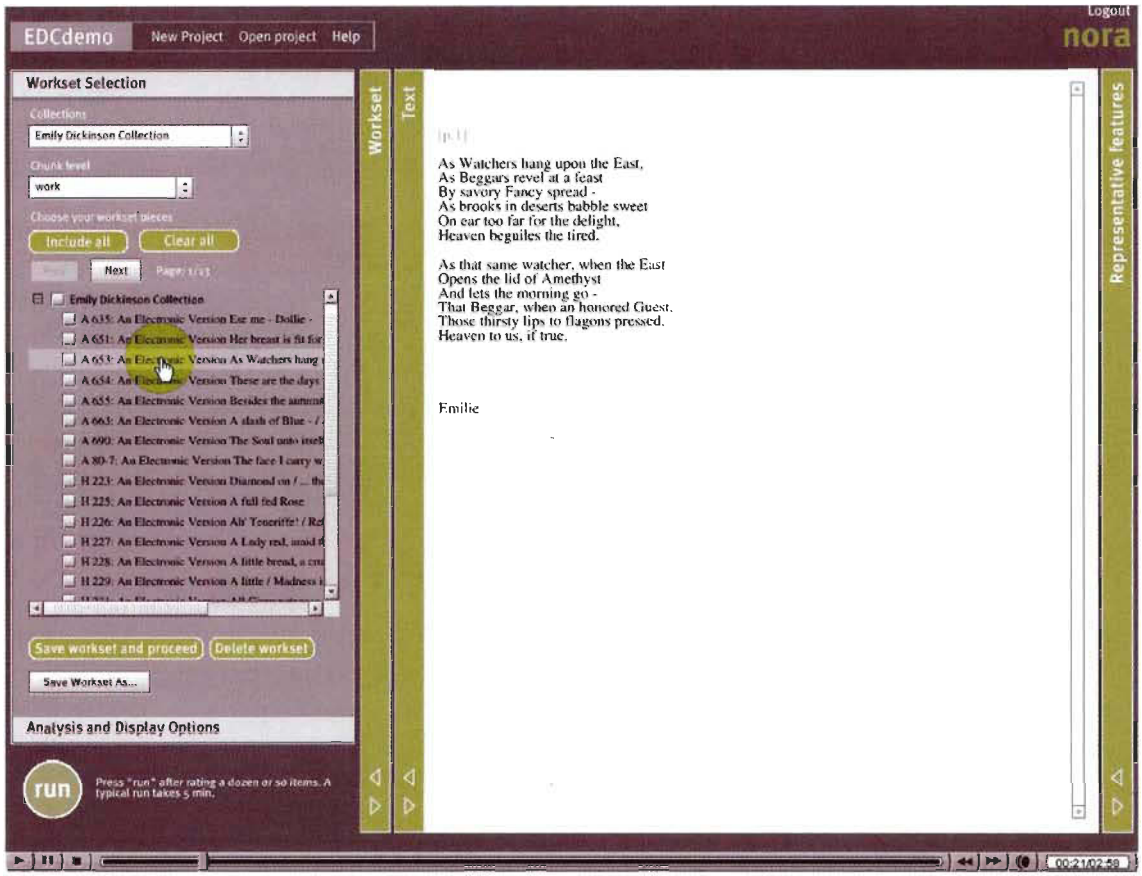


Figure 15 : Capture d'écran du projet NORA.

La figure 15 présente l'application. On peut remarquer que cette application est relativement facile à utiliser. La plate-forme NORA semble principalement construite sur le langage JAVA. La plate-forme est donc construite en Orienté Objet. L'application fonctionne néanmoins de façon fonctionnelle. On applique les traitements un à la suite des autres. On a accès à un nombre de composants pour faire notre travail. Si nous en avons besoin de plus, il semble qu'avec le code-source, il y ait un moyen de pouvoir en créer nous même. Cela admet qu'il faut une expertise en programmation et que la personne doit apprendre comment fonctionne la plate-forme.

2.7 UIMA

(http://incubator.apache.org/uima/)

(http://domino.research.ibm.com/comm/research_projects.nsf/pages/uima.index.html)

Ce projet a débuté sous les gites d'IBM et a été donné au gens de l'open source. La société Apache a repris le développement par la suite. Ce projet a été créé afin de prendre des informations non structurées comme le texte, les textes de clavardage, les bandes sonores, courriels,... Cette information subit une transformation pour être structurée. À la base, on ne parle pas d'une plate-forme mais d'un ensemble d'outils (Kit de développement). Plusieurs entreprises ayant repris ce projet utilisent la plate-forme Eclipse pour le mettre en œuvre. On doit cependant comprendre ce que le code fait et pour cela, il faut avoir des connaissances en programmation. On parle plus d'un kit de développement que d'une application modulaire. Le code est fourni sur le site web. C'est à chaque personne de faire ce qu'il veut avec ce projet.

2.8 En résumé

Knime et D2K, T2K sont des outils qui utilisent complètement l'Orienté Objet. L'utilisateur utilise des classes qui ont été créées et s'amuse à les connecter. Cette technologie peut être vu comme la conception à base de classes. L'utilisateur s'amuse à connecter les classes un environnement graphique. Cela ressemble à un gros jeu de briques Légos.

Les données techniques de ces plates-formes sont assez bien gardées particulièrement pour D2K / T2K. Ces deux outils sont propriétaires. Knime relève plus de l'open source. Son code devrait normalement être disponible. Aucune explication réelle n'est démontrée concernant la communication entre les modules ni sur la façon dont les chaînes de traitement sont sauvegardées sur ces plates-formes. La seule chose que nous savons, c'est que toute l'architecture est construite avec le langage Java. Il est

important de spécifier que nous ne pouvons pas prendre des modules de Knime et les insérer dans D2K / T2K ou vice versa. Les modules ne répondent pas à la spécification d'une ou de l'autre de ces plates-formes. Même si les deux types de modules sont construits avec de la technologie Java, ils n'ont pas les mêmes spécifications. Il est impossible de les faire fonctionner ensemble dans l'état actuel des choses.

Une des principales raisons pourquoi de tels projets sont entièrement construits en objet, c'est qu'il est plus facile de gérer tout le processus d'exécution des objets. Les problèmes d'erreurs et de la communication sont relativement vite réglés. Un module ne pourra pas être indépendant de la plate-forme. Ces plates-formes demandent d'avoir une expertise dans le domaine de l'informatique. Si vous ne trouvez pas les modules que vous avez besoin, vous allez devoir vous le programmer. De la documentation existe. Elle est très basique.

En ce qui concerne les autres plates-formes telles que Gates, Nora, WordHoard. Elles sont des plates-formes pour le traitement linguistique uniquement. Elles permettent de faire un peu de programmation avec un langage de script. Gates et WordHoard ainsi que NORA sont conçus en Java.

Pourquoi ces plates-formes sont-elles programmées en Java? Pour la plupart, ces projets ont émergés des universitaires dans d'autres domaines que directement l'informatique. Les gens avaient probablement très peu de connaissances dans les langages de programmation. Le langage Java est enseigné dans les universités comme base en programmation. On apprend aux étudiants à développer avec Java pour leurs enseigner l'Orienté Objet. De ce fait, il est normal de retrouver des plates-formes qui permettent de faire la création de chaînes ressemblant beaucoup au schéma UML.

Il nous reste UIMA. Cette plate-forme ou plutôt son kit de développement est donné à la communauté open-source. Il est très en avance sur les prédécesseurs que nous venons de parler. L'environnement est très riche en termes de fonctions. Ce projet

vise les données sonores, visuelles, textuelle,... Sans doute va-t-il avoir un très bon avenir dans le domaine de la recherche. Il offre plusieurs méthodes afin de faire de la recherche. Son code source est entièrement disponible. Un simple utilisateur peut avoir de la difficulté à s'y retrouver. Il pourrait contenir beaucoup de fonctions intéressantes pour notre projet.

Nous venons de voir une panoplie de différentes plates-formes. Certaines sont plus intéressantes que les autres. Un gros problème est que ces plates-formes sont très enracinées dans la logique des programmeurs et il faut presque tout le temps être un programmeur pour pouvoir les utiliser ou du moins avoir de bonne base en informatique. Il est difficile de penser à programmer des extensions pour arriver à notre projet. La seule partie qui pourrait être envisageable d'intégrer notre projet est UIMA. Nous pourrions utiliser ses fonctions dans notre projet. Dans le prochain chapitre, nous allons décrire les aspects théoriques de notre solution.

CHAPITRE 3
RECHERCHE THÉORIQUE

3.1 Introduction

Au premier chapitre, nous avons étudié différents paradigmes. Chacun a ses forces et ses faiblesses. On ne peut pas prétendre qu'un seul de ces paradigmes résoudra tous les problèmes. Il faut les faire coopérer entre eux. Dans le second chapitre nous avons étudié les plates-formes existantes. Nous avons vu que dans ce qui existe déjà rien ne peu vraiment nous servir mis à part le kit UIMA. Dans ce chapitre, nous allons voir en détails les concepts de notre solution. Au début du chapitre, nous allons voir les aspects théoriques. Par la suite, nous verrons plusieurs exemples de l'application de la théorie. Notre solution se base essentiellement sur la logique combinatoire. Cette logique va nous permettre d'emboiter nos modules comme le jeu de briques Lego. Les pièces s'emboîtent en respectant certaines règles. La logique combinatoire va nous donner ces règles. De plus, cette logique va nous donner l'ordre d'exécution de nos modules très rapidement. Comme nous allons le voir durant le chapitre, nous donnons un schéma à l'ordinateur et ce dernier doit réussir à exécuter le tout dans l'ordre. Il ne faut pas que le second module s'exécute avant le premier. Le premier s'exécute, le second s'exécute… Cet ordre est primordial pour avoir un traitement juste. On ne peut pas commencer par afficher les données avant de les saisir. Il faut une logique. La logique combinatoire va nous donner l'ordre d'exécution directement des modules. Nous n'aurons pas à faire de tests pour approuver cet ordre. Toute l'architecture de la solution est construite autour de la logique combinatoire.

3.2 La logique combinatoire

La logique combinatoire est un principe mathématique qui n'est pas très connu. Elle a eu, comme père fondateur, Gottlob Frege [Cardone et Al. 2006] qui en énonça les grandes lignes vers 1891. Ce n'était que le début. En 1920, Moses Schönfinkel a introduit la notion de combinateur [Cardone et Al. 2006]. Un combinateur est un

opérateur abstrait qui nous permet de construire une suite d'opérateurs de plus en plus complexes. Nous partons d'une forme basique et que nous la complexifions. Chaque combinateur agit selon une règle qui lui est propre. Cette règle est appelée la β-réduction.

La logique combinatoire repose sur deux combinateurs de base. À partir de ces combinateurs, nous pouvons construire des combinateurs plus complexes. Les deux combinateurs de base sont le combinateur de composition S et le combinateur d'effacement K.

3.2.1 Le combinateur S de composition ou substitution

Ce combinateur de base va nous permettre de faire des compositions. Il prend trois arguments. La règle de la réduction est :

$S\ x\ y\ z \rightarrow x\ z\ (y\ z)$

x et y sont des opérateurs et z est une opérande. L'action de S est de distribuer l'opérande z aux opérateurs x et y. y appliqué à z retourne les opérandes de la fonction complexe obtenues à partir de la substitution de x par z

3.2.2 Le combinateur K d'effacement

Ce combinateur va nous servir à effacer les autres opérateurs. La règle de la réduction est la suivante :

$K\ x\ y \rightarrow x$

x et y sont des opérateurs. L'action de K sur les opérateurs a pour effet d'effacer l'opérateur y.

Nous pouvons écrire toute la logique combinatoire à partir de ces deux combinateurs. Par contre, plus nous entrons dans la complexité et plus les expressions combinatoires peuvent être lourdes à lire. Pour cela, nous allons donner des équivalences pour représenter des actions. Les expressions qui sont entre les parenthèses respectent la priorité des opérations. Nous donnerons, pour chacun des combinateurs élémentaires, l'équivalence avec les combinateurs S et K.

3.2.3 Le combinateur I d'identité

Un combinateur I appliqué à un opérateur donnera l'opérateur lui-même. L'opérateur restera inchangé. L'action du combinateur est décrite avec la règle de réduction suivante :

$I\ x \rightarrow x$

x est un opérateur. L'action du combinateur I sur l'opérande sera de donner son identité. Ici, l'identité de x sera x.

Équivalence en S et K
$I = S\ K\ K$

3.2.4 Le combinateur B de composition

Ce combinateur va combiner deux opérateurs entre eux afin de former un opérateur complexe. Prenons x et y, 2 opérateurs et appliqués à un argument x afin de concevoir un opérateur complexe. Voici la règle de la réduction :

$B\ x\ y\ z \rightarrow x\ (y\ z)$

x et y sont des opérateurs et z est un opérande. L'action du combinateur B est de faire la composition d'un opérateur complexe avec un opérateur et une opérande. Ici,

l'opérateur y sera composé avec l'opérande z. Cela va nous donner l'opérateur complexe.

Équivalence en S et K
B = S (K S) K

3.2.5 Le combinateur Φ de coordination

Ce combinateur va distribuer un opérateur à deux opérandes. Voici la règle de la réduction :

$\Phi\ x\ y\ z\ u \rightarrow x\ (y\ u)\ (z\ u)$

x, y et z sont des opérateurs et u est une opérande. L'action du combinateur de coordination sera de coordonner l'opérande avec les deux opérateurs qui le précède. Ici, l'opérande u est coordonné à y et z.

Équivalence en S et K
Φ = B (B S) B

Nous pouvons remarquer que le combinateur de coordination est réécrit avec le combinateur B. Si nous avions tout écrit en S et K, la chaîne aurait été très lourde. Voici la substitution en opérateur de base à titre d'exemple.

Φ = B (B S) B = (S (S K) K) ((S (S K) K) S) (S (S K) K)

Avec cet exemple, on comprend mieux l'introduction de combinateurs complexes. Cela allège grandement les expressions.

3.2.6 Le combinateur Ψ de distribution

L'action du combinateur Ψ de distribution, comme son nom l'indique, va servir à faire la distribution d'un opérateur à deux opérandes. Remarquez la différence avec le combinateur de coordination Φ qui celui-là, distribue l'opérande à deux opérateurs. Voici la règle de réduction :

$\Psi\ x\ y\ z\ u \rightarrow x\ (y\ z)\ (y\ u)$

x, y des opérateurs et z et u sont des opérandes. L'action du combinateur sera de faire la distribution de l'opérateur y aux opérandes z et u.

Équivalence en S et K

$\Psi = \Phi\ (\Phi\ (\Phi\ B)))\ B\ (K\ K)$

3.2.7 Le combinateur C de permutation

Ce combinateur, comme son nom l'indique, va nous servir à permuter deux opérandes. Ces deux opérandes seront reliés à un opérateur. Si y et z sont des opérandes reliés à l'opérateur x, alors le combinateur C agira sur nos deux opérandes avec la règle de la réduction suivante :

$C\ x\ y\ z \rightarrow x\ z\ y$

x est un opérateur et y et z sont des opérandes. L'action du combinateur de permutation est de permuter les opérandes y et z pour obtenir z et y.

3.2.8 Le combinateur C* de changement de type

Ce combinateur va nous donner la possibilité de faire changer un type pour un autre. L'action de ce combinateur est décrite avec la réduction suivante :

$C* XY \rightarrow YX$

X est un opérateur et Y est une opérande. L'action du combinateur sera de faire le changement du type de l'opérateur X. X va passer du type opérateur au type Y d'opérande.

Équivalence en S et K
C = S (B B S) (K K)

3.2.9 Le combinateur W de duplication

Comme son nom l'indique, nous allons utiliser ce combinateur afin de dupliquer son opérande. La règle de la réduction est la suivante :

$W\ x\ y \rightarrow x\ y\ y$

Si x est un opérateur et y un opérande, l'application de ce combinateur va nous permettre de dupliquer l'opérande y.

Équivalence en S et K
W = S S (K (S K K))

3.2.10 Les combinateurs complexes

À partir de tous ces combinateurs, nous avons la possibilité de créer un combinateur complexe. Un combinateur complexe est composé de plusieurs combinateurs élémentaires. Par exemple :

B C x y z
B C W x y z
Φ B C x y z u

Comme nous pouvons le voir, nous pouvons faire des agencements de combinateurs complexes. En faisant cela, nous avons une possibilité infinie de combinateurs complexes. Chaque expression combinatoire peut devenir un combinateur complexe.

3.2.11 Puissance d'un combinateur

Nous avons un cas plus particulier des combinateurs complexes. Ce cas est la puissance d'un combinateur. Il indiquera le nombre d'opérandes qui sera affectées par le combinateur. La règle de la réduction est : si X est un combinateur alors X^n itère n fois l'action du combinateur X tel que :

$X^1 = X$ et $X^n = BXX^{(n-1)}$

Exemple :
B^2 a b c d e → a (b c d)
B^4 x y z u v w → x (y z u v w)

3.2.12 Combinateurs à distance

Un autre cas des combinateurs complexes est celui de la distance. Cela nous donnera l'endroit où se déroulera l'action du combinateur. La règle de la réduction est : si X est un combinateur alors X_n diffère son action de n pas.

$X_n = B^n X$ et $X_0 = X$

Exemple :

C_2 a b c d e $\rightarrow$ B^2 C a b c e d $\rightarrow$ a b c e d

B_3 a b c d e $\rightarrow$ B^3 B a b c d e $\rightarrow$ a b c (d e)

En utilisant les combinateurs de la logique combinatoire, est-ce qu'il est possible de pouvoir créer des enchaînements afin de produire une expression logique pour représenter des chaînes?

3.3 Un module

Un module est une entité au même sens que pour le paradigme fonctionnel. Ce n'est pas un module au même titre que dans la programmation de Microsoft en Visual Basic. Le module est comme une fonction mathématique.

3.3.1 La base

Un module fait un traitement. Il n'aura pas de descendant avec une hiérarchisation de classes comme dans l'Orienté Objet. Un module peut être construit à l'aide du paradigme Orienté Objet. Chaque module sera indépendant des autres sans aucun lien hiérarchique. Comme une boîte noire, nous savons que la fonction effectue un traitement. Nous ne savons pas comment elle le fait, cela ne nous regarde pas. C'est le concepteur qui décide.

Un module devra être indépendant des autres modules au même titre que les agents dans le paradigme Agent. Il devra avoir la capacité de pouvoir communiquer avec ses pairs. Il devra effectuer une seule action. Jusque où l'action devrait-elle être démembrée? Cela regarde le concepteur du module. Deux concepteurs décident de faire un module ou une série de modules pour le même traitement. Un prend la décision, que pour lui, un seul module fait l'affaire. Son confrère décide qu'il lui en faut cinq. Cela est tout à fait possible. C'est le programmeur qui décide. Il n'a qu'à respecter les règles de la conception. Chaque personne aura une vision différente du problème. Le module sera utilisé dans la plate-forme. Il pourra être utilisé hors de la plate-forme.

Comment les modules peuvent-ils coopérer? Les modules vont communiquer à l'aide d'un protocole comme dans le paradigme Agent. Ils devront respecter des règles claires afin d'uniformiser toute la communication. Cette communication pourra se passer dans la communauté modulaire. Ce sera une communication de un à la fois. Un module parle et les autres écoutent. Il faudra un modérateur.

Un module devra fournir des informations sur lui-même. Quand nous parlons d'indépendance, cela en fait partie. Le module qui se retrouvera hors de la plate-forme, devra dire lui-même ce qu'il est. Il devra fournir son nom. Il devra fournir une description de lui-même et dire aux autres ce qu'il fait comme traitement. Il aura à fournir la liste de ses entrées et de sa sortie. Pour que le module puisse recevoir l'information d'un confrère, il lui faut un point d'entrée. Si le module veut pouvoir envoyer des informations, il lui faut un point de sortie. Un module ne doit pas nécessairement contenir des entrées ou une sortie.

3.3.2 Communication entre les modules (Les types)

Pour que la communication entre les modules soit efficace, elle doit être claire et simple. Une entrée et une sortie sont identiques à l'exception d'un petit détail. Une

entrée va permettre à un module de recevoir des données à traiter. La sortie va permettre au module d'envoyer les données à un autre module. Si nous enlevons ce détail important, l'entrée et la sortie sont identiques. Pour que la communication puisse se produire entre deux modules, nous devons connecter la sortie d'un premier module sur l'entrée d'un second. Est-ce possible de connecter une entrée sur une sortie? Y a-t-il des règles?

Premièrement, nous devons nous assurer que les domaines de la sortie et de l'entrée sont identiques. Un domaine est le type de données. Il faut que le domaine de sortie du module *A* corresponde au domaine d'entrée du module *B*. Voici la grammaire des entrées et de la sortie :

Domaine = Theme, Domaine
| Theme

Theme = Caractere
| ChaineCaractere
| Entier
| Reel
| Liste (Caractere | ChaineCaractere | Entier | Reel | Liste)*

Liste = Liste(Caractere
| ChaineCaractere
| Entier
| Reel
| Liste
| (Caractere | ChaineCaractere | Entier | Reel | Liste)*)

Cette grammaire est composée de quatre types de structures de base. Soit, les *Caractères*, les *Chaînes de Caractères*, les *Entiers* ainsi que les *Réels*: Nous avons la possibilité de construire des Listes.

Il est possible de créer des types de domaines vraiment variés. Voici quelques exemples de domaines :

- Entier,Entier,Liste(Entier)
- Liste(ChaineCaractere)
- Caractere,Entier,Reel,Liste(Liste(Entier))

Nous venons de présenter 3 domaines différents. Ces domaines vont nous indiquer la compatibilité de la sortie avec l'entrée. Si une sortie prend comme domaine un *Entier* et que l'entrée est un *Reel*, alors il ne sera pas possible pour les deux modules d'être reliés. Il serait possible de les relier en utilisant par un module pour la conversion d'un *Entier* en *Reel*. Ce module, s'il n'existe pas, devrait être créé.

Un module peut présenter de zéro à plusieurs entrées. Par contre, il ne peut avoir que zéro ou une sortie. Pourquoi une telle règle? Quand nous faisons un traitement, ce dernier prend plusieurs entrées et produit un résultat unique à la fin. Prenons, par exemple, un examen chez le médecin. Qu'est ce que le médecin fait? En premier, il va probablement vous examiner la bouche. Par la suite, il prendra un test sanguin. Il vous regardera les oreilles et finalement, prendra votre pression artérielle. Tous ces traitements vont donner un diagnostic vous disant que vous êtes malade ou en santé. Nous prenons des intrants. Ils produiront un extrant. Ce cas concerne le domaine de la médecine. Nous pouvons très bien l'appliquer à plein d'autres situations. Par exemple, une automobile. Prenez l'usine de fabrication. D'un côté, les pièces détachées rentrent et de l'autre, vous avez le produit fini qui en sort. Vous avec une automobile à la sortie.

3.4 Une chaîne de traitements

Une chaîne de traitements est un agencement de modules. Bien que la vraie description d'une chaîne soit cela, elle peut être représentée par un module seul. Si nous avons un module seul, nous le considérons automatiquement comme étant une chaîne. Il n'y a pas de limite à la longueur de la chaîne. En fait, nous avons deux règles principales dans une chaîne :

1. La chaîne doit contenir un minimum de un module.
2. La chaîne est syntaxiquement correcte.

Quand nous parlons de « syntaxiquement correcte », c'est que chaque sortie qui est connectée sur une entrée doit avoir le même domaine pour être relié. Si ce n'est pas le cas, alors ces modules ne seront pas connectés. Ils formeront 2 chaînes de traitements comme dans la figure 16.

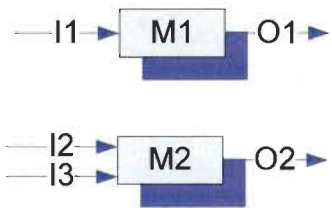


Figure 16 : Deux chaînes de traitements.

Nous ne parlons pas de la sémantique de la chaîne. Reconnaître qu'une chaîne est sémantiquement correcte n'est pas du ressort des mathématiques ou des ordinateurs et encore moins du programmeur. Un expert pourra dire si la chaîne, pour un domaine en particulier, est sémantiquement valide.

Une chaîne n'a pas non plus de lois sur l'ordonnancement de ses modules. Certaines plates-formes existantes ont fourni un schéma. Ce schéma nous indique comment préparer les données, les traiter et finalement les présenter. Il est du ressort de l'expert à décider ce qui convient de faire.

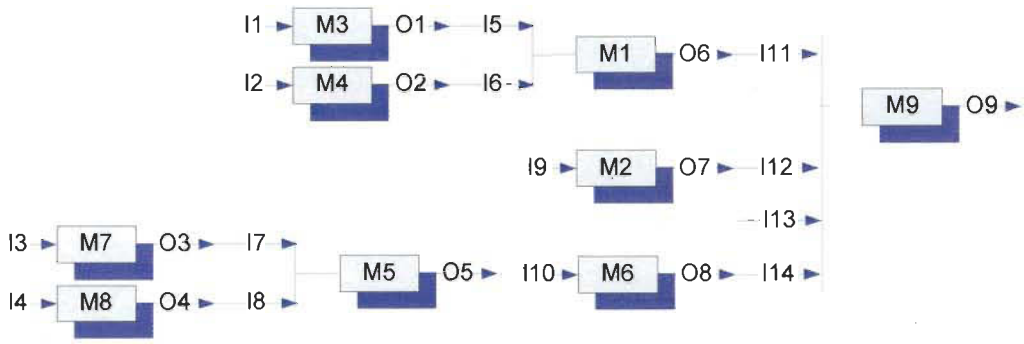


Figure 17 : Modules non ordonnés.

Une chaîne de modules peut être représentée en un module. Elle doit obligatoirement respecter les deux règles que nous avons formulées plus haut. Elle peut prendre différentes formes. Si le premier module de la chaîne ne comporte pas d'entrées, est-ce qu'il s'agit quand même d'une chaîne? Oui, une chaîne n'est pas obligée d'avoir d'entrées ou une sortie. Nous disions, plus haut, qu'un module peut avoir de zéro à plusieurs entrées et de zéro à une sortie. Cette définition respecte quand même nos deux lois. Si un module ne contient ni entrées et ni sortie, alors ce dernier est considéré comme un module unique. Il respecte la première loi qui dit qu'une chaîne contient au minimum un seul module. Il respectera la seconde loi qui dit que les modules doivent être syntaxiquement corrects. Il est seul. Il n'est pas nécessaire de l'attacher à un autre module. Il respecte la seconde loi. Ce module sera une chaîne à lui seul. On peut appeler ce cas le cas d'une chaîne fermée. Elle s'auto suffit à elle-même.

Nous avons notre chaîne. En théorie, tout cela se passe très bien. En pratique, comment est-elle exécutée? Un module est un programme dans l'environnement technologique. Il lui faut un déclencheur. Nous entrons dans la méta-programmation.

3.5 Paradigme de la Meta-Programmation

Une chaîne est exécutée dans un ordre précis. Si vous avez trois modules, chacun sera exécuté dans l'ordre où l'utilisateur le voudra. Les modules sont séparés. Aucun n'est lié comme à la figure 18.

Figure 18 : Module individuel.

Si nous avons une chaîne de trois modules, alors nous devons nous inspirer de notre paradigme Agent. Dans l'univers des agents, nous avons des agents qui donnent des tâches à chacun des autres agents. Notre agent sera nommé contrôleur de modules. C'est lui qui déterminera qui doit être exécuté en premier, en deuxième et ainsi de suite. C'est ce qui est appelé la Méta-Programmation. Le contrôleur agit en tant qu'arbitre. Il s'assure qu'il n'y a qu'un module qui s'exécute à la fois. La figure 19 présente le contrôleur en tant qu'arbitre. C'est lui qui va prendre la décision de l'ordre d'exécution.

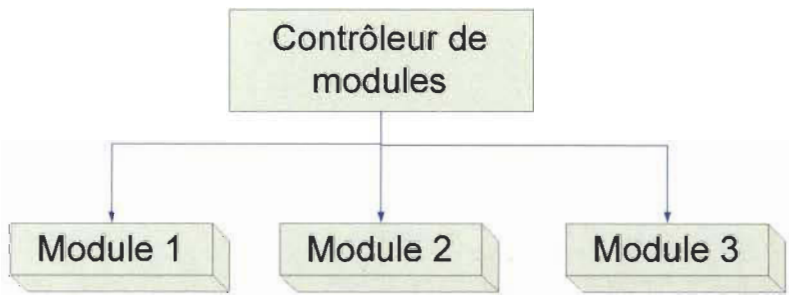


Figure 19 : Contrôleur de modules.

Le contrôleur peut agir sur d'autres contrôleurs. Nous parlons qu'une chaîne de modules peut être considérée comme un module. Une chaîne devient un module. Cela veut dire qu'elle peut être insérée à l'intérieur d'une autre chaîne.

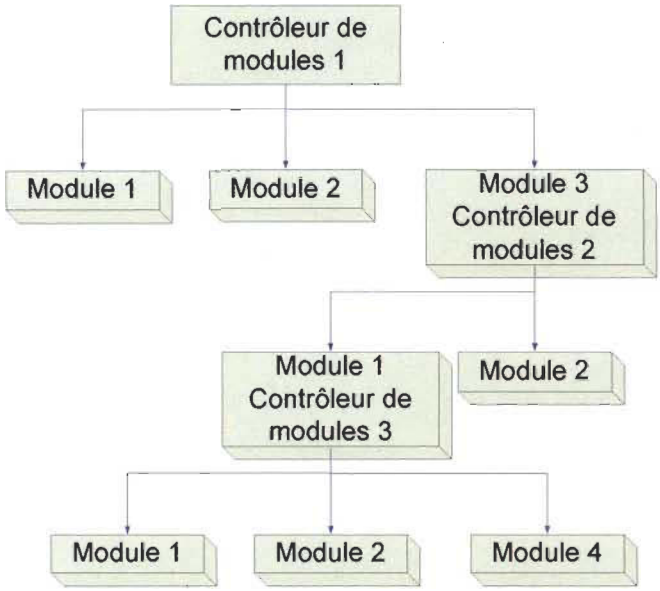


Figure 20 : Chaîne de modules et contrôleur de modules.

La chaîne de la figure 20 expose un tel arrangement de contrôleurs. Voici la décortication de la chaîne.

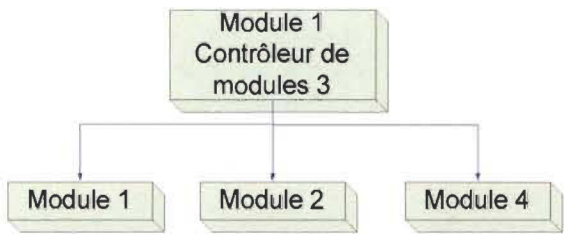


Figure 21 : Chaîne de modules Contrôleur 3.

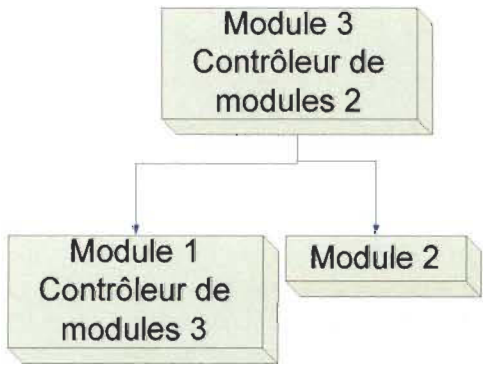


Figure 22 : Chaîne de modules Contrôleur 2.

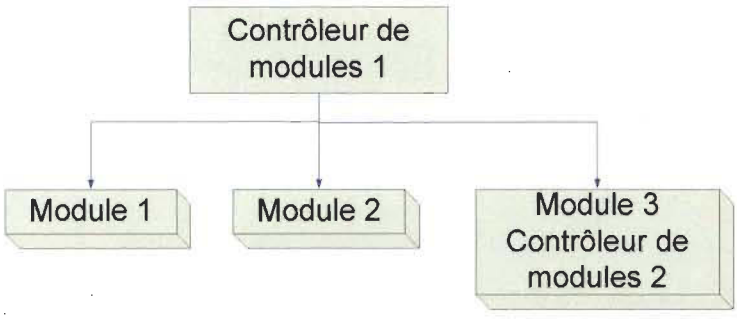


Figure 23 : Chaîne de modules Contrôleur 1.

Comme nous le voyons, la chaîne de contrôleurs est en cascade. Avant chaque exécution, on doit s'assurer que son prédécesseur soit exécuté. Quand nous entrons dans un contrôleur, avant que le module suivant ne soit exécuté, il exécutera tous les modules qui sont à lui. La Méta-Programmation assure l'ordre d'exécution de tous les modules. Le contrôleur n'a toujours pas son plan. Son plan sera donné par la logique combinatoire. Une expression combinatoire égale une chaîne combinatoire.

3.6 Représentation graphique des chaînes

Avant de voir comment on rédige le plan d'une chaîne de traitements, il est nécessaire de présenter deux formes de présentation de chaînes de traitements.

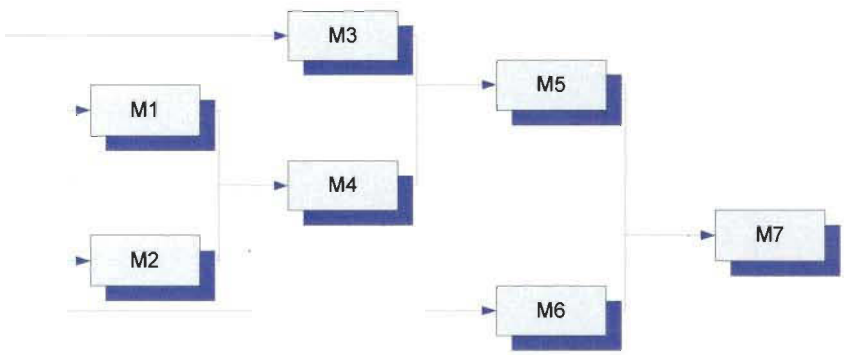


Figure 24 : Représentation d'une chaîne de traitements Horizontale.

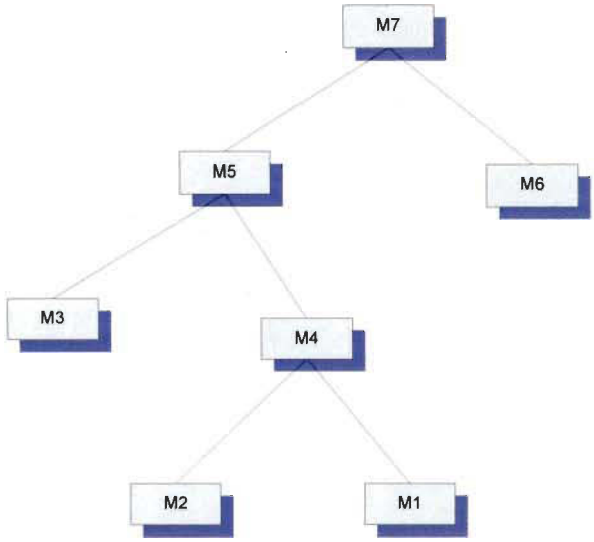


Figure 25 : Représentation d'une chaîne de traitements sous forme d'arbre.

Ces deux représentations graphiques représentent la même chaîne de traitement vu sous un angle différent.

3.7 Modélisation théorique des chaînes de traitements

La logique combinatoire va nous aider à modéliser une chaîne pour pouvoir retracer les connexions entre les modules. Nous avons des dessins pour représenter les chaînes, mais aucun langage formel. La logique combinatoire nous apporte une solution pour la représentation logique. Avec une chaîne combinatoire, nous pouvons la transmettre à nos collègues sans devoir donner un schéma avec. En plus, la logique combinatoire nous donne l'ordre d'exécution direct que nous verrons plus tard.

Pour construire une chaîne de traitements, nous avons besoin d'une information essentielle. C'est la liste des modules avec leurs entrées et leurs sorties que nous avons besoin. Cette liste sera importante autant pour la construction que la déconstruction de la chaîne.

3.8 Représentation des modules

Chacun des modules est marqué d'un M et est suivi d'un numéro. Donc, M1, M2, M... Il ne peut y avoir deux modules portant le même nom logique dans une même chaîne. Deux modules peuvent faire le même traitement mais ne porteront jamais le même nom logique. Chaque module a ses entrées à lui. Les entrées sont numérotées dans l'ordre I1, I2, ... I étant pour *input* en anglais. Les entrées sont libres si elles ne sont pas connectées à une sortie d'un module. Pour les sorties, nous avons soit une sortie ou soit aucune. La sortie est nommée O1. Voici l'exemple d'un premier module M1 à la figure 26. Ce module à une entrée et une sortie. La sortie peut avoir comme domaine I1 = {Entier, Entier, Caractere, Entier, Liste(Entier)} et le domaine de la sortie peut être O1 = {Entier}. La sortie et l'entrée n'ont aucun lien en ce qui concerne le domaine. Les deux domaines sont différents.

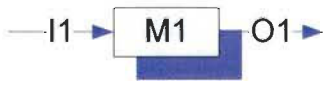


Figure 26 : Représentation d'un module à entrée unaire et sortie unaire.

Dans une chaîne de traitements, les noms doivent être unique autant pour les modules que pour les entrées et les sorties. L'attribution des noms par une personne lui est personnelle. Aucune règle n'existe. Pour nous, nous allons utiliser la numérotation en continue. Le module M1 ayant une entrée I1 et une sortie O1. Le module M2 ayant deux entrées I2, I3 et une sortie O2 et le module M3 ayant 2 entrées I4, I5 et une sortie O3. La numérotation sera ainsi de suite. Dans la pratique, l'application devra gèrer elle-même la numérotation.

Le module de la figure 27 se nomme M1. Il a deux entrées et une sortie. Les entrées sont indépendantes. Les entrées pourraient être I1 = {Entier, Entier, Reel} et I2 = {Entier, ChaineCaractere, Liste(Reel)}. Le domaine de la sortie est O1 = {Entier, Liste(Entier)}.

Figure 27 : Représentation d'un module à entrée binaire et sortie unaire.

3.9 Vérification des types

Avant de pouvoir connecter les modules entre eux, il est important de vérifier les trois conditions suivantes :

1. Le premier module comporte une sortie.

2. Le second module comporte une entrée.

3. Il faut que le domaine de l'entrée du premier module et du second module soient identiques. Par exemple, la sortie du module M1 O1 = {Entier, Entier} et l'entrée du module M2 I2 = {Entier, Entier}. Les deux domaines sont identiques. Ils peuvent être connectés comme à la figure 28. Si la conception est faite manuellement, la personne qui conçoit la chaîne de traitements devra s'assurer que les modules soient compatibles. Si on utilise un logiciel, ce dernier fera la vérification des types pour l'utilisateur. Durant tous les exemples sur la représentation des chaînes de traitements qui suivront, nous prendrons pour acquis que les domaines de tous les entrées et sorties sont conformes et qu'ils peuvent être connectés.

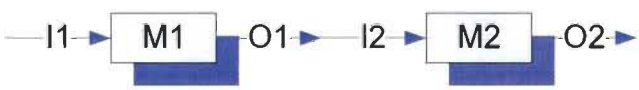


Figure 28 : Représentation de deux modules connectés ayant le même domaine.

3.10 Rappelons certaines notions.

Pour former des expressions combinatoires, nous allons utiliser des éléments de la logique combinatoire. Le tableau I présente les schémas que nous utiliserons durant la conception des chaînes. Nous présentons seulement ceux que nous utiliserons.

Tableau I

Tableau des combinateurs utilisés dans les expressions combinatoires

Combinateur	Description	Règle de la réduction
B	Composition	B f g x → f (g x)
C	Permutation	C x y z → x z y
Φ	Coordination	Φ x y z u → x (y u) (z u)

3.11 Chaîne de traitements en série

3.11.1 Cas d'un seul module

Nous avons un module M1 comme à la figure 29. Ce module comprend une entrée I1 et une sortie O1. Ce module est unique dans la chaîne de traitements. Nous n'avons besoin d'aucun des combinateurs pour le joindre à la chaîne. Ce module prend une entrée et produit une sortie. L'expression combinatoire s'écrit comme cela : O1 = M1 I1. Les entrées seront toujours à droite complètement. M1 doit produire la sortie O1. Nous pouvons remarquer que la notation est empruntée du paradigme fonctionnel y = f (x). y, est représenté par notre O1. f est la fonction qui est représentée par M1 et I1 est notre variable x.

Figure 29 : Cas d'un module simple O1 = M1 I1.

Le module peut avoir plusieurs entrées. Ces entrées sont obligatoirement présentes dans l'expression combinatoire. Par exemple, un module qui contient deux entrées aura comme expression combinatoire O1 = M1 I1 I2 comme à la figure 30.

Figure 30 : Cas d'un module à deux entrées. O1 = M1 I1 I2.

Nous pouvons facilement généraliser ce cas. Nous ajoutons n entrées à la fin de notre expression combinatoire comme à la figure 31.

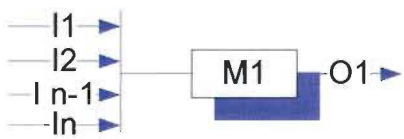


Figure 31 : Module ayant plusieurs entrées. O1 = M1 I1, I2, In-1, In.

3.11.2 Cas de deux modules connectés en série

Ce cas est le plus général. Nous avons deux modules M1 et M2 possédant chacun une entrée comme à la figure 32. Nous avons I1 pour le module M1 et I2 pour le module M2. Chacun possède une sortie O1 pour M1 et O2 pour M2. La construction des chaînes de traitements se fait toujours de droite à gauche. C'est-à-dire que nous prenons le module qui est le plus à droite en premier et nous construisons en revenant en arrière.

Nous prenons le module M2. Nous posons la première hypothèse que O2 = M2 I2. Nous avons notre début de la chaîne. Nous devons connecter notre module M1 au M2. En fait, ce que nous connectons, se sera l'entrée du module M2 avec la sortie du module M1. Nous reprenons notre première transformation et nous insérons le module M1 tel que O2 = M2 (M1 I1). Comme on peut le remarquer, I2 disparaît pour laisser la place au module M1 et nous n'insérons pas de sortie O1. M1 et I1

doivent se retrouver entre les parenthèses tant qu'ils ne sont pas combinés. La sortie étant connectée à l'entrée directement, le résultat sera récupéré automatiquement. Lorsque le module sera joint avec des combinateurs, nous pourrons enlever les parenthèses.

Nous devons combiner les deux modules. Pour cela, le combinateur de composition B est tout à fait approprié. Rappelons sa définition qui est B x y z→ x (y z). x (y z) ressemble à notre schéma M2 (M1 I1). La transformation sera B M2 M1 I1. Nous pouvons retirer les parenthèses. Nous avons notre première étape dans la logique combinatoire. La seconde étape est de calculer la distance du combinateur. Dans la chaîne de traitements, tous les combinateurs seront à gauche de la chaîne et tous les modules ainsi que les entrées seront à droite. Nous devons, maintenant, indiquer au combinateur sur qui il va agir. Le combinateur B combine le module M1 au M2. Le M1 se trouve à 1 de distance. La règle est la suivante : $X_n = B^n X$ et $X_0 = X$.

La distance → 0 1
O2 = B_0 M2 M1 I1

Nous commençons le compte immédiatement à la suite du combinateur B. Le compte débute à 0. Nous arrivons au M1 dont la distance est de 1. Puisque la règle nous dit de retrancher 1, alors la distance sera de 0.

Il nous reste la puissance. La puissance va nous servir pour retrouver les entrées qui sont rattachés au module. Le module, par sa définition, saura combien il a d'entrées. Pour savoir qui est rattaché avec qui, nous devons l'indiquer dans la chaîne combinatoire. La définition de la puissance est la suivante : $X^n = BXX^{(n-1)}$ et $X^1 = X$. En terme plus clair, notre module M1 comporte une entrée. La puissance appliquée au combinateur sera de 1. Voici notre chaîne de traitements complétée

avec la puissance et la distance : O2 = B_0^1 M2 M1 I1. Il est important de spécifier que la chaîne de traitements est O2 = B_0^1 M2 M1 I1.

Nous n'avons plus besoin des parenthèses. Le combinateur B remplace les parenthèses à distance. Toute cette chaîne va retourner une valeur dans O2. À partir de la chaîne combinatoire, nous pouvons retrouver notre chaîne initiale. Avec le combinateur de la composition, nous retrouvons grâce à la distance que celui-ci est appliqué à M1. Avec la puissance, nous savons que ce module a une seule entrée. La composition veut que M1 soit composée avec M2. La chaîne retrouvée est M2 (M1 I1).

Figure 32 : Deux modules en série. O2 = B_0^1 M2 M1 I1.

Résumé
Construction de la chaîne

1.	O2 = M2 I2	Hypothèse de départ
2.	O2 = M2 (M1 I1)	I2 remplacée par M1 I1, sortie O1 étant branché sur I2
3.	O2 = B_0^1 M2 M1 I1	Règle du combinateur B avec le combinateur de la distance et de puissance

Retrouver l'état initial

1.	O2 = B_0^1 M2 M1 I1	Notre expression combinatoire
2.	O2 = M2 (M1 I1)	Règle du combinateur B en appliquant la règle de la puissance et de la distance
3.	O2 = M2 I2	I2 retrouvée en détachant la sortie O1 du module M1

3.11.3 Cas de trois modules en série

Continuons avec le cas général et rajoutons un second module à la chaîne. Nous partons toujours de la gauche vers la droite. Notre liste de modules sera le M1 avec

une entrée I1 et une sortie O1 étant raccordée à l'entrée I2 du module M2 qui possède une sortie O2 étant raccordée elle-même à l'entrée I3 du module M3. La figure 33 représente cette chaîne. La sortie O3 nous donnera le résultat de notre chaîne. Nous posons, comme première hypothèse, que O3 = M3 I3. Nous pouvons continuer en posant que O3 = M3 (M2 I2). La sortie O2 étant connectée sur l'entrée I3 du module M3. Nous savons que la sortie de O1 du M1 est connectée sur l'entrée I2 de M2. Nous avons O3 = M3 (M2 (M1 I1)). Les parenthèses sont importantes. Sans elles, nous ne pourrions pas savoir les paramètres des combinateurs ni où appliquer les combinateurs.

Reprenons notre énoncé qui n'est pas encore une chaîne. Ajoutons-lui les combinateurs pour faire disparaître les parenthèses. O3 = M3 (M2 (M1 I1)), combinons pour commencer M2. La combinaison se fera exactement comme le premier cas de deux modules en série. Nous aurons O3 = B_0^1 M3 M2 (M1 I1). Nous pouvons remarquer que la première série de parenthèses est enlevée. Le combinateur B prend la relève du premier ensemble de parenthèses. La puissance du combinateur sera de 1. Cette puissance nous est donnée par le nombre de parenthèses suivant le module M2. Si nous avions eu une chaîne qui aurait été B_0^1 M3 M2 (M1 I1)(M4 I8)(M5 I9), nous aurions eu une puissance de 3. La distance se calcul selon la distance entre le B et le M2. Elle sera de 0 selon la règle. Combinons le module M1 avec le reste de la chaîne. Nous allons ajouter en premier le combinateur B. O3 = B B_0^1 M3 M2 (M1 I1). La puissance de ce combinateur sera de 1 puisque que le module possède une seule entrée. La chaîne est la suivante : O3 = $B^1 B_0^1$ M3 M2 (M1 I1). Il nous reste à attribuer la distance pour retirer la série de parenthèses. La distance du deuxième combinateur B sera de 2. Selon la règle de ce combinateur complexe, nous comptons les sauts et retirons 1. 3 étant la distance qui sépare le combinateur de son module, cela nous donne 2.

La distance → 0 1 2 3

O3 = B_2^1 B_0^1 M3 M2 M1 I1

La chaîne finalisée est $O3 = B_2^! B_0^! M3\ M2\ M1\ I1$. Elle n'a plus de parenthèses. Nous pouvons remarquer que les combinateurs sont à gauche de la chaîne. Les modules sont au centre et que les entrées sont à droite.

Figure 33 : Trois modules en série : $O3 = B_2^! B_0^! M3\ M2\ M1\ I1$.

Résumé
Construction de la chaîne

1. $O3 = M3\ I3$	Hypothèse de départ
2. $O3 = M3\ (\ M2\ I2\ )$	Incorporation du module M2 connecté à I3
3. $O3 = M3\ (\ M2\ (\ M1\ I1\ )\ )$	Incorporation du module M1 connecté à I2
4. $O3 = B_0^! M3\ M2\ (\ M1\ I1\ )$	Règle du combinateur B appliquée au module M2 avec les combinateurs complexes de distance et de puissance
5. $O3 = B_2^! B_0^! M3\ M2\ M1\ I1$	Règle du combinateur B appliquée au module M1 avec les combinateurs complexes de distance et de puissance

Retrouver l'état initial

1. $O3 = B_2^! B_0^! M3\ M2\ M1\ I1$	Notre expression combinatoire
2. $O3 = B_0^! M3\ M2\ (\ M1\ I1\ )$	Règle du combinateur B. Nous retrouvons notre module M1 entre les parenthèses
3. $O3 = M3\ (\ M2\ (\ M1\ I1\ )\ )$	Règle du combinateur B. Nous retrouvons notre module M2 entre les parenthèses
4. $O3 = M3\ (\ M2\ I2\ )$	Nous pouvons extraire M1 I1
5. $O3 = M3\ I3$	Extraction de M2 I2 et nous retrouvons notre module M3 Initial

3.11.4 Généralisation du cas en série.

Nous venons de voir le cas en série de deux, puis de trois modules. Un certain modèle tend à apparaître. Chaque ajout doit être appliqué à l'aide du combinateur B. Regardons les cas suivant :

Cas de un module O1 = M1 I1

Cas de deux modules O2 = B_0^1 M2 M1 I1

Cas de trois modules O3 = B_2^1 B_0^1 M3 M2 M1 I1

Cas de quatre modules O4 = $B_4^1 B_2^1$ B_0^1 M4 M3 M2 M1 I1

Nous pouvons remarquer qu'avec un seul module, nous n'avons pas de combinateur puisque le module est seul. Aussitôt que nous commençons à ajouter des modules, nous devons ajouter un combinateur B par module. Ce cas de généralisation concerne seulement les chaînes en séries où tous les modules sont uniformes. C'est-à-dire que tous les modules sont composés d'une entrée et d'une sortie. La puissance du combinateur B sera toujours de 1 puisque tous les modules possèdent qu'une seule entrée. La distance débutera à 0 pour le premier module combiné à la chaîne et incrémentera de deux à chaque fois. Nous devons spécifier que c'est le module combiné parce que le premier module de la chaîne n'a pas à être combiné avec un autre module au départ. Le nombre de combinateurs B sera toujours égal au nombre de modules moins un.

3.11.5 Cas d'un module avec aucune entrée et une sortie

Ce cas de module est particulier par le fait qu'il possède une sortie mais aucune entrée. La figure 34 représente ce cas. Ce module ne pourra jamais se retrouver à la fin d'une chaîne de traitements puisqu'aucun autre module ne peut se rattacher à une de ses entrées puisqu'il n'en a pas. Il pourra cependant se retrouver uniquement au

début d'une chaîne de traitements. Ce module peut raccorder sa sortie avec une entrée d'un autre module. Pour représenter ce module seul, son expression combinatoire est : O1 = M1. Ce module pourra se retrouver comme étant seul dans une chaîne de traitements. S'il est combiné, c'est lui qui généra les données de départ.

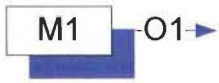


Figure 34 : Module avec aucune entrée et ayant une entrée. O1 = M1.

3.11.6 Cas de module avec aucune entrée et une sortie combiné

Ce cas ressemble au cas des modules en série avec une différence. Il y a aucune entrée pour le module M1. Quand nous combinerons le module M1 avec le module M2, le combinateur aura une puissance de 0. L'expression combinatoire suivante représente bien cette chaîne : $O2 = B_0^0$ M2 M1. La distance se calcul normalement. Ce qui change, c'est la puissance. Si vous avez bien suivi, j'ai dit que le module M1 se retrouvait en début de chaîne. Pourtant, sur la figure 35, il est en premier et sous la forme combinatoire, il se retrouve en dernier. Pour lire la liste de modules de la chaîne combinatoire, il faut toujours commencer par la droite vers la gauche.

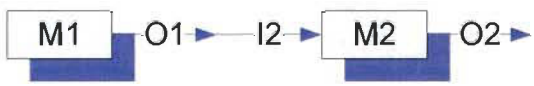


Figure 35 : Deux modules en série particuliers : $O2 = B_0^0$ M2 M1.

3.11.7 Cas d'un module avec une entrée et aucune sortie.

Ce cas de module est l'inverse du précédant. Au lieu d'une seule sortie et aucune entrée, nous nous retrouvons avec une seule entrée et aucune sortie. Ce module peut opérer seul ou dans une chaîne. Dans le cas qu'il serait seul, son expression combinatoire est : M1 I1. Ce module est dit autonome pour la sortie. Il ne donnera

aucune donnée à son successeur puisqu'il n'en possède pas. Par contre, par affichage d'une façon quelconque, comme à l'écran, il pourra présenter des données, des schémas, ...

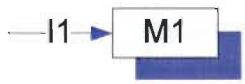


Figure 36 : Module avec une entrée et aucune sortie. M1 I1.

3.11.8 Cas de module avec une entrée et aucune sortie combiné.

Ce cas, ressemble au cas des chaînes de traitements en série. La différence, c'est qu'il n'y aura pas de sortie pour cette chaîne. La sortie pourra être un processus à l'écran ou avec d'autres procédés de sortie sur un média (disque dur, disquette,...). La seule restriction, c'est qu'il ne peut pas donner des données à aucun autre module. Il ne peut que les recevoir. De plus, en n'ayant aucune sortie, ce dernier ne pourra jamais avoir de module qui le suit. La figure 37 représente bien ce cas. Pour représenter un module M1 ayant une sortie et une entrée combinée avec un module M2 ayant une entrée et aucune sortie, l'expression combinatoire est : B_0^2 M2.M1 I1. Au contraire du dernier cas, ce module ayant aucune sortie se retrouvera à la fin de la chaîne. Dans l'expression combinatoire, il sera le premier module à droite, donc, le dernier à être lu.

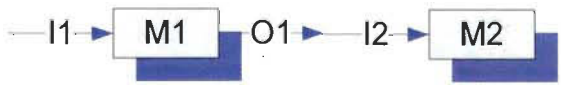


Figure 37 : Deux modules en série particulier B_0^2 M2 M1 I1.

3.11.9 Cas d'un module avec aucune entrée et aucune sortie.

Ce cas va créer un module indépendant comme à la figure 38. Il ne possède aucune entrée et aucune sortie. Il sera toujours seul dans sa chaîne de traitements. Il généra ses données et il devra les traiter lui-même. Son expression combinatoire est : M1.

Figure 38 : Module avec aucune entrée et aucune sortie. M1.

3.11.10 Chaîne de traitements en parallèle

Les modules contenant seulement une entrée ne pourront jamais former de chaînes en parallèles. Ces chaînes sont formées en introduisant des modules qui contiennent plus de deux entrées. Si de tels modules ne sont pas présents dans la chaîne, c'est que nous avons affaire uniquement à une chaîne en série.

3.11.11 Cas de deux modules module en parallèle

À la base, nous avons deux cas qui se distinguent. Le premier est le cas d'un module à deux entrées dont la première entrée est connectée à un autre module (figure 40). Le second cas est celui ou la sortie d'un module est connectée à la deuxième entrée du module (figure 39).

3.11.11.1 Premier cas : Module connecté à la première entrée

Ce premier cas est un module avec une seule entrée et une sortie qui est reliée à un module ayant deux entrées et une sortie. La notion de parallélisme ne veut pas obligatoirement dire que nous aurons des modules connectés comme dans la figure 39.

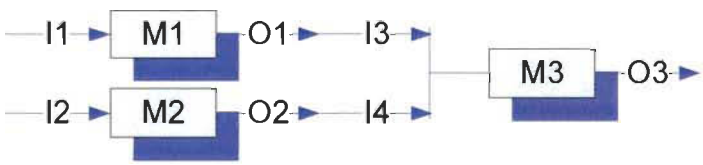


Figure 39 : Traitement en parallèle de trois modules.

Il se peut très bien qu'une des entrées du module M3 ne soit pas connectée. Il s'agit quand même d'un traitement en parallèle. Nous allons voir ce cas.

Nous avons un module M2 qui possède deux entrées I2 et I3 ainsi qu'une sortie O2. Le module M1, qui possède une entrée I1 et une sortie O1 qui est connectée au module M2 par son entrée I2. Nous débutons notre analyse en posant que O2 = M2 I2 I3. Déjà, nous voyons que ce module a deux entrées. Regardons le module M1. Sa sortie O1 est raccordée au module M2 par son entrée I2. Nous posons que O2 = M2 (M1 I1) I3. La différence avec les modules en série c'est qu'ici, nous avons deux entrées pour un module. Nous insérons dans la première entrée à un module connecté par sa sortie. Procédons avec les combinateurs. Commençons par relier ces deux modules avec le combinateur de composition B : O2 = B M2 M1 I1 I3. La puissance de ce combinateur sera 1. Bien que B soit relié à M1 et qu'il y ait deux entrées qui suivent, le module M1 comporte dans sa description une seule entrée. La distance sera de 0 puisqu'on applique la règle.

La distance → 0 1

$O2 = B_0^1$ M2 M1 I1 I3

Notre expression combinatoire sera $O2 = B_0^1$ M2 M1 I1 I3. Cette expression n'est pas tellement différente du cas en série que nous avons déjà vu. La seule différence est l'ajout d'une entrée qui est reliée à M2. Il se trouve à distance puisque nous avons un module avec son entrée entre les deux.

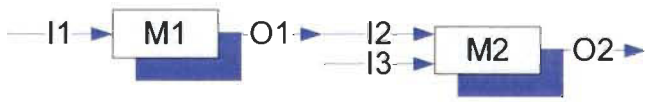


Figure 40 : Module connecté sur la première entrée. $O2 = B_0^1$ M2 M1 I1 I3.

Résumé
Construction de la chaîne

1. O2 = M2 I2 I3	Notre hypothèse de départ
2. O2 = M2 (M1 I1) I3	I2 remplacée par M1 I1, sortie O1 étant branché sur I2. I3 se distance de M2
3. O2 = B_0^1 M2 M1 I1 I3	Règle du combinateur B avec le combinateur de distance et de puissance

Retrouver l'état initial

1. O2 = B_1^0 M2 M1 I1 I3	Notre expression combinatoire
2. O2 = M2 (M1 I1) I3	Règle du combinateur B pour retrouver notre chaîne avec des parenthèses. Ici, il faut respecter notre distance et le nombre d'entrée rattachée à ce module.
3. O2 = M2 I2 I3	Nous retrouvons notre module M2 avec ses 2 entrées.

Nous commençons à constater qu'il est important de bien paramétrer le combinateur de composition B. Quand il s'agissait de modules en série avec juste une entrée, cela ne causait pas de problèmes. Ici, il faut qu'on retrouve le nombre exact d'entrées.

3.11.11.2 Deuxième cas Module connecté à la deuxième entrée

Notre second cas ressemble au précédent. La seule différence, c'est que le module se connecte sur la deuxième entrée plutôt que la première. Comme à la figure 39, nous avons un module M2 comportant une sortie O2 ainsi que deux entrées I2 et I3. Le module M1 contient une entrée I1 et une sortie O1 qui est connectée sur l'entrée I3 du module M2. Précédemment, c'était sur l'entrée I2 que nous étions. Voyons comment résoudre ce cas.

Posons en premier que O2 = M2 I2 I3. Introduisons notre second module M1 dans cette expression. O2 = M2 I2 (M1 I1). Comme vous pouvez le remarquer, le module M1 se retrouve à la fin avec son entrée. Appliquons notre combinateur B.

O2 = B M2 I2 M1 I1 . La puissance de ce combinateur sera de 1. Elle ne change pas. Nous combinons le module M1 avec M2 par la sortie O1 et l'entrée I3. Le nombre d'entrées de M1 est toujours un. Par contre, la distance changera. Si vous calculez bien, cette distance n'est pas la même.

La distance → 0 1 2
$O2 = B_1^! M2\ I2\ M1\ I1$

La distance après le calcul donne 1. Nous avons, entre M2 et M1, l'entrée I2 qui entre en compte. En appliquant la règle de la distance, cela nous donne 1. Nous obtenons l'expression combinatoire $O2 = B_1^! M2\ I2\ M1\ I1$. Un petit problème se pose ici. Nous avons dit plus haut que nous désirons avoir l'ensemble des combinateurs à gauche. Ensuite, nous avons l'ensemble des modules et pour terminer l'ensemble des entrées. Cette expression ne respecte pas cette règle. Il faut reculer le module M1 dans la chaîne.

Introduisons le combinateur de permutation C. Ce combinateur va être utile pour nous permettre de décaler nos modules à l'intérieur de l'expression. Chaque combinateur de composition C permutera de droite à gauche les modules d'un espace. Nous ne pouvons pas permuter des combinateurs. Ce combinateur ne possède pas de puissance. Il n'agit que sur le module. Il possède une distance pour savoir sur qui il agit. Introduisons notre combinateur de permutation à l'intérieur de notre expression combinatoire. Il s'ajoutera à la suite à gauche du dernier combinateur, en l'occurrence ici, le B. Voici l'expression $O2 = C\ B_1^! M2\ M1\ I2\ I1$. Le calcul de la distance se fera de la même façon.

La distance → 0 1 2
$O2 = C_1\ B_1^! M2\ M1\ I2\ I1$

La distance sera de 1. Notre nouvelle expression combinatoire sera $O2 = C_1\ B_1^1\ M2\ M1\ I2\ I1$.

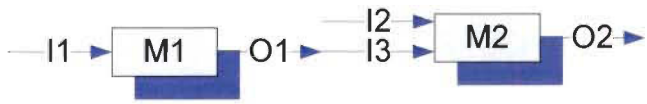


Figure 41 : Module connecté sur la deuxième entrée. $O2 = C_1\ B_1^1\ M2\ M1\ I2\ I1$.

Résumé
Construction de la chaîne

1.	$O2 = M2\ I2\ I3$	Notre hypothèse de départ
2.	$O2 = M2\ I2\ (\ M1\ I1\ )$	Nous remplaçons I3 par M1 I1. I3 du module M2 se connecte avec O1 de M1.
3.	$O2 = B_1^1\ M2\ I2\ M1\ I1$	Nous procédons à la composition de l'expression à l'aide du combinateur B et en appliquant les règles de puissance et de distance.
4.	$O2 = C_1\ B_1^1\ M2\ M1\ I2\ I1$	Nous procédons à la permutation avec le combinateur C sur le module M1 et appliquons la règle de la distance

Retrouver l'état initial

1.	$O2 = C_1\ B_1^1\ M2\ M1\ I2\ I1$	Notre expression combinatoire
2.	$O2 = B_1^1\ M2\ I2\ M1\ I1$	Nous permutons M1 de un élément vers la droite.
3.	$O2 = M2\ I2\ (\ M1\ I1\ )$	Règle du combinateur B. Nous retrouvons notre module M1 avec son entrée I1
4.	$O2 = M2\ I2\ I3$	Nous retrouvons notre module M2 avec ses entrées I2 et I3.

Nous venons d'introduire le combinateur de permutation C. Ce combinateur, quand nous en avons besoin avec le combinateur B, prend toujours la même distance que ce dernier. Regardons un cas avec un module contenant 4 entrées comme dans la figure 42.

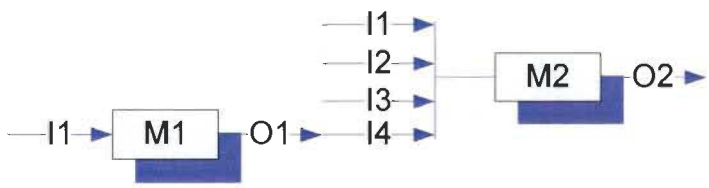


Figure 42 : Cas de module à plusieurs entrées.

Voici l'expression directement : $O2 = C_3\ C_3\ C_3\ B_3^1\ M2\ M1\ I2\ I3\ I4\ I1$. Comme nous pouvons le constater, nous avons permuté notre module M1 3 fois. Au départ, avant la combinaison nous avions $O2 = B_3^1\ M2\ I2\ I3\ I4\ M1\ I1$. Chacune des permutations nous donne 3 pour la distance. Voici la partie des permutations :

Permutation 1
La distance → 0 1 2 3 4
$O2 = C_3\ B_3^1\ M2\ I2\ I3\ M1\ I4\ I1$

Permutation 2
La distance → 0 1 2 3 4
$O2 = C_3\ C_3\ B_3^1\ M2\ I2\ M1\ I3\ I4\ I1$

Permutation 3
La distance → 0 1 2 3 4
$O2 = C_3\ C_3\ C_3\ B_3^1\ M2\ M1\ I2\ I3\ I4\ I1$

Si on remarque bien, chaque fois que nous ajoutons le combinateur de permutation, nous reculons notre M1 d'une place. La distance est toujours égale entre les deux. Il faut remarquer que la distance des combinateurs de permutation C est lié à la distance du combinateur de composition B.

3.11.12 Cas de modules en parallèle (3 modules)

Ce cas regroupe trois modules comme à la figure 43. Un des trois modules a deux entrées et les deux autres sont connectées sur ses entrées par leurs sorties. Nous avons notre premier module M3 comprenant une sortie O3 ainsi que deux entrées I3 et I4. Nous avons le module M1 comprenant une entrée I1 et une sortie O1 qui est connectée sur l'entrée I3 du module M3. Nous avons le module M2 qui comprend une entrée I1 et une sortie O2 qui est connectée sur l'entrée I4 du module M3.

Commençons à construire notre chaîne. Posons O3 = M3 I3 I4. Connectons le module M1 au module M3. O3 = M3 (M1 I1) I4. Ajoutons le module M2 à notre hypothèse. O3 = M3 (M1 I1) (M2 I2). Les parenthèses sont très importantes pour ne pas perdre la logique de la chaîne. Incorporons nos combinateurs. Commençons par M1. Nous devons le combiner à M3 : O3 = B M3 M1 I1 (M2 I2). Nous devons ajouter la distance qui est 0 selon la règle. La puissance sera 1 puisque M1 n'a qu'une seule entrée. Nous avons $O3 = B_0^1$ M3 M1 I1 (M2 I2). Combinons le module M2 avec le reste de la chaîne : $O3 = B\ B_0^1$ M3 M1 I1 M2 I2. La puissance sera encore ici de 1. La distance sera de 3 selon la règle.

La distance → 0 1 2 3 4
$O3 = B_3^1\ B_0^1$ M3 M1 I1 M2 I2

Nous avons $O3 = B_3^1\ B_0^1$ M3 M1 I1 M2 I2. Ce n'est pas tout. Nous remarquons qu'il y a une entrée entre le module M2 et le module M1. Il est important que le M2 soit à côté du M1. Décalons notre module M2 d'un espace avec le combinateur de permutation C. Nous avons $O3 = C_3\ B_3^1\ B_0^1$ M3 M1 M2 I1 I2. La distance du combinateur sera de 3. Nous avons maintenant notre expression combinatoire complète.

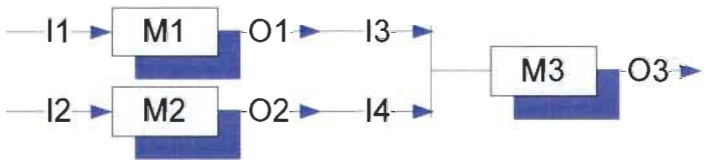


Figure 43 : Cas de modules en parallèle. O3 = C_3 B_3^1 B_0^1 M3 M1 M2 I1 I2 .

Résumé

Construction de la chaîne

1. O3 = M3 I3 I4	Notre hypothèse de départ.
2. O3 = M3 (M1 I1) I4	Nous remplaçons I3 par le module M1 I1. I3 du module M3 se connecte sur la sortie O1 du M1
3. O3 = M3 (M1 I1) (M2 I2)	Nous remplaçons I4 par le module M2 I2. I4 du module M3 se connecte à la sortie O2 du M2.
4. O3 = B_0^1 M3 M1 I1 (M2 I2)	Nous procédons à la composition de M3 avec M1 à l'aide du combinateur B et en appliquant les règles de puissance et de distance.
5. O3 = B_3^1 B_0^1 M3 M1 I1 M2 I2	Nous procédons à la composition de M3 avec M2 à l'aide du combinateur B et en appliquant les règles de puissance et de distance.
6. O3 = C_3 B_3^1 B_0^1 M3 M1 M2 I1 I2	Nous procédons au décalage du module M2 afin de conserver les règles des chaînes de traitement.

Retrouver l'état initial

1. O3 = C_3 B_3^1 B_0^1 M3 M1 M2 I1 I2	Notre expression combinatoire
2. O3 = B_3^1 B_0^1 M3 M1 I1 M2 I2	Nous permutons le module M2 en appliquant l'inverse de la permutation du combinateur C. Ce qui revient à dire qu'on va permuter vers la droite à la position 3.
3. O3 = B_0^1 M3 M1 I1 (M2 I2)	Nous retrouvons notre dernière composition en décomposant M2 I2 selon les règles de B et de la distance ainsi que la puissance.
4. O3 = M3 (M1 I1) (M2 I2)	Nous retrouvons notre première composition en décomposant M1 I1 selon les règles de B et de la distance ainsi que la puissance.
5. O3 = M3 (M1 I1) I4	Par l'ordre des entrées de M3, nous retrouvons l'entrée I4 et nous sortons M2 I2.
6. O3 = M3 I3 I4	Par l'ordre des entrées de M3, nous retrouvons l'entrée I3 et nous sortons M1 I1 et nous obtenons notre chaîne de départ.

3.11.13 Cas de module en parallèle (4 modules)

Regardons si nous pouvons généraliser ce cas. Nous allons faire une combinaison avec quatre modules. Nous avons un premier module M4 qui possède trois entrées I4, I5, I6 et qui possède une sortie. Nous avons le module M1 qui possède une entrée I1 et dont la sortie O1 est connectée sur l'entrée I4 du module M4. Nous avons le module M2 qui possède une entrée I2 et dont la sortie O2 est connectée sur I5 de M4. Nous avons le module M3 qui possède une entrée I3 et dont la sortie O3 est connectée sur I6 de M4. La figure 42 nous présente bien le cas.

Posons notre première hypothèse. Nous avons O4 = M4 I4 I5 I6. Entrons chacun des modules : M1, M2 et M3. O4 = M4 (M1 I1) I5 I6, O4 = M4 (M1 I1) (M2 I2) I6, O4 = M4 (M1 I1) (M2 I2) (M3 I3). Dès lors, nous pouvons commencer à construire notre expression combinatoire. Combinons notre M1 en premier. $O4 = B_0^l$ M4 M1 I1 (M2 I2) (M3 I3). Combinons ensuite M2 : $O4 = B_3^l\ B_0^l$ M4 M1 I1 M2 I2 (M3 I3). Nous devons décaler notre module M2 afin que la chaîne respecte les règles de l'expression combinatoire. Nous devons le permuter une seule fois : $O4 = C_3\ B_3^l\ B_0^l$ M4 M1 M2 I1 I2 (M3 I3). Nous devons maintenant intégrer le module M3 : $O4 = B_7^l\ C_3\ B_3^l\ B_0^l$ M4 M1 M2 I1 I2 M3 I3. Ici, nous devons permuter le module M3 deux fois afin que l'expression combinatoire respecte les règles : $O4 = C_7\ C_7\ B_7^l\ C_3\ B_3^l\ B_0^l$ M4 M1 M2 M3 I1 I2 I3. Nous avons notre expression combinatoire.

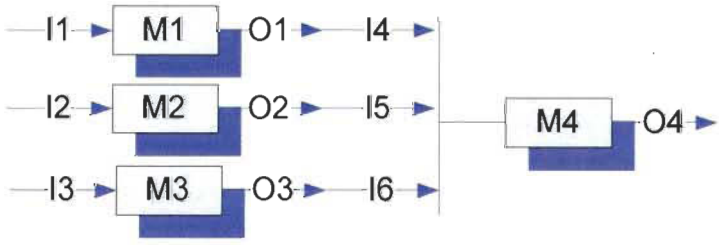


Figure 44 : Cas de quatre modules en parallèle.

$O4 = C_7\ C_7\ B_7^l\ C_3\ B_3^l\ B_0^l$ M4 M1 M2 M3 I1 I2 I3

Existe-t-il une généralité entre le cas de 3 et de 4 modules. L'expression combinatoire pour 3 modules est $O3 = C_3\ B_3^1\ B_0^1$ M3 M1 M2 I1 I2 et celle de 4 est $O4 = C_7\ C_7\ B_7^1\ C_3\ B_3^1\ B_0^1$ M4 M1 M2 M3 I1 I2 I3. Le début est identique. Avec celle de 4, nous avons ajouté un module de plus. Il est normal que l'expression soit plus lourde. Reprenons notre modèle de 4 modules et retirons le module M2 de façon à obtenir la figure 45.

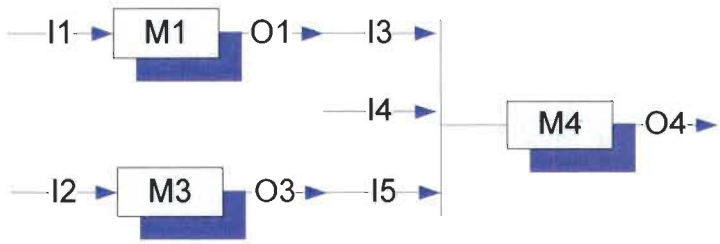


Figure 45 : Cas trois modules en parallèle dont une sortie libre.

$O4 = C_4\ C_4\ B_4^1\ B_0^1$ M4 M1 M3 I1 I4 I2

L'expression combinatoire de ce cas est $O4 = C_4\ C_4\ B_4^1\ B_0^1$ M4 M1 M3 I1 I4 I2. L'entrée I4 du module M4 est libre. Aucun module n'est connecté à cette dernière.

On ne peut pas généraliser le cas des chaînes de traitements en parallèles. Trop de variables rentrent en jeu. Premièrement, le nombre d'entrées de chacun des modules. Deuxièmement, les modules qui sont connectés ou pas aux autres. Nous pouvons cependant affirmer une chose. À chaque fois que nous combinons à l'aide du combinateur de composition B, nous devons regarder si nous devons faire des permutations avec le combinateur C. Le combinateur C est lié au combinateur B. Le combinateur C ne peut pas exister seul. Il a besoin d'un autre combinateur.

3.11.14 Cas d'un module coordonnant sa sortie.

Il y a un cas spécial qui consiste à distribuer une sortie d'un module à 2 ou plusieurs autres modules. Attention, ce cas ne peut pas représenter une chaîne de traitements. Quand nous distribuons la sortie vers 2 ou plusieurs modules, nous avons plus

qu'une sorties à la fin. Lorsque nous avons parlé des chaînes de traitements, nous avons dit qu'une chaîne peut avoir aucun ou une seule sortie. Pour que la distribution entre en jeu et puisse être intégrée dans une chaîne de traitements, il faut s'assurer qu'à la fin, nous n'ayons qu'une seule sortie.

Prenons l'exemple d'un module M1 ayant une entrée I1 et une sortie O1 ainsi que deux modules, M2 ayant une entrée I2 connectée à la sortie du module M1 et ayant une sortie O2. Il y a aussi M3 ayant une entrée I3 connectée à la sortie du module M1. La figure 46 représente bien cette chaîne. Nous devons émettre l'hypothèse que I2 = I3 = O1 = M1 I1. Ce qui est logique puisque l'entrée du module M2 I2 doit être identique à l'entrée du module M3 qui est I3. Ils doivent avoir les mêmes types. Ces deux entrées doivent être identiques à la sortie du module M1 O1. Si ce n'est pas le cas, ils ne pourront pas être connectées ensemble. Nous avons O1 = M1 I1. Là aussi c'est logique, cela se présente comme un module seul.

Pour construire la chaîne de traitements, on commence toujours de droite à gauche. Ici, nous commençons par le module M2. Nous posons O2 = M2 I2. Nous avons notre module M3. Nous posons alors O3 = M3 I3. Il faut réunir nos deux hypothèses. Nous ferons O2 O3 = (M2 I2) (M3 I3). Comment va-t-on introduire notre module M1? Ce que nous voulons faire, c'est de coordonner la sortie O1 du module M1 vers I2 et I3. Il serait tout à fait approprié d'utiliser le combinateur de coordination Φ. Avant d'appliquer ce combinateur, il faut entrer notre module M1 dans notre hypothèse. O2 O3 = (M2 (M1 I1)) (M3 (M1 I1)). Nous savons que O1 = M1 I1. Nous pouvons poser que O2 O3 = (M2 O1) (M3 O1) [M1 I1]. L'ajout de la sortie O1 va nous servir de coordination. Ensuite, entre crochets [ ], nous avons introduit M1 I1. M1 I1 est un élément de la chaîne, M1 va devoir faire partie de l'expression avec le combinateur Φ. La sortie sera à droite de la chaîne. Elle devra toujours se retrouver immédiatement après la liste des modules avant la liste des entrées.

Introduisons le nouveau combinateur. Nous avons O2 O3 = (M2 O1) (M3 O1) [M1 I1]. Les sorties devrons toujours être immédiatement à droite de leurs modules. Par

exemple, ici O1 vient immédiatement à droite de M2 et M3. Si cela n'avait pas été le cas, nous aurions dû décaler à gauche avec le combinateur de permutation C. Nous verrons un exemple plus tard. Le combinateur de coordination obéit à la règle $\Phi\, f\, g\, h\, x \rightarrow f\,(g\,x)\,(h\,x)$. Donc, nous avons O2 O3 = Φ M1 M2 M3 O1 I1. L'entrée de M1 ira immédiatement après la sortie O1. Est-ce qu'il y a une puissance? Bien sûr qu'il y a une puissance. Comme le combinateur de composition B, celui de coordination coordonne la sortie d'un module sur plusieurs autres entrées. Nous aurons la puissance de 2 puisque la coordination se fera sur deux modules. Il y a une distance. Nous devons savoir qui doit distribuer sa sortie. Dans notre cas, le combinateur de distance agira sur la sortie O1. La distance sera de 2 dans notre cas.

La distance → 0 1 2 3

O2 O3 = Φ_2^2 M1 M2 M3 O1 I1

Notre expression combinatoire sera O2 O3 = Φ_2^2 M1 M2 M3 O1 I1. Nous pouvons remarquer une chose. Notre combinateur indique que l'expression distribue la sortie sur les deux modules. Tout ceci grâce au combinateur de puissance. Si vous remarquez, les deux modules viennent immédiatement avant la sortie O1. Vous allez dire que nous retrouvons à la fois O1 et M1 dans l'expression. Ce fait est réel et nous ne pouvons rien y changer. La règle voudra que lors de l'exécution de la chaîne comprenant un tel combinateur, cette dernière devra exécuter une seule fois le module qui coordonne sa sortie.

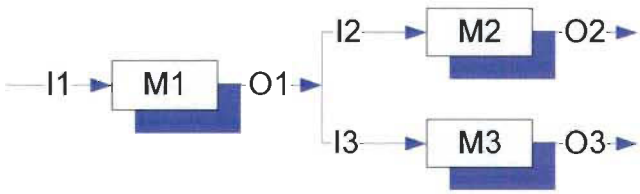


Figure 46 : Cas d'un module coordonnant sa sortie. O2 O3 = Φ_2^2 M1 M2 M3 O1 I1

Résumé

Construction de la chaîne

1. O2 = M2 I2	Notre hypothèse de départ
2. O3 = M3 I3	Notre second module
3. O2 O3 = (M2 I2) (M3 I3)	Nous regroupons sous une même expression les deux modules
4. O2 O3 = (M2 (M1 I1)) (M3 (M1 I1))	Nous posons que I2 = I3 = O1 = M1 I1, donc on remplace I2 et I3 par M1 I1
5. O2 O3 = (M2 O1) (M3 O1) [M1 I1]	Nous posons que O1 = M1 I1 et on remplace M1 I1 par O1
6. O2 O3 = Φ_2^2 M1 M2 M3 O1 I1	Nous appliquons la règle du combinateur Φ avec les règles de puissance et de distance qui sont importantes

Retrouver l'état initial

1. O2 O3 = Φ_2^2 M1 M2 M3 O1 I1	Notre expression combinatoire de départ
2. O2 O3 = (M2 O1) (M3 O1) [M1 I1]	Nous appliquons la règle du combinateur Φ où on retrouve notre forme. Ajoutons que nous adopterons de placer notre module de sortie à gauche de l'expression et non à droite comme dans la définition. Grâce au combinateur de distance, nous savons que O1 est le point de sortie. Avec celui de la puissance, nous savons que O1 agira sur 2 modules en arrière de lui.
3. O2 O3 = (M2 (M1 I1)) (M3 (M1 I1))	Nous posons que M1 I1 = O1
4. O2 O3 = (M2 I2) (M3 I3)	Nous savons que M2 et M3 ont comme entrée I2 et I3.**
5. O2 = M2 I2	Par définition
6. O3 = M3 I3	Par définition

** Nous savons que nos modules ont absolument au moins une entrée au départ. Nous le savons, car il y a une liste des modules avec leurs entrées et leurs sorties qui doivent suivre l'expression combinatoire. En plus, M2 et M3 ne comporte qu'une seule entrée. Si ces modules avaient possédé plus d'une entrée, celles qui n'auraient pas été utilisées auraient été dans l'expression combinatoire.

3.11.15 Cas d'un module coordonnant sa sortie.

Prenons un autre cas d'un module coordonnant sa sortie à l'intérieur d'une chaîne. Ce cas regroupera 4 modules. La figure 47 représente cette chaîne. Nous avons 4 modules. Le module M2 qui coordonnera sa sortie O2 possède aussi une entrée. Le module M4 ayant une sortie O4 ainsi qu'une entrée I5 qui est connectée au module M2 par sa sortie O2. Le module M3 possède une sortie O3 ainsi que deux entrées. La première entrée I4 qui sera connectée au module M2 par sa sortie O2. Il y a l'entrée I3 qui sera connectée à la sortie O1 du module M1 ayant lui aussi une entrée I1. Voyons l'expression combinatoire.

En premier, posons notre hypothèse. O3 = M3 I3 I4. Nous avons aussi O4 = M4 I5. Notre hypothèse de départ est O3 O4 = (M3 I3 I4) (M4 I5). Intégrons M1 : O3 O4 = (M3 (M1 I1) I4) (M4 I5). Nous sommes à l'étape de l'intégration de notre module qui coordonnera sa sortie. Posons que
O2 O3 = (M3 (M1 I1) (M2 I2)) (M3 (M2 I2)) puisque I4 = I5 = O2 = M2 I2. Remplaçons M2 I2 par O2 puisque O2 = M2 I2.
O2 O3 = (M3 (M1 I1) O2) (M3 O2) [M2 I2]. Nous pouvons remarquer que M3 est distancé de O2. Nous devons le rapprocher avec le combinateur de permutation C. Cela va nous donner O2 O3 = C_1(M3 (M1 O2 I1)) (M3 O2) [M2 I2]. Nous avons décalé de un. Nous venons d'entrer dans une parenthèse. C'est tout à fait normal puisque nous entrons dans notre processus combinatoire. Nous devons toujours garder l'expression de la chaîne correcte. Pour cela, nous devons décaler une seconde fois pour sortir de la parenthèse et se retrouver à côté de M3 : O2 O3 = C_1 C_1(M3 O2 (M1 I1)) (M3 O2) [M2 I2]. Maintenant, notre chaîne respecte la règle voulant que la sortie soit immédiatement à gauche du module. Nous pouvons procéder au reste du traitement. Coordonnons avec le combinateur Φ le module M2. Cela va nous donner O2 O3 = Φ C_1 C_1 M2 M3 (M1 I1) M4 O2 I2. Notre puissance sera de deux puisque la sortie est coordonnée sur deux entrées. La distance sera de 6 puisque :

La distance → 0 1 2 3 4 5 6 7

$O2\ O3 = \Phi_6^2\ C_1\ C_1\ M2\ M3\ (\ M1\ I1\ )\ M4\ O2\ I2$

M1 et I1 sont comptés même s'ils ne sont pas encore combinés officiellement. Il nous reste à combiner notre module M1 avec le combinateur de combinaison. $O2\ O3 = B_4^1\ \Phi_6^2\ C_1\ C_1\ M2\ M3\ M1\ I1\ M4\ O2\ I2$. Notre puissance est de 1 puisque ce dernier a une seule entrée. Sa distance est de 4. Vous pouvez remarquer que l'entrée I1 se retrouve au milieu de modules. Pour l'extraire de là, nous devons reculer le module M4 afin de déplacer I1. Cela nous donnera $O2\ O3 = C_6\ B_4^1\ \Phi_6^2\ C_1\ C_1\ M2\ M3\ M1\ M4\ I1\ O2\ I2$. Un autre problème se pose. Nous devons avoir notre sortie O2 immédiatement à la suite de la chaîne de modules. Transférons le O2 à gauche de I1 : $O2\ O3 = C_8\ C_6\ B_4^1\ \Phi_6^2\ C_1\ C_1\ M2\ M3\ M1\ M4\ O2\ I1\ I2$. Nous avons notre expression combinatoire complète et qui respecte les différentes règles de la conception.

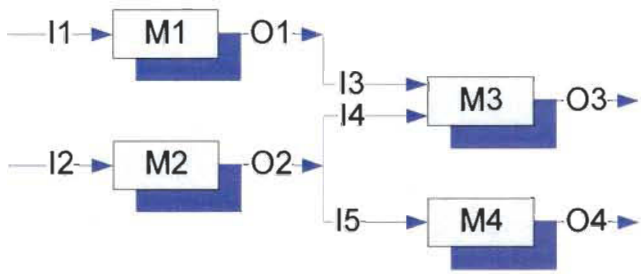


Figure 47 : Module M2 coordonnant sa sortie.

$O2\ O3 = C_8\ C_6\ B_4^1\ \Phi_6^2\ C_1\ C_1\ M2\ M3\ M1\ M4\ O2\ I1\ I2$

Résumons les règles que nous avons :

- La sortie coordonnée doit se trouver immédiatement à droite des modules à l'intérieur des parenthèses.
- Une fois que la coordination est faite, nous devons combiner le reste des modules à la chaîne coordonnée.
- La sortie doit se retrouver immédiatement à droite du dernier module.

Nous venons de voir les principales règles. Pourtant, quand on regarde le schéma modulaire de la figure 47, cela ne semble pas compliqué. Imaginez des chaînes comprenant une vingtaine de modules dont certains ont une sortie coordonnée à plusieurs entrées. Cela devient très lourd. La sortie O2 se retrouve à droite du dernier module. Cela n'est pas toujours le cas avec les modules coordonnés. Voyons un exemple particulier et expliquons pourquoi cela n'est pas toujours possible.

Nous pouvons dire que le combinateur de permutation C ne s'emploie pas seulement avec le combinateur de composition B. Nous pouvons aussi l'utiliser avec le combinateur de coordination Φ. Dans notre dernier exemple, le combinateur C est employé avant le combinateur de coordination. Il est important de dire que le combinateur de permutation est utilisé en relation avec d'autres combinateurs et jamais seul.

3.11.16 Cas d'une chaîne ayant un module avant un module coordonnée

Nous allons étudier un autre cas d'un module coordonné. Nous allons voir ce qui arrive lorsqu'un module est connecté à un module coordonné par sa sortie comme à la figure 48. Nous posons, au départ, que O3 = M3 I3 ainsi que O4 = M4 I4. Ce que nous allons retrouver est que O3 O4 = (M3 I3) (M4 I4). Nous pouvons poser que O3 O4 = (M3 (M2 I2)) (M4 (M2 I2)). Nous posons que I3 = I4 = O2 = M2 I2 . Nous posons que O3 O4 = (M3 O2) (M4 O2) [M2 I2]. Intégrons notre dernier

module M1 : O3 O4 = (M3 O2) (M4 O2) [M2 M1 I1]. Coordonnons notre module M2 : O3 O4 = Φ M2 M3 M4 O2 (M1 I1). Notre puissance sera de 2 puisqu'il y a deux modules qui se séparent la sortie de M2 et sa distance sera de 2. C'est ce qui nous donnera l'expression combinatoire O3 O4 = Φ_2^2 M2 M3 M4 O2 (M1 I1). Intégrons notre module M1 avec le combinateur de composition : O3 O4 = B Φ_2^2 M2 M3 M4 O2 M1 I1. La puissance du combinateur sera de 1 et sa distance sera de 4. O3 O4 = B_4^1 Φ_2^2 M2 M3 M4 O2 M1 I1. Que devons nous faire maintenant?

Nous devrions normalement permuter le module M1 afin qu'il se trouve à gauche de la sortie O2. Si nous faisons cela, nous allons créer un grave conflit dans l'ordre d'exécution des modules. Si vous prenez la partie des modules, lors de la création de la chaîne, en incluant les sorties, vous partez de droite vers la gauche et vous êtes capable d'exécuter tous les modules avec ses prédécesseurs dans l'ordre. Ici, si nous permutons le module M1, il s'exécuterait après le module M2 qui a besoin des données de M1 pour pouvoir s'exécuter. Nous devons établir comme règle que tous les modules qui sont nécessaires à l'exécution du module coordonnée doivent se trouver à droite de la sortie du module coordonnée. Avec cette nouvelle règle, nous avons notre chaîne combinatoire qui sera : O3 O4 = B_4^1 Φ_2^2 M2 M3 M4 O2 M1 I1.

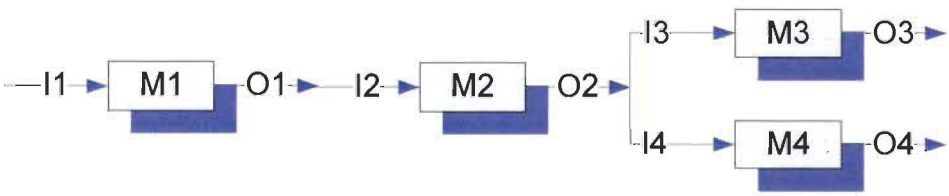


Figure 48 : Expression combinatoire avec module avec la coordination.

O3 O4 = B_4^1 Φ_2^2 M2 M3 M4 O2 M1 I1

Si nous avions rencontré un cas comme à la figure 49, il aurait été important que les modules soient à gauche des entrées. On aurait eu comme expression combinatoire : O4 O5 = C_7 B_7^1 B_4^1 Φ_2^2 M3 M4 M5 O3 M1 M2 I1 I2. Nous pouvons remarquer que

les modules M1 et M2 se retrouvent à gauche des entrées, mais à droite de la sortie O3. Les entrées, dans tous les cas, se retrouverons toujours à gauche du dernier module de l'expression combinatoire.

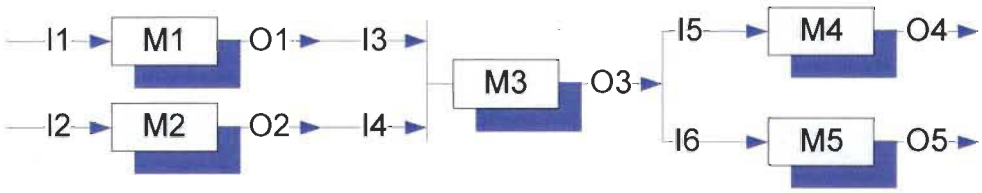


Figure 49 : Expression combinatoire avec plusieurs modules avant celui coordonné.

$$O4\ O5 = C_7\ B_7^l\ B_4^l\ \Phi_2^2\ M3\ M4\ M5\ O3\ M1\ M2\ I1\ I2$$

3.12 Regardons différents exemples de chaînes de traitements

3.12.1 Cas de neuf modules

Notre premier cas sera celui de neuf modules en série et en parallèle. Nous ne nommerons pas chacun des modules en détail. Reportez vous à la figure 50 pour avoir les détails.

Nous allons commencer par poser que O9 = M9 I11 I12 I13 I14.

Posons que O9 = M9 (M3 15 I 6) (M4 I9) I13 (M8 I10).

Continuons avec le prochain niveau pour insérer les modules M1, M2 et M3.
O9 = M9 (M3 (M1 I1) (M2 I2)) (M4 I9) I13 (M8 (M7 17 I8)).

Nous pouvons vite remarquer que la chaîne devient lourde. Continuons avec le dernier niveau en intégrant M5 et M6 :

O9 = M9 (M3 (M1 I1) (M2 I2)) (M4 I9) I13 (M8 (M7 (M5 I3) (M6 14))).

Notre chaîne est maintenant complétée. Nous pouvons commencer à la combiner. Nous allons commencer à combiner le module M3 avec M9 :

$O9 = B_0^2$ M9 M3 (M1 I1) (M2 I2) (M4 I9) I13 (M8 (M7 (M5 I3) (M6 14)))

Combinons M1 et M2 avec M3 :

$O9 = B_2^1\ B_0^2$ M9 M3 M1 I1 (M2 I2) (M4 I9) I13 (M8 (M7 (M5 I3) (M6 14)))

et

$O9 = B_5^1\ B_2^1\ B_0^2$ M9 M3 M1 I1 M2 I2 (M4 I9) I13 (M8 (M7 (M5 I3) (M6 14)))

Ici, nous pouvons remarquer que M2 à une entrée qui le sépare de M1, il faut le permuter :

$O9 = C_5\ B_5^1\ B_2^1\ B_0^2$ M9 M3 M1 M2 I1 I2 (M4 I9) I13 (M8 (M7 (M5 I3) (M6 14)))

.

Passons à la combinaison de M4 :

$O9 = B_9^1\ C_5\ B_5^1\ B_2^1\ B_0^2$ M9 M3 M1 M2 I1 I2 M4 I9 I13 (M8 (M7 (M5 I3) (M6 14)))

Nous pouvons remarquer que M4 est distancé de deux entrées. Permutons le pour le mettre avec les autres modules :

$O9 = C_9\ C_9\ B_9^1\ C_5\ B_5^1\ B_2^1\ B_0^2$ M9 M3 M1 M2 M4 I1 I2 I9 I13 (M8 (M7 (M5 I3) (M6 14)))

Notre chaîne combinatoire commence de plus en plus à prendre forme. Combinons le module M8 :

$O9 = B_{15}^1\ C_9\ C_9\ B_9^1\ C_5\ B_5^1\ B_2^1\ B_0^2$ M9 M3 M1 M2 M4 I1 I2 I9 I13 M8 (M7 (M5 I3) (M6 14))

Procédons à la permutation du module M8 afin qu'il soit avec les autres modules :

$O9 = C_{15}\ C_{15}\ C_{15}\ C_{15}\ B_{15}^1\ C_9\ C_9\ B_9^1\ C_5\ B_5^1\ B_2^1\ B_0^2$
M9 M3 M1 M2 M4 M8 I1 I2 I9 I13 (M7 (M5 I3) (M6 14))

Combinons le module M7 avec la chaîne :

$O9 = B_{21}^2\ C_{15}\ C_{15}\ C_{15}\ C_{15}\ B_{15}^1\ C_9\ C_9\ B_9^1\ C_5\ B_5^1\ B_2^1\ B_0^2$
M9 M3 M1 M2 M4 M8 I1 I2 I9 I13 M7 (M5 I3) (M6 14)

Permutons le module M7 pour qu'il soit avec les autres modules :

$O9 = C_{21}\ C_{21}\ C_{21}\ C_{21}\ B^2_{21}\ C_{15}\ C_{15}\ C_{15}\ C_{15}\ B^1_{15}\ C_9\ C_9\ B^1_9\ C_5\ B^1_5\ B^1_2\ B^2_0$
M9 M3 M1 M2 M4 M8 M7 I1 I2 I9 I13 (M5 I3) (M6 I4)

Combinons les modules M5 et M6 puis nous procéderons à la permutation de ces deux modules afin que la chaîne soit conforme :

Pour M5 :

La composition

$O9 = B^1_{27}\ C_{21}\ C_{21}\ C_{21}\ C_{21}\ B^2_{21}\ C_{15}\ C_{15}\ C_{15}\ C_{15}\ B^1_{15}\ C_9\ C_9\ B^1_9\ C_5\ B^1_5\ B^1_2\ B^2_0$
M9 M3 M1 M2 M4 M8 M7 I1 I2 I9 I13 M5 I3 (M6 I4)

La permutation

$O9 = C_{27}\ C_{27}\ C_{27}\ C_{27}\ B^1_{27}\ C_{21}\ C_{21}\ C_{21}\ C_{21}\ B^2_{21}\ C_{15}\ C_{15}\ C_{15}\ C_{15}\ B^1_{15}\ C_9\ C_9\ B^1_9\ C_5\ B^1_5\ B^1_2\ B^2_0$
M9 M3 M1 M2 M4 M8 M7 M5 I1 I2 I9 I13 I3 (M6 I4)

Pour M6 :

La composition

$O9 = B^1_{34}\ C_{27}\ C_{27}\ C_{27}\ C_{27}\ B^1_{27}\ C_{21}\ C_{21}\ C_{21}\ C_{21}\ B^2_{21}\ C_{15}\ C_{15}\ C_{15}\ C_{15}\ B^1_{15}\ C_9\ C_9\ B^1_9\ C_5\ B^1_5\ B^1_2\ B^2_0$
M9 M3 M1 M2 M4 M8 M7 M5 I1 I2 I9 I13 I3 M6 I4

La permutation

$O9 = C_{34}\ C_{34}\ C_{34}\ C_{34}\ C_{34}\ B^1_{34}\ C_{27}\ C_{27}\ C_{27}\ C_{27}\ B^1_{27}$
$C_{21}\ C_{21}\ C_{21}\ C_{21}\ B^2_{21}\ C_{15}\ C_{15}\ C_{15}\ C_{15}\ B^1_{15}\ C_9\ C_9\ B^1_9\ C_5\ B^1_5\ B^1_2\ B^2_0$
M9 M3 M1 M2 M4 M8 M7 M5 M6 I1 I2 I9 I13 I3 I4

Nous venons de donner l'expression combinatoire représentant la figure 50. On voit vite que la chaîne grandit. Particulièrement quand il y a beaucoup d'entrées vides

dans la chaîne et que certains modules se retrouvent à la fin de la chaîne pour la composition. Bien que cela soit lourd, c'est une excellente façon de pouvoir se partager des chaînes de traitements. Pour un être humain, il est très difficile de pouvoir construire de telles chaînes, mais pour une machine, cela est tout à fait réalisable.

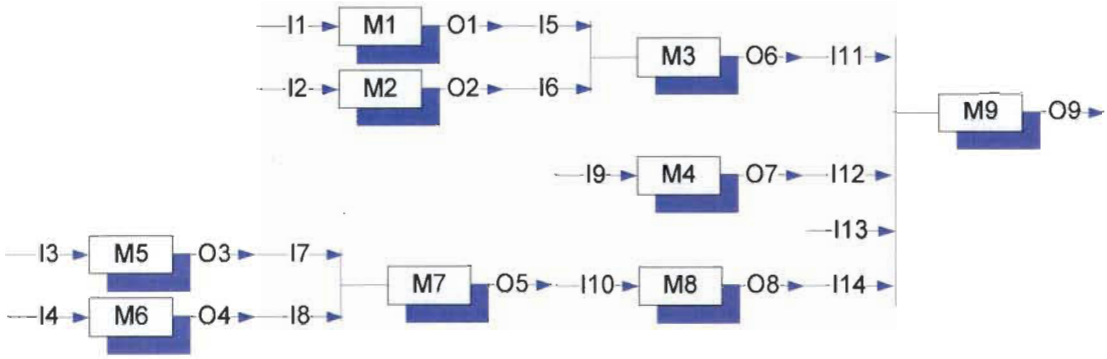


Figure 50 : Cas de 9 modules

$O9 = C_{34}\ C_{34}\ C_{34}\ C_{34}\ C_{34}\ B^{1}_{34}\ C_{27}\ C_{27}\ C_{27}\ C_{27}$
$B^{1}_{27}\ C_{21}\ C_{21}\ C_{21}\ C_{21}\ B^{2}_{21}\ C_{15}\ C_{15}\ C_{15}\ C_{15}\ B^{1}_{15}\ C_{9}\ C_{9}\ B^{1}_{9}\ C_{5}\ B^{1}_{5}\ B^{1}_{2}\ B^{2}_{0}$
M9 M3 M1 M2 M4 M8 M7 M5 M6 I1 I2 I9 I13 I3 I4

3.12.2 Cas de 7 modules

Nous allons voir un cas dont certains modules ne possèdent pas d'entrées. Nous allons étudier une chaîne comprenant 7 modules ainsi que 12 entrées. La figure 51 représente notre cas.

Posons notre hypothèse de départ O7 = M7 I8 I9 I10 I11 I12. Intégrons notre second niveau à la chaîne : O7 = M7 I8 I9 I10 (M5 I3 I4 I5) (M6 I6 I7).

Nous pouvons voir que nous avons déjà beaucoup d'entrées qui sont libres. Il y en a déjà 3. Intégrons le troisième niveau :
O7 = M7 I8 I9 I10 (M5 I3 I4 (M2)) (M6 (M3 I1) (M4 I2)).

Il nous reste plus que le module M1 à intégrer :

O7 = M7 I8 I9 I10 (M5 I3 I4 (M2)) (M6 (M3 I1) (M4 (M1))) .

Commençons à combiner notre premier module M5 :

O7 = B_3^3 M7 I8 I9 I10 M5 I3 I4 (M2) (M6 (M3 I1) (M4 (M1)))

On peut vite remarquer qu'il doit être permuté afin de se retrouver à côté de M7. Nous devons le faire permuter 3 fois (3 étant le nombre d'entrées entre les 2) :

O7 = C_3 C_3 C_3 B_3^3 M7 M5 I8 I9 I10 I3 I4 (M2) (M6 (M3 I1) (M4 (M1)))

Notre prochaine étape sera la combinaison de M2 avec la chaîne :

O7 = B_{10}^0 C_3 C_3 C_3 B_3^3 M7 M5 I8 I9 I10 I3 I4 M2 (M6 (M3 I1) (M4 (M1)))

Nous devons décaler le module M2 puisque ce dernier n'est pas à sa position. Vous pouvez remarquer que la puissance du combinateur B est à zéro puisque le module combiné M2 n'a pas d'entrées :

O7 = C_{10} C_{10} C_{10} C_{10} C_{10} B_{10}^0 C_3 C_3 C_3
B_3^3 M7 M5 M1 I8 I9 I10 I3 I4 (M6 (M3 I1) (M4 (M1)))

Passons à l'intégration du module M6 :

O7 = B_{17}^2 C_{10} C_{10} C_{10} C_{10} C_{10} B_{10}^0 C_3 C_3 C_3 B_3^3
M7 M5 M2 I8 I9 I10 I3 I4 M6 (M3 I1) (M4 (M1))

Nous devons procéder à la permutation de M6 pour lui faire rejoindre sa place :

O7 = C_{17} C_{17} C_{17} C_{17} C_{17} B_{17}^2 C_{10} C_{10} C_{10} C_{10} C_{10} B_{10}^0 C_3 C_3 C_3 B_3^3
M7 M5 M2 M6 I8 I9 I10 I3 I4 (M3 I1) (M4 (M1))

Procédons à la combinaison de M3 :

O7 = B_{24}^1 C_{17} C_{17} C_{17} C_{17} C_{17} B_{17}^2 C_{10} C_{10} C_{10} C_{10} C_{10} B_{10}^0 C_3 C_3 C_3 B_3^3
M7 M5 M2 M6 I8 I9 I10 I3 I4 M3 I1 (M4 (M1))

Procédons à la permutation de M3 :

$O7 = C_{24}\ C_{24}\ C_{24}\ C_{24}\ C_{24}\ B^{1}_{24}\ C_{17}\ C_{17}\ C_{17}\ C_{17}\ C_{17}\ B^{2}_{17}\ C_{10}\ C_{10}\ C_{10}\ C_{10}\ C_{10}\ B^{0}_{10}\ C_{3}\ C_{3}\ C_{3}\ B^{3}_{3}$
M7 M5 M2 M6 M3 I8 I9 I10 I3 I4 I1 (M4 (M1))

Procédons à l'intégration de M4 :

$O7 = B^{1}_{32}\ C_{24}\ C_{24}\ C_{24}\ C_{24}\ C_{24}\ B^{1}_{24}\ C_{17}\ C_{17}\ C_{17}\ C_{17}\ C_{17}$
$B^{2}_{17}\ C_{10}\ C_{10}\ C_{10}\ C_{10}\ C_{10}\ B^{0}_{10}\ C_{3}\ C_{3}\ C_{3}\ B^{3}_{3}$
M7 M5 M2 M6 M3 I8 I9 I10 I3 I4 I1 M4 (M1)

La permutation de M4 :

$O7 = C_{32}\ C_{32}\ C_{32}\ C_{32}\ C_{32}\ C_{32}\ B^{1}_{32}\ C_{24}\ C_{24}\ C_{24}\ C_{24}\ C_{24}\ B^{1}_{24}\ C_{17}\ C_{17}\ C_{17}\ C_{17}\ C_{17}$
$B^{2}_{17}\ C_{10}\ C_{10}\ C_{10}\ C_{10}\ C_{10}\ B^{0}_{10}\ C_{3}\ C_{3}\ C_{3}\ B^{3}_{3}$
M7 M5 M2 M6 M3 M4 I8 I9 I10 I3 I4 I1 (M1)

L'intégration de M1 :

$O7 = B^{0}_{40}\ C_{32}\ C_{32}\ C_{32}\ C_{32}\ C_{32}\ C_{32}\ B^{1}_{32}\ C_{24}\ C_{24}\ C_{24}\ C_{24}\ C_{24}\ B^{1}_{24}$
$C_{17}\ C_{17}\ C_{17}\ C_{17}\ C_{17}\ B^{2}_{17}\ C_{10}\ C_{10}\ C_{10}\ C_{10}\ C_{10}\ B^{0}_{10}\ C_{3}\ C_{3}\ C_{3}\ B^{3}_{3}$
M7 M5 M2 M6 M3 M4 I8 I9 I10 I3 I4 I1 M1

La permutation de M1 :

$O7 = C_{40}\ C_{40}\ C_{40}\ C_{40}\ C_{40}\ C_{40}\ B^{0}_{40}\ C_{32}\ C_{32}\ C_{32}\ C_{32}\ C_{32}\ C_{32}\ B^{1}_{32}\ C_{24}\ C_{24}\ C_{24}\ C_{24}\ C_{24}\ B^{1}_{24}$
$C_{17}\ C_{17}\ C_{17}\ C_{17}\ C_{17}\ B^{2}_{17}\ C_{10}\ C_{10}\ C_{10}\ C_{10}\ C_{10}\ B^{0}_{10}\ C_{3}\ C_{3}\ C_{3}\ B^{3}_{3}$
M7 M5 M2 M6 M3 M4 M1 I8 I9 I10 I3 I4 I1

Nous avons notre chaîne finale quand nous terminons de combiner et de permuter notre dernier module qui est M1. Vous pouvez remarquer que M1 ne contient pas de sortie. On indique zéro à la puissance du combinateur qui le combine à la chaîne de traitements. Malgré que le modèle soit plus petit en nombre de modules que le dernier modèle étudié, celui-ci a une expression combinatoire presque de la même grandeur que la dernière chaîne analysée.

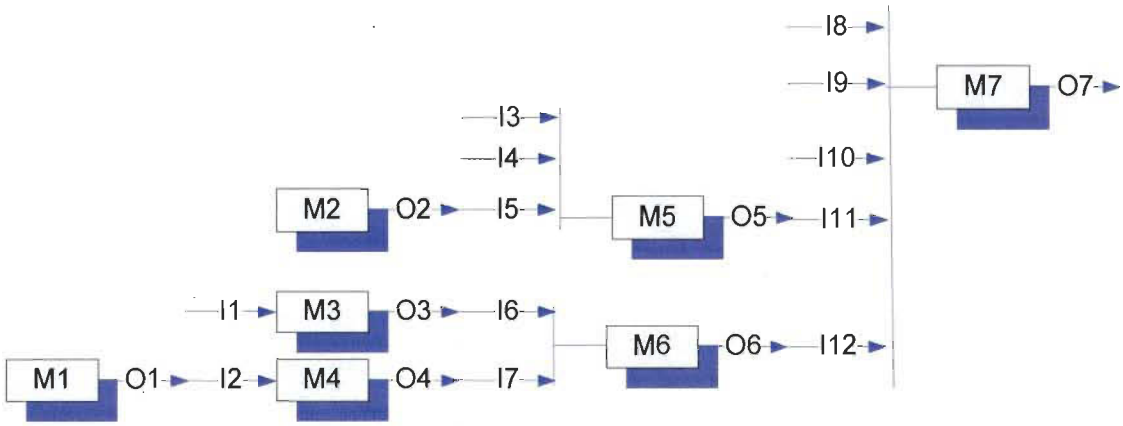


Figure 51 : Cas de 7 modules.

$O7 = C_{40}\ C_{40}\ C_{40}\ C_{40}\ C_{40}\ C_{40}\ B^{0}_{40}\ C_{32}\ C_{32}\ C_{32}\ C_{32}\ C_{32}\ C_{32}\ B^{1}_{32}\ C_{24}\ C_{24}\ C_{24}$
$C_{24}\ C_{24}\ B^{1}_{24}\ C_{17}\ C_{17}\ C_{17}\ C_{17}\ C_{17}\ B^{2}_{17}\ C_{10}\ C_{10}\ C_{10}\ C_{10}\ C_{10}\ B^{0}_{10}\ C_{3}\ C_{3}\ C_{3}\ B^{3}_{3}$
M7 M5 M2 M6 M3 M4 M1 I8 I9 I10 I3 I4 I1

3.12.3 Cas de 3 modules

Nous allons voir un cas où la chaîne de traitements n'aura aucune entrée et aucune sortie. Cette chaîne sera composée de trois modules. Elle est présentée à la figure 52.

Posons notre hypothèse de départ. Le module de départ n'a pas de sortie, alors il sera directe : M3 I3 I4. Ajoutons le module M1 et M2 : M3 (M1) (M2). Comme on peut le remarquer, M1 et M2 n'ont aucune entrée. Ils sont quand même entre parenthèses parce qu'ils ne sont pas encore combinés à la chaîne. Procédons à la première combinaison avec M1 : B^{0}_{0} M3 M1 (M2). La distance est de 0, là rien ne change. La puissance du combinateur est à 0. La raison est parce que le module M1 a zéro entrée. Combinons le module M2 avec la chaîne : B^{0}_{2} B^{0}_{0} M3 M1 M2. Encore ici, nous n'avons aucune entrée, la puissance sera de zéro. Nous avons déjà notre chaîne, elle est très simple.

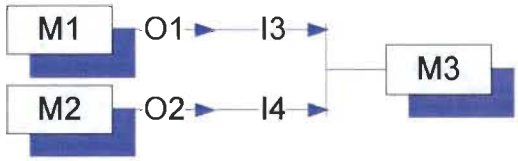


Figure 52 : Cas d'une chaîne à trois modules. B_2^0 B_0^0 M3 M1 M2

3.12.4 Cas d'une chaîne contenant une sortie coordonnée

Cette chaîne comprend un module ayant une sortie combinée avec deux modules. Le module est le M2. Il est important de dire que, contrairement à l'exemple, le combinateur de coordination ne se retrouvera pas en premier. Il faut commencer par intégrer les modules qui viennent avant. La figure 53 est le schéma que nous allons combiner en expression combinatoire.

Nous commençons par poser l'hypothèse de départ :

O8 = M8 I10 I11

Intégrons le deuxième niveau des modules, soit les modules M6 et M7 :

O8 = M8 (M6 I7 I8) (M7 I9)

Intégrons le troisième niveau qui est M3 et M5 :

O8 = M8 (M6 (M3 I3) (M4 I4 I5)) (M7 (M5 I6))

Intégrons le quatrième niveau qui est M2 :

O8 = M8 (M6 (M3 I3) (M4 (M1 I1) (M2 I2))) (M7 (M5 (M2 I2)))

Posons comme hypothèse que I5 = I6 = O2 = M2 I2. Nous pouvons alors substituer dans notre expression M2 I2 par O2 et sortir M2 I2 entre crochets [] :

O8 = M8 (M6 (M3 I3) (M4 (M1 I1) O2)) (M7 (M5 O2)) [M2 I2]

Combinons le module M6 avec la chaîne :

$O8 = B_0^2$ M8 M6 (M3 I3) (M4 (M1 I1) O2) (M7 (M5 O2)) [M2 I2]

Combinons maintenant le module M3 avec la chaîne :

$O8 = B_2^1\ B_0^2$ M8 M6 M3 I3 (M4 (M1 I1) O2) (M7 (M5 O2)) [M2 I2]

Nous devons ramener les sorties, ici O2, immédiatement à droite de leurs modules auxquels ils vont coordonnés leurs sorties :

$O8 = C_7\ B_2^1\ B_0^2$ M8 M6 M3 I3 (M4 (M1 O2 I1)) (M7 (M5 O2)) [M2 I2]

$O8 = C_7\ C_7\ B_2^1\ B_0^2$ M8 M6 M3 I3 (M4 O2 (M1 I1)) (M7 (M5 O2)) [M2 I2]

Combinons M7 puisque ce dernier doit être attaché avant que nous ajoutions le combinateur de coordination :

$O8 = B_{11}^1\ C_7\ C_7\ B_2^1\ B_0^2$ M8 M6 M3 I3 (M4 O2 (M1 I1)) M7 (M5 O2) [M2 I2]

Décalons le module M7 afin de le ramener avec les autres modules. En faisant cela, nous nous assurons que ce module sera exécuté après ses modules préalables :

$O8 = C_{11}\ C_{11}\ C_{11}\ C_{11}\ C_{11}\ B_{11}^1\ C_7\ C_7\ B_2^1\ B_0^2$
M8 M6 M3 M7 I3 (M4 O2 (M1 I1)) (M5 O2) [M2 I2]

Coordonnons la sortie de M2 avec M4 et M5 :

$O8 = \Phi_{19}^2\ C_{11}\ C_{11}\ C_{11}\ C_{11}\ C_{11}\ B_{11}^1\ C_7\ C_7\ B_2^1\ B_0^2$
M8 M6 M3 M7 I3 M2 M4 (M1 I1) M5 O2 I2

Nous pouvons maintenant combiner le module M1 :

$O8 = B_{17}^1\ \Phi_{19}^2\ C_{11}\ C_{11}\ C_{11}\ C_{11}\ C_{11}\ B_{11}^1\ C_7\ C_7\ B_2^1\ B_0^2$
M8 M6 M3 M7 I3 M2 M4 M1 I1 M5 O2 I2

Remarquez que M1, M2, M4 et M5 sont distancés par les entrées I1 et I3. Nous devons permuter ces modules afin de les ramener à droite des modules.

Pour M2 :

$O8 = C_{15}\ B^{1}_{17}\ \Phi^{2}_{19}\ C_{11}\ C_{11}\ C_{11}\ C_{11}\ C_{11}\ B^{1}_{11}\ C_{7}\ C_{7}\ B^{1}_{2}\ B^{2}_{0}$
M8 M6 M3 M7 M2 I3 M4 M1 I1 M5 O2 I2

Pour M4 :

$O8 = C_{17}\ C_{15}\ B^{1}_{17}\ \Phi^{2}_{19}\ C_{11}\ C_{11}\ C_{11}\ C_{11}\ C_{11}\ B^{1}_{11}\ C_{7}\ C_{7}\ B^{1}_{2}\ B^{2}_{0}$
M8 M6 M3 M7 M2 M4 I3 M1 I1 M5 O2 I2

Pour M1 :

$O8 = C_{19}\ C_{17}\ C_{15}\ B^{1}_{17}\ \Phi^{2}_{19}\ C_{11}\ C_{11}\ C_{11}\ C_{11}\ C_{11}\ B^{1}_{11}\ C_{7}\ C_{7}\ B^{1}_{2}\ B^{2}_{0}$
M8 M6 M3 M7 M2 M4 M1 I3 I1 M5 O2 I2

Pour M5 (deux permutations) :

$O8 = C_{22}\ C_{19}\ C_{17}\ C_{15}\ B^{1}_{17}\ \Phi^{2}_{19}\ C_{11}\ C_{11}\ C_{11}\ C_{11}\ C_{11}\ B^{1}_{11}\ C_{7}\ C_{7}\ B^{1}_{2}\ B^{2}_{0}$
M8 M6 M3 M7 M2 M4 M1 I3 M5 I1 O2 I2

$O8 = C_{22}\ C_{22}\ C_{19}\ C_{17}\ C_{15}\ B^{1}_{17}\ \Phi^{2}_{19}\ C_{11}\ C_{11}\ C_{11}\ C_{11}\ C_{11}\ B^{1}_{11}\ C_{7}\ C_{7}\ B^{1}_{2}\ B^{2}_{0}$
M8 M6 M3 M7 M2 M4 M1 M5 I3 I1 O2 I2

Nous devons aussi ramener la sortie à gauche des modules. (2 permutations) :

$O8 = C_{25}\ C_{22}\ C_{22}\ C_{19}\ C_{17}\ C_{15}\ B^{1}_{17}\ \Phi^{2}_{19}\ C_{11}\ C_{11}\ C_{11}\ C_{11}\ C_{11}\ B^{1}_{11}\ C_{7}\ C_{7}\ B^{1}_{2}\ B^{2}_{0}$
M8 M6 M3 M7 M2 M4 M1 M5 I3 O2 I1 I2

$O8 = C_{25}\ C_{25}\ C_{22}\ C_{22}\ C_{19}\ C_{17}\ C_{15}\ B^{1}_{17}\ \Phi^{2}_{19}\ C_{11}\ C_{11}\ C_{11}\ C_{11}\ C_{11}\ B^{1}_{11}\ C_{7}\ C_{7}\ B^{1}_{2}\ B^{2}_{0}$
M8 M6 M3 M7 M2 M4 M1 M5 O2 I3 I1 I2

Nous avons finalement notre expression combinatoire finale. Comme vous le voyez, à part pour les modules en série, on ne peut pas faire ressortir un modèle unique pour construire une chaîne combinatoire.

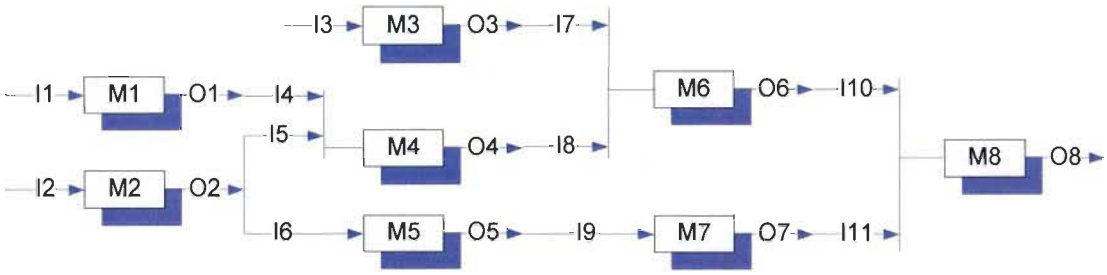


Figure 53 : Cas d'une chaîne avec un module coordonné.

$O8 = C_{25}\ C_{25}\ C_{22}\ C_{22}\ C_{19}\ C_{17}\ C_{15}\ B^{1}_{17}\ \Phi^{2}_{19}\ C_{11}\ C_{11}\ C_{11}\ C_{11}\ C_{11}\ B^{1}_{11}\ C_{7}\ C_{7}\ B^{1}_{2}\ B^{2}_{0}$

M8 M6 M3 M7 M2 M4 M1 M5 O2 I3 I1 I2

3.13 Exécution d'une chaîne de traitements.

Comme nous l'avons vu plus haut, l'ordre des modules est important pour l'exécution d'une chaîne de traitements. Si vous prenez la chaîne de traitements de la figure 54, vous pouvez remarquer qu'on ne peut pas tout exécuter pêle-mêle. Nous devons exécuter dans un ordre précis. La logique combinatoire nous aide à mettre de l'ordre dans tout ceci.

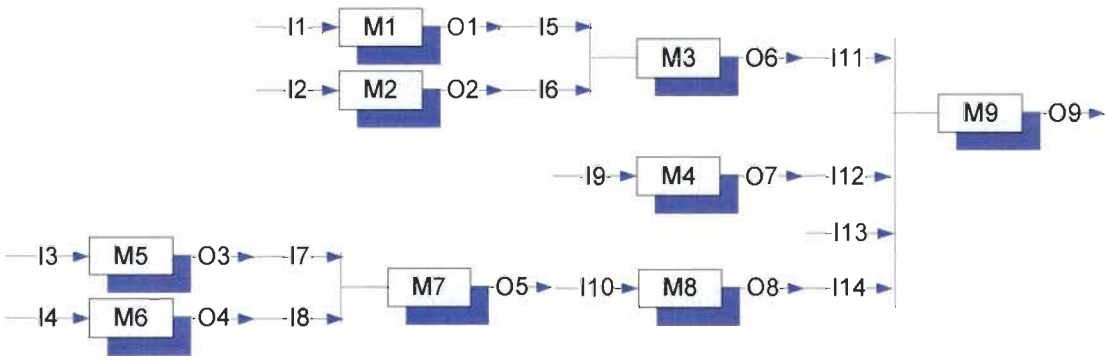


Figure 54 : Chaîne de traitements.

3.13.1 Exécution

Les chaînes combinatoires s'alourdissent quand le nombre de modules augmentent. Elles restent toujours structurées. Prenez par exemple l'expression combinatoire

suivante : $O4 = C_7\ C_7\ B_7^1\ C_3\ B_3^1\ B_0^1$ M4 M1 M2 M3 I1 I2 I3. Cette chaîne est structurée en 4 grandes parties. Nous pouvons dire trois puisque la partie à gauche de l'égalité n'est pas nécessaire à l'exécution de la chaîne.

La première partie de la chaîne est la liste des combinateurs. Tous les combinateurs se retrouvent dans cette partie sans exception. La seconde partie est celle des modules. Là, on retrouve tous les modules de l'expression. Nous pouvons cependant séparer cette partie en 2. Elle sera séparée dans un cas précis. Celui où nous avons la coordination. Il y aura les modules à gauche de la sortie et les modules à droite de la sortie. Finalement, nous avons la partie des entrées libres. Nous pouvons rajouter la partie à gauche de l'égalité. La figure 55 représente bien cette décortication.

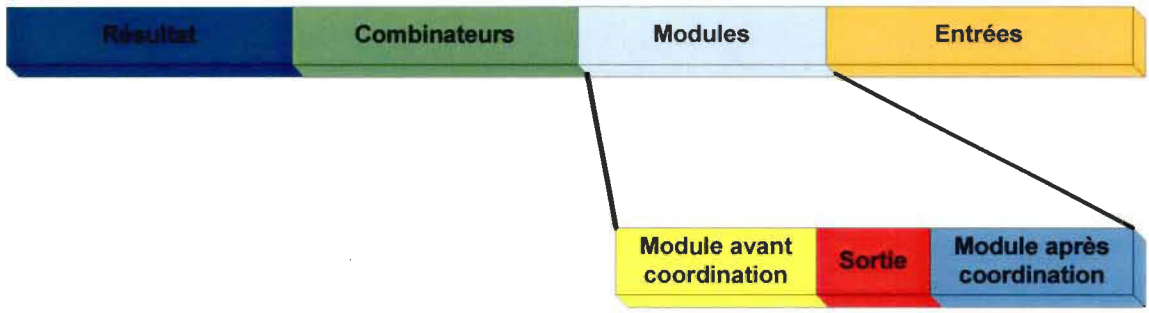


Figure 55 : Représentation d'une chaîne de traitements par partie.

3.13.2 Cas d'une chaîne de traitements série ou parallèle

Prenons une première expression combinatoire :

$O4 = C_7\ C_7\ B_7^1\ C_3\ B_3^1\ B_0^1$ M4 M1 M2 M3 I1 I2 I3.

Que faut-il faire pour procéder à l'exécution? Premièrement, il faut que les entrées soient remplies. Qu'entendons-nous par remplies? Il faut les remplir avec des données. Les entrées libres sont en fait les entrées appartenant aux modules dont les informations doivent être saisies manuellement. Première chose importante à faire,

c'est de combler les données dans chacune des entrées libres. Ici, nous avons I1, I2 et I3.

Deuxièmement, il s'agit de procéder à l'exécution des modules dans un ordre précis. Quel est cet ordre précis? Il s'agit de l'ordre dans laquelle la logique combinatoire a placée notre partie des modules. Cette partie se lit toujours de droite à gauche. Notre expression combinatoire pour la partie des modules nous a donnée M4 M1 M2 M3. Cet ordre va se lire comme suit : M3 en premier, M2 en deuxième, M1 en troisième et M4 en dernier. La figure 56 reprend cet ordre.

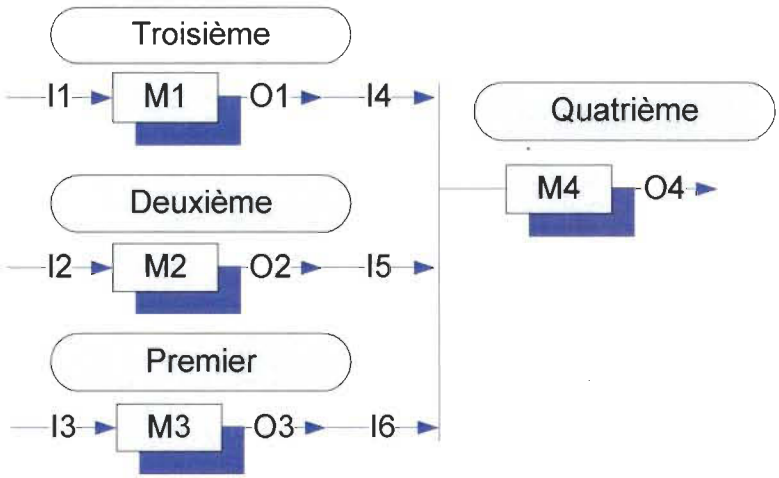


Figure 56 : Ordre d'exécution de la chaîne de traitements.

$O4 = C_7\ C_7\ B_7^1\ C_3\ B_3^1\ B_0^1$ M4 M1 M2 M3 I1 I2 I3

Nous savons que le résultat de la chaîne de traitements sera donné par le module M4. Il est logique que ce dernier ne soit pas le premier exécuté. Reculons d'un niveau. Ce qui nous mène au niveau des modules M1, M2 et M3. Qui, entre ces trois, devraient être exécutés en premier? Pour un de ces trois modules, outre le fait qu'ils ont tous une sortie de libre, y en a-t-il un qui a un module qui lè précède et qui doit être exécuté avant? La réponse est non. Logiquement, nous sommes dans un cas de parallélisme. Les trois modules pourraient être exécutés en même temps. Quand nous construisons notre chaîne combinatoire, l'ordre des entrées est préservé. Par exemple,

dans notre exemple nous posons que $O4 = M4\ I4\ I5\ I6$. Nous posons que $O4 = M4\ (M1\ I1)\ I5\ I6$. Nous pouvons remarquer que les entrées sont toujours dans l'ordre. Tout ce que nous déplaçons, ce sont les modules lors des permutations. Comme nous commençons de la droite vers la gauche pour l'exécution, il est normal que cela soit le module le plus à droite dans la chaîne qui est pour nous le M3. Par la suite, il y a le module M2 et M1 à exécuter. Finalement, lorsque les conditions préalables sont remplies pour le module M4, ce dernier peut être exécuté.

3.13.3 Cas d'un module à sortie coordonnée.

Prenons l'expression combinatoire $O3\ O4 = B_4^1\ \Phi_2^2\ M2\ M3\ M4\ O2\ M1\ I1$. Comment fait-on pour exécuter notre chaîne? Nous sommes dans le cas où nous avons une sortie dans la liste des modules. Qui nous donnera nos résultats? Ce sont les modules M3 et M4. Qui doit être exécuté au préalable des modules M3 et M4? Pour M3, nous avons le module M2 et pour M4 nous avons aussi le module M2. C'est le cas d'un module avec sa sortie coordonnée. Pouvons-nous exécuter le module M2 comme cela? Non, l'entrée du module est connectée. Il faut aller voir le module qui le précède. Ce dernier est le M1. Pouvons-nous l'exécuter directement? Non, nous devons remplir son entrée puisque cette dernière est libre. Lorsque M1 est exécuté, nous pouvons passer à M2 et M3 ainsi que M4.

L'ordre est définit dans la partie des modules de l'expression combinatoire. Cette partie comprend une sortie. L'ordre des modules nous donne comme liste M2 M3 M4 O2 M1. Le traitement se passera comme cela. Au préalable, les entrées libres sont remplies. Par la suite, nous exécutons M1. Nous rencontrons notre sortie O2. O2 est en fait par sa définition O2 = M2 I2. Cette définition est très importante. Elle va nous permette de faire l'exécution du module M2 immédiatement. Attention, si vous remarquez dans la partie modulaire, O2 apparaît déjà. Nous n'exécuterons pas deux fois le module M2. Par O2, nous allons exécuter M2. Nous suivons l'ordre des modules qui exécutera les modules M4 et M3. Nous tombons sur le module M2.

Nous ne l'exécutons pas. Ce dernier est déjà exécuté par O2. Voici l'ordre par rapport au schéma dans la figure 57 :

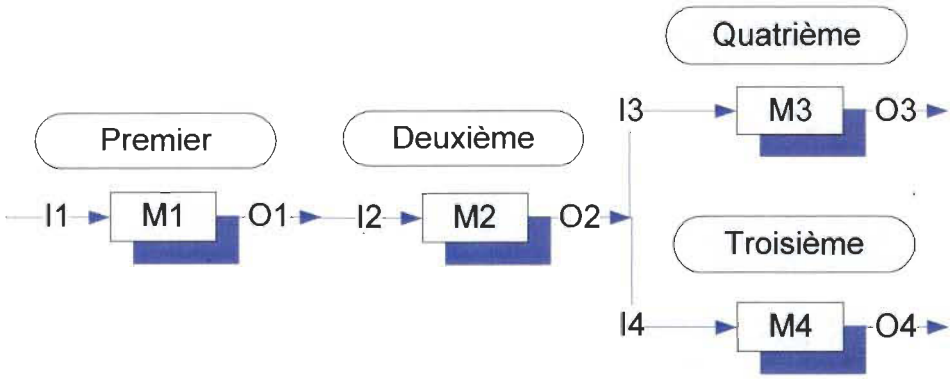


Figure 57 : Ordre d'exécution des modules avec la coordination.

$$O3\ O4 = B_4^1\ \Phi_2^2\ M2\ M3\ M4\ O2\ M1\ I1$$

Nous venons de voir la logique combinatoire. Elle est au centre de notre solution. Comme nous avons pu le constater, elle est utile tant à construire notre solution qu'à l'exécuter. La construction se fait un peu comme un jeu. Je vous l'accorde, un jeu qui peut être très compliqué plus on ajoute de nouveaux modules. Bien que la notation soit simple, plus on agrandit la chaîne et plus elle est dure à suivre pour l'être humain. Elle nous donne les balises à respecter pour connecter les modules entre eux. Même avec ces balises, cela deviens très lourd de construire des chaînes. Imaginez une chaîne contenant pas moins que 100 modules. Pour réussir à utiliser la logique combinatoire efficacement, l'ordinateur devra nous venir en l'aide. Dans le prochain chapitre, nous allons voir une application réelle qui fait tout le travail de la logique à notre place. Il nous reste plus qu'à construire notre chaîne en faisant du glissé-déposé. Un peu comme notre jeu de briques Lego.

CHAPITRE 4
IMPLÉMENTATION

Au dernier chapitre nous avons vu comment on compose une chaîne combinatoire. Nous avons parlé que plus on ajoute des modules à une chaîne et plus elle devient lourde à composer. Il nous faut une solution visuelle. Nous allons maintenant présenter une solution qui composera pour nous cette chaîne. Cet outil devra être inspiré du jeu des briques Lego. Cela veut dire que l'utilisateur s'amusera avec des pièces Lego. Nous voulons éviter que l'utilisateur se casse trop la tête pour comprendre l'outil. Cet outil devra procéder à la composition de la chaîne avec la logique combinatoire. Il ne sera donc pas nécessaire que l'utilisateur ait certaines bases dans ce domaine pour utiliser l'outil. L'outil se chargera de tous les côtés techniques pour laisser à l'utilisateur le droit de construire sa chaîne de traitement et lui éviter le plus possible les soucis techniques.

4.1 Ce que nous recherchons

Au point où nous en sommes, nous pouvons nous demander ce dont nous avons besoin. Nous avons besoin d'une plate-forme modulaire. Une plate-forme qui répond à plusieurs types d'utilisateurs en même temps tout en cachant les problèmes techniques. Nous allons devoir répondre aux besoins de trois types d'utilisateurs : le concepteur, le chercheur et finalement l'utilisateur final. Pour répondre à différents utilisateurs, nous devons fournir une plate-forme en 3 niveaux. Chacun des niveaux devra répondre aux besoins de chacun des trois types d'utilisateurs. La plate-forme est divisée en atelier, en laboratoire et finalement en application finale Regardons la description des trois niveaux.

4.1.1 Niveau atelier

L'atelier est la base de la plate-forme. C'est-elle qui contient les modules. Elle doit permettre d'ajouter, modifier et de supprimer des modules. Elle doit offrir la gestion des chaînes comme la sauvegarde ou la modification. Elle doit cacher tous les problèmes techniques. L'utilisateur doit être capable de se déplacer à travers un environnement complètement graphique avec du déplacer déposer (*drag and drop*). Aucune connaissance poussée en informatique ne doit être requise pour ce niveau.

4.1.2 Niveau laboratoire

Ce niveau permet de faire de la recherche. Une chaîne de traitements peut être techniquement utilisable en respectant tous les types. Par contre, du point de vue de la sémantique, celle-ci ne pourrait peut-être pas être viable. C'est pour cela que l'expert du domaine devra essayer des combinaisons, les tester et voir si la chaîne est correcte. Le chercheur devra s'amuser comme avec un jeu de briques afin de trouver la bonne combinaison. Ce niveau doit être entièrement graphique comme le niveau atelier.

4.1.3 Niveau application finale

Ce niveau ne fera pas partie de la plate-forme. Ce niveau est pour l'utilisateur final. Comme nous le mentionnons au début du chapitre 2 dans la description des modules, un module doit être à la fois dépendant et indépendant de la plate-forme. Cette dépendance est là pour réussir à faire la création des chaînes de traitements. Lorsque cette dernière est créée, elle est transformée dans un module qui est autant dépendant ainsi qu'indépendant. Cette indépendance pourra prendre la forme d'une application. Un module est une application aussi.

Tableau II

Les utilisateurs vis-à-vis leurs niveaux

Type d'utilisateur	Niveau
Concepteur	Atelier
Chercheur	Laboratoire
Utilisateur final	Application finale

Les grandes chaînes de traitements ne peuvent pas être composées par un être humain. Cela est très difficile. Pour cela, la plate-forme doit offrir une façon de concevoir graphiquement chacune des chaînes de traitements. En construisant la chaîne, la plate-forme construit l'expression combinatoire automatiquement et la présente à l'utilisateur. L'utilisateur n'interviendra pas dans la logique combinatoire. Le tableau II présente le type d'utilisateur et le niveau qu'il lui est rattaché.

4.2 Voyons ce que cette plate-forme représente en réalité.

SATIM (Système d'analyse et de traitement de l'information multidimensionnelle) est une plate-forme qui doit être une boîte à outils visuelle pour son utilisateur. Elle doit permettre au simple utilisateur ainsi qu'à l'expérimenté de l'utiliser. Pour cela, il faut un environnement graphique pour toutes les opérations.

Nous avons beaucoup parlé de plate-forme permettant au débutant de pouvoir l'utiliser. Pour qu'une telle plate-forme existe, il faut l'alimenter avec des modules. La conception des modules est du ressort du programmeur. Pour cela, nous devons inclure un outil pour permettre la création d'un module.

Notre application se sépara en deux grandes applications différentes. Celui de l'utilisation des modules ainsi que celui de la création des modules. Débutons avec l'application de l'utilisation des modules. L'architecture SATIM regroupe ces deux unités.

4.2.1 L'application à la base de SATIM

Cette application est la base. Elle se nomme l'Atelier. L'Atelier est une application complète et indépendante. L'écran de l'Atelier est présenté à la figure 58.

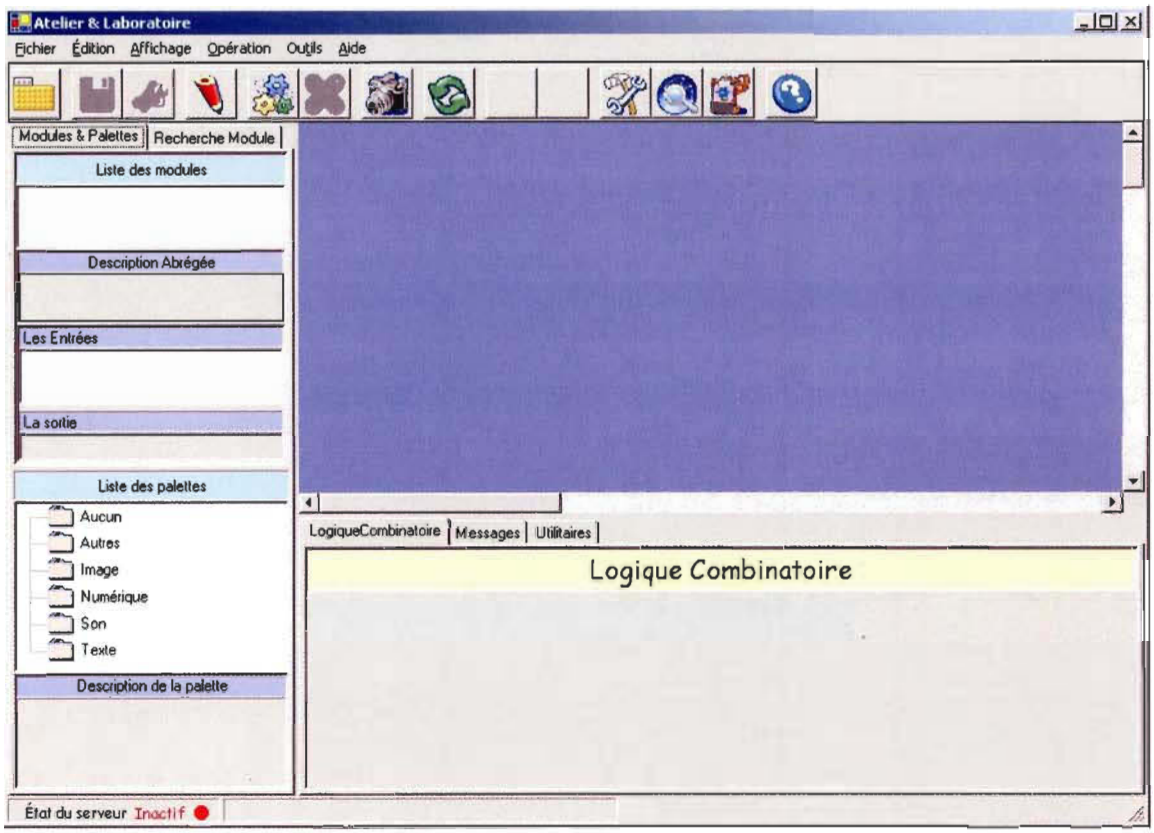


Figure 58 : Capture de l'écran principal de l'Atelier.

4.2.1.2 L'idée derrière la plate-forme

La plate-forme doit être autant accessible à des experts en informatique qu'à des nouveaux. Pour cela, on doit penser à une plate-forme graphique où l'utilisateur peut se promener dans une interface.

L'environnement graphique qui a été implémenté est du type glissé-déposé (*drag and drop*). Les utilisateurs vont glisser des formes sur une feuille de travail et les

connecter ensembles. Cette connexion se fait grâce à des câbles d'attaches. Ce principe est le même que la plupart des outils du type Microsoft Visual Visio. L'utilisateur n'a que très rarement à entrer du texte au clavier.

Notre plate-forme graphique sera séparée en 3 parties distingues. La figure 59 présente bien ces trois parties.

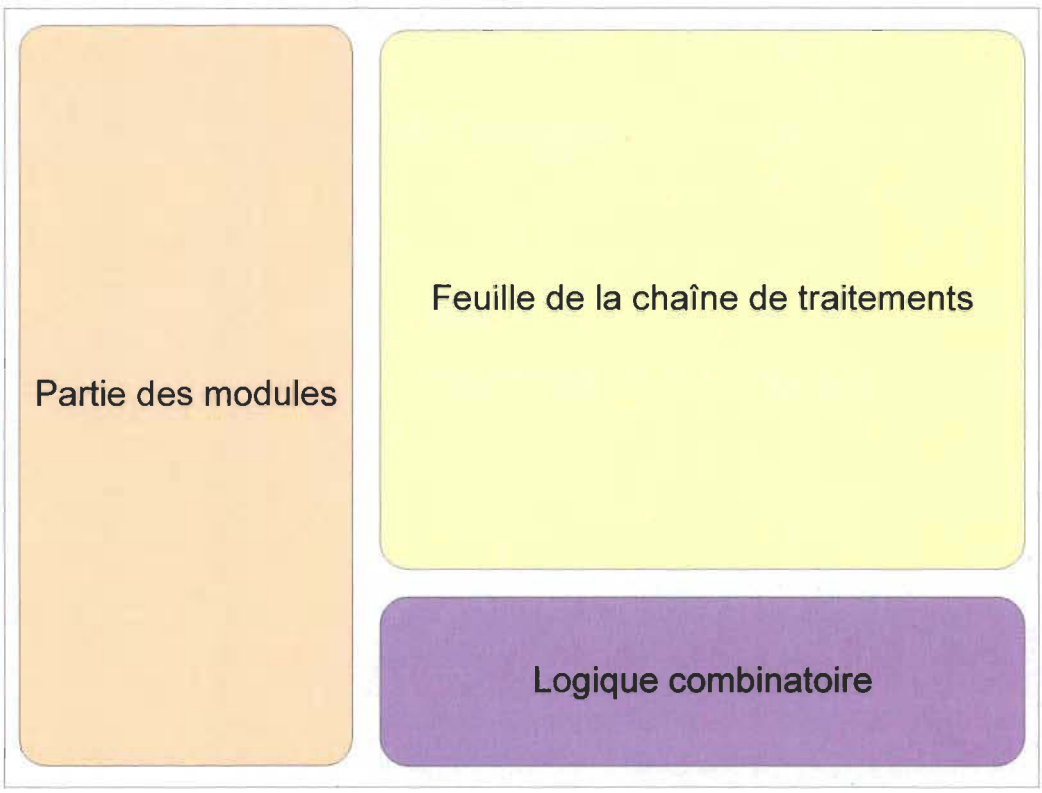


Figure 59 : Les trois parties de l'Atelier.

Nous avons notre partie des modules. Elle contient tous les modules disponibles dans la base de données de la plate-forme. La feuille de la chaîne de traitements est la partie de l'interface où les gens doivent faire glisser les modules de la *Partie des modules*. Cette partie est entièrement graphique et permet aux personnes de déplacer des objets. Elle permet de faire la connexion des modules. La partie logique combinatoire est celle qui construit, pour l'utilisateur, son expression combinatoire. Elle est construite par rapport au manipulation de l'utilisateur entre l'interaction de la *partie des modules* ainsi que la partie *feuille de la chaîne de traitement*. La chaîne est

construite par l'ordinateur et ne permet pas à l'utilisateur de faire des modifications manuelles à cette chaîne. L'utilisateur peut avoir plusieurs chaînes sur sa feuille de conception. Pour chaque chaîne, une expression combinatoire sera présentée.

4.2.2 Le constructeur de modules

Le constructeur de modules est un outil pour assister à la création des modules. Un module pourrait bien être créé en utilisant seulement le compilateur du Framework. Il faudrait que le créateur respecte à la ligne les normes de création de modules. Créer un module conforme aux spécifications n'est pas facile. Nous voulons avoir un module qui est autant dépendant qu'indépendant de la plate-forme. Pour cela, il doit assurer certains services. Dans la normalité des choses, cela serait la plate-forme elle-même qui s'en chargerait. Le constructeur de modules est là pour assister le programmeur dans sa création. Il laisse toute la latitude au programmeur de laisser libre cours à son imagination. Pour cela, l'application fait passer un questionnaire au programmeur afin de lui générer le code de base du module. Par la suite, le programmeur peut se concentrer à sa principale tâche. Cette tâche est la conception de l'action propre au module. Il sauve beaucoup de temps sur les questions techniques puisque le constructeur s'en occupe pour lui. Quand la création du module est terminée, la plate-forme de création des modules permet de faire la compilation du code. Elle permet de faire un test dans la plate-forme. La figure 60 résume bien l'architecture.

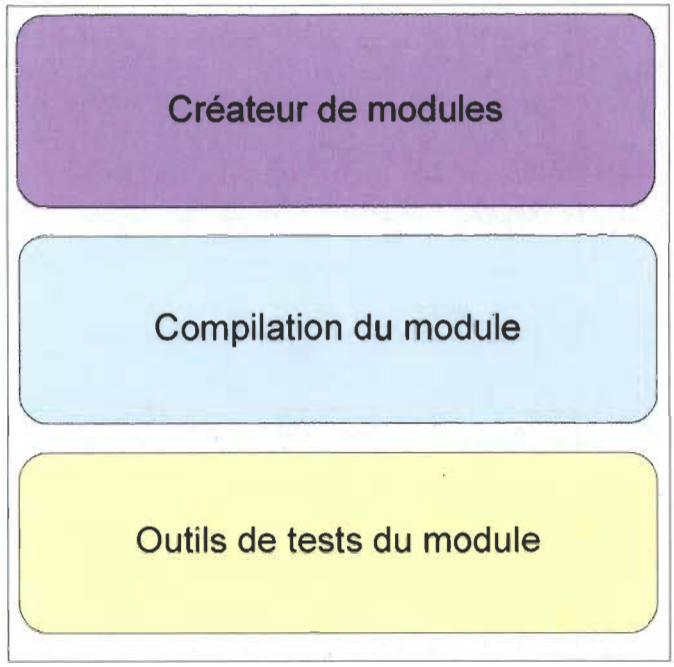


Figure 60 : Architecture du créateur de modules.

L'Atelier est construit en différentes parties qui sont en étroites relations. Le créateur de modules est conçu trois couches. Première couche : la création du module. Seconde couche : la compilation du module et finalement la couche des tests. Cette plate-forme est complémentaire à l'Atelier. Bien qu'elle ne soit pas totalement indispensable, elle apporte une grande aide au programmeur. Elle lui génère du code automatiquement.

4.3 Détails des deux plates-formes

4.3.1 L'Atelier

4.3.1.1 La gestion des modules

La gestion des modules passe par la gestion des palettes. Une palette est en fait une façon d'ordonner les modules. Imaginez avoir un millier de modules non triés. Cela serait vraiment interminable de faire une recherche à travers cela. Les palettes sont l'équivalent des dossiers dans votre classeur. La plate-forme SATIM offre des

palettes de base qui ne peuvent être effacées ou modifiées. Voici la liste de ces palettes :

- Image
- Numérique
- Son
- Texte

Pourquoi les palettes sont organisées ainsi? Plus haut, nous avons parlé que SATIM se définit comme étant un Système d'Analyse et de Traitement de l'information Multidimensionnelle. Cette information multidimensionnelle est en fait les données que nous devons traiter. Dans notre monde, nous avons un premier type qui est le texte. Le texte est un des domaines où la recherche est très forte. Quand on parle de numérique, nous avons les opérations de base comme que l'addition, la soustraction,... Il ne faut cependant pas oublier que les nombres vont bien plus loin que cela. Nous pouvons parler des nombres binaires, du chiffrement, de la géométrie, etc. Nous avons le son et les images. Depuis l'avènement de l'Internet, le son et les images ont commencés à occuper une très grande place dans la recherche. Est-ce vraiment possible me direz-vous de faire de la recherche sur cela? Et bien oui, des techniques existent. Ils représentent un domaine assez important pour avoir une axe de recherche. C'est la même chose que pour le son. Les gens ne pourront pas modifier ni ajouter des palettes à la base. Ils pourront en ajouter à celles qui existent déjà comme le démontre l'exemple de la figure 61. La palette images contient une sous palette Image Raster. Nous spécialisons notre palette.

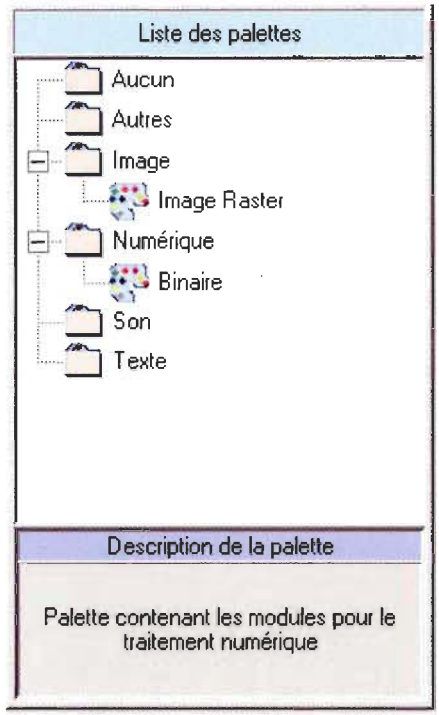


Figure 61 : Exemple des palettes

Les palettes peuvent contenir des palettes mais aussi des modules. Le module est le point fort de la plate-forme. Sans les modules, la plate-forme n'a aucune raison d'exister. Les modules doivent être tous ajoutés un à la suite des autres. Il n'y a pas moyen d'ajouter des modules en groupe. Plusieurs informations concernent un module. Nous avons le nom du fichier, le nom du module, la description de ce dernier, la liste des fichiers du module, le type de la configuration, le type des données, le type des entrées, le type de la sortie, les auteurs et la date de création. La figure 62 présente ces informations.

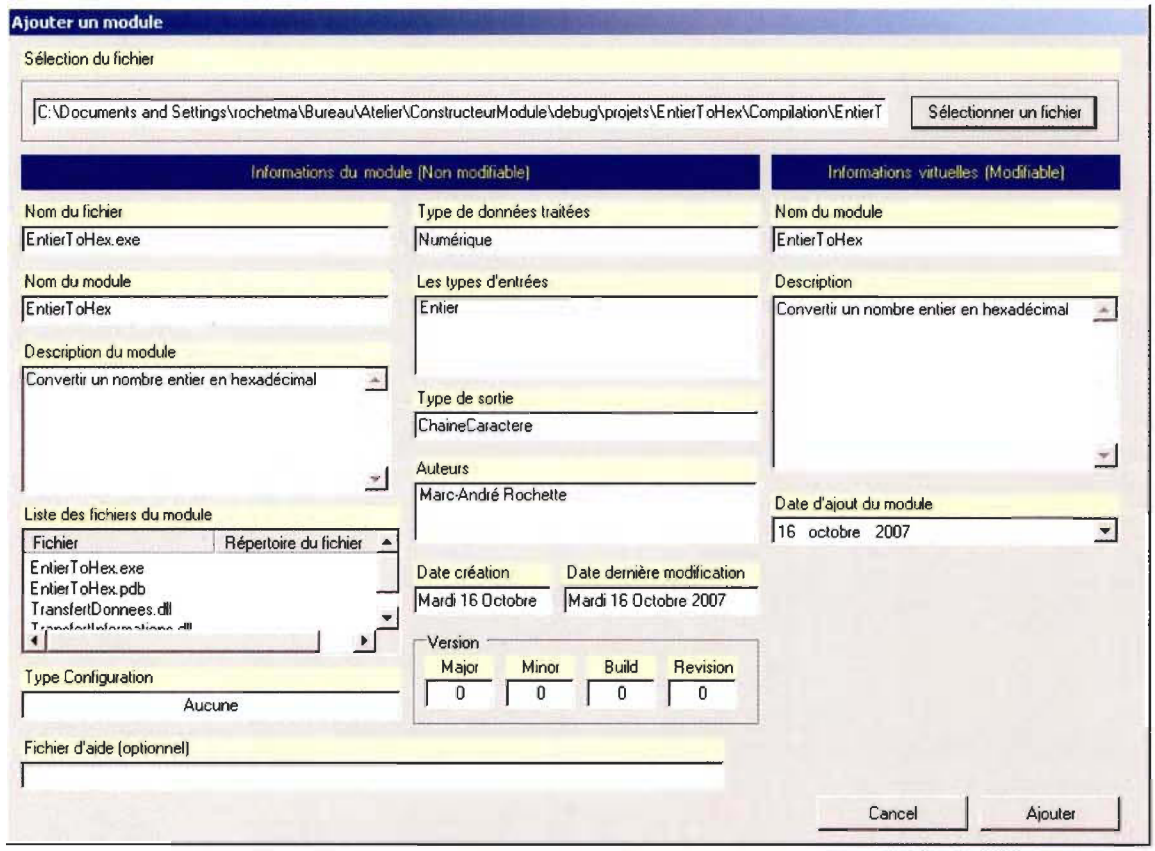


Figure 62 : Information sur un module.

Dans la partie de gauche, ce sont les informations venant directement du module. Dans la section de droite, ce sont des informations modifiables par l'utilisateur.

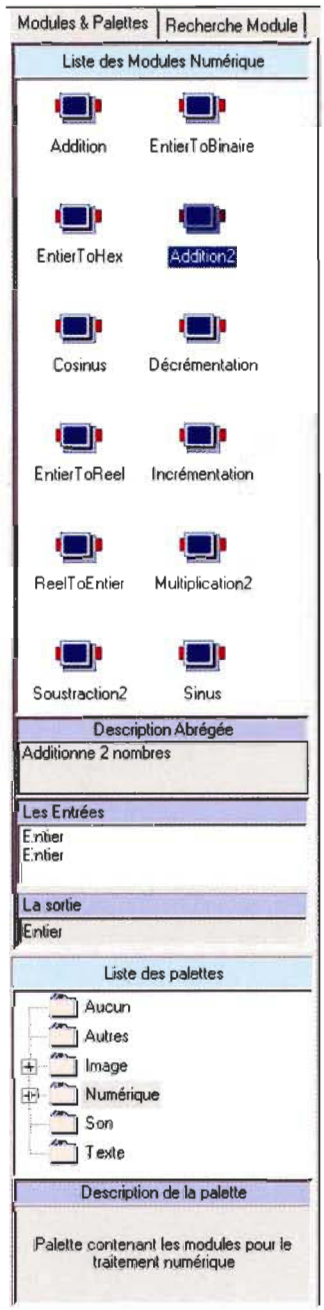


Figure 63 : Choix des modules.

La figure 63 présente la liste des palettes ainsi que la liste des modules compris dans la palette qui est sélectionnée. À l'écran, la personne a une brève description du module ainsi que ses entrées et sa sortie. S'il veut plus d'informations, il n'a qu'à faire afficher avec le bouton droit de la souris sur le module.

4.3.1.2 Le glissé déposé (*drag and drop*)

Le glissé-déposé (*drag and drop*) est un point central de la plate-forme. C'est ce qui rend la plate-forme plus conviviale pour l'utilisateur. Vous prenez des modules et vous les faites glisser jusqu'à l'endroit que vous désirez sur le schéma. La figure 64 présente bien ce qui ce produit.

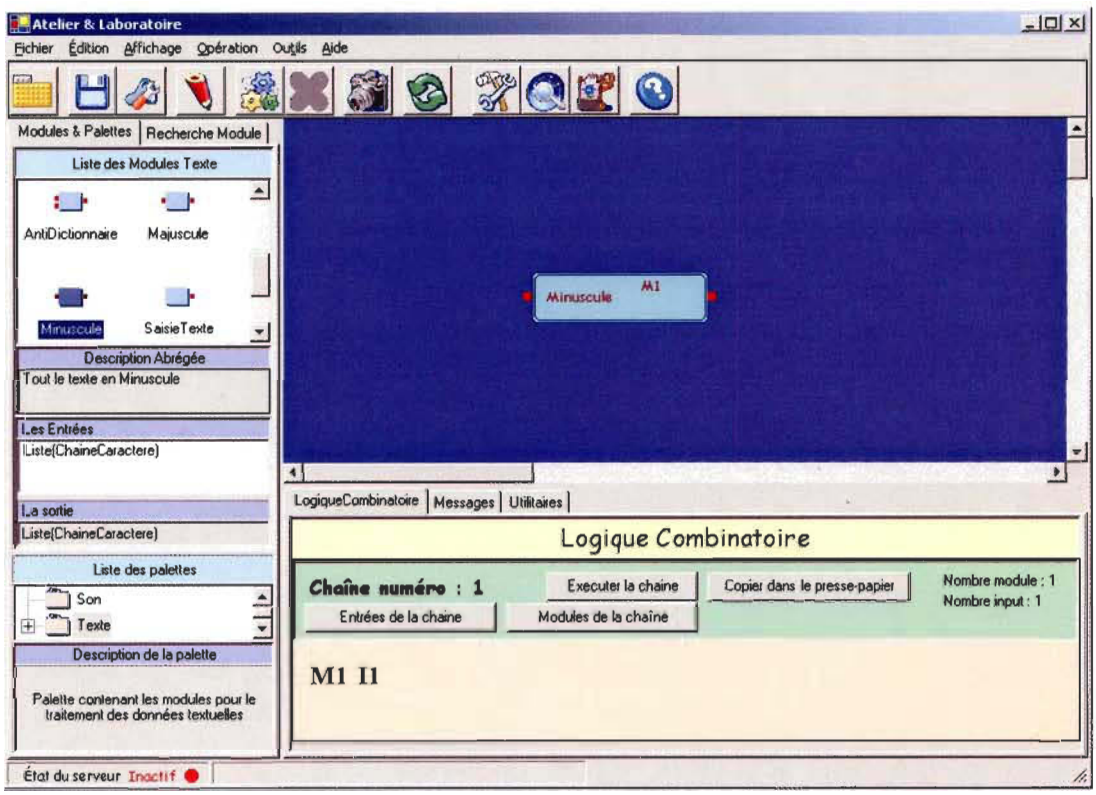


Figure 64 : Glissé déposé de la plate forme.

Il est important de dire que chacun des modules peut être ajouté plus d'une fois sur la feuille. Le nom ne rentrera pas en conflit avec la logique combinatoire. Il faut, dans une chaîne donnée, que tous les noms soient uniques. Pour cela, nous n'utiliserons pas les noms réels des modules mais des noms virtuels qui seront assignés par l'Atelier lorsque le module sera déposé sur la feuille. Le premier module aura le nom M1, le second M2 ainsi de suite.

4.3.1.3 La construction d'une chaîne

La construction d'une chaîne de traitements est en lien étroit avec le glissé-déposé (*drag and drop*). La principale propriété pour qu'une chaîne existe est qu'il y ait au minimum un module dans la chaîne. Aussitôt que nous déposons le premier module, nous avons un premier module. La partie de la logique combinatoire de l'Atelier entre en fonction. C'est elle qui va construire nos chaînes combinatoires. Nous disons nos, parce qu'il est possible d'en avoir plus que une. Si vous ajoutez un autre module qui n'est pas rattaché, cela va vous créer une deuxième chaîne comme dans la figure 65.

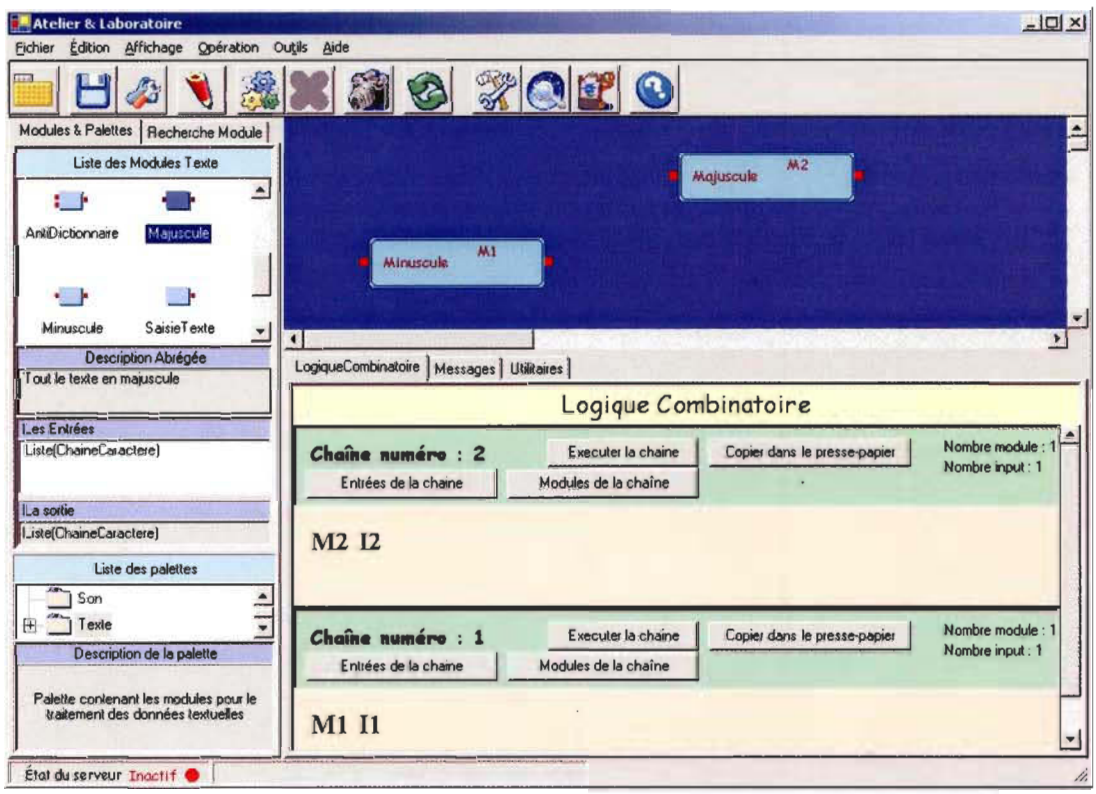


Figure 65 : Deux chaînes de traitements.

La figure 66 présente seulement le gestionnaire de la logique combinatoire. Ce gestionnaire va nous présenter notre chaîne à chaque étape de sa création.

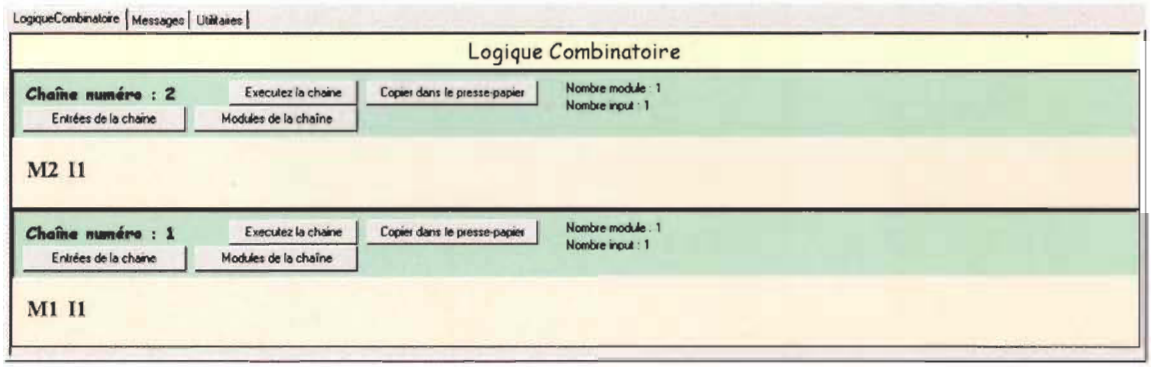


Figure 66 : Gestionnaire des chaînes combinatoires.

Si nous ne faisons qu'ajouter un module, alors nous avons une autre chaîne de traitements dans la feuille. Aussitôt que nous relions deux modules entre eux, on vient de fusionner deux chaînes entre elles. La figure 67 fusionne les deux chaînes dans une seule en série.

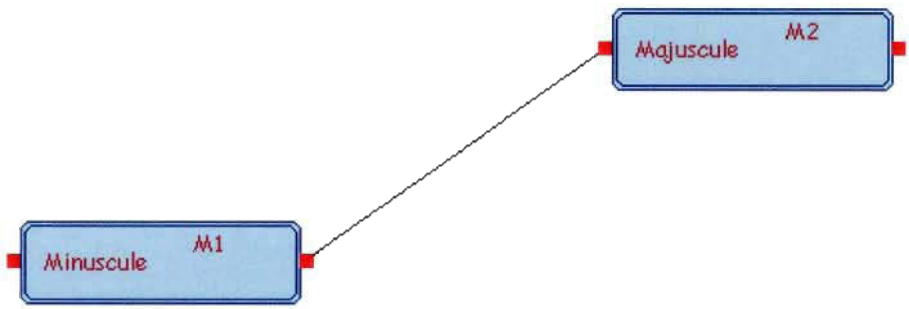


Figure 67 : Deux modules en série dans la plate-forme.

Dans cette figure, un cordon est tracé entre le module M1 et M2. Ce tracé se fait à l'aide de la souris. Comme on peut le constater, nous faisons le tout de façon graphique. Nous n'avons pas à taper une touche au clavier qui est une peur pour la plupart des utilisateurs inexpérimentés.

La figure 68 démontre bien que la chaîne combinatoire est devenue une seule chaîne et non pas deux comme avant de relier les deux modules entre eux.

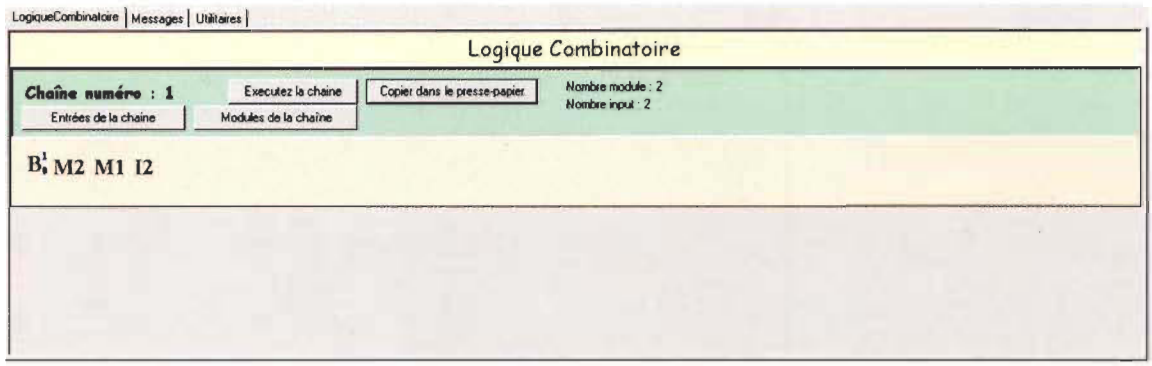


Figure 68 : Chaîne combinatoire de deux modules en série

4.3.1.4 La logique combinatoire

La logique combinatoire, après les modules, est l'un des points importants de la plate-forme. L'intégrité des chaînes de traitements repose sur la logique combinatoire. Nous avons une méthode pour représenter de façon mathématique une chaîne de traitements. Avant, il fallait faire beaucoup de détours afin de sauvegarder une chaîne. Aujourd'hui, nous avons une solution avec des règles simples et qui suit une logique.

La composition de la chaîne de traitements construite au fur et à mesure que l'utilisateur connecte les boîtes ensembles. Prenons un premier exemple pour valider que la logique combinatoire de l'application n'est pas erronée. Il est important de mentionner que dans l'application, les entrées ne porterons peut être pas le même nom que sur le papier. Pour cette raison, nous nous intéressons en premier à la partie combinateur.

Nous allons prendre la figure 69 et la comparer avec l'Atelier. La chaine combinatoire est la suivante $O3 = C_3\ B'_3\ B'_0$ M3 M1 M2 I1 I2

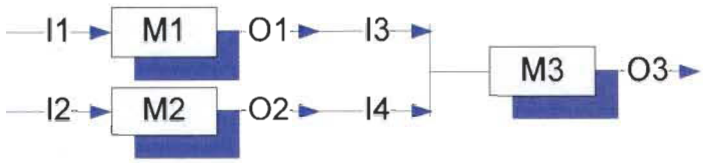


Figure 69 : Cas de modules en parallèle. $O3 = C_3\ B_3^1\ B_0^1$ M3 M1 M2 I1 I2

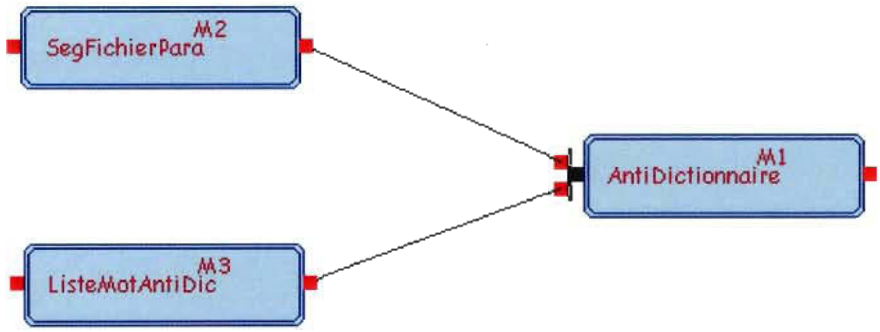


Figure 70 : Représentation modulaire avec l'Atelier de SATIM.

$C_3\ B_3^1\ B_0^1$ M1 M2 M3 I3 I4

Figure 71 : Représentation de la chaîne combinatoire.

La chaîne combinatoire de la figure 70 qui représente la chaîne module de la figure 69. Nous pouvons observer que celle-ci est identique à notre chaîne de la figure 70. La seule différence est l'appellation des noms. L'atelier compte les modules dès l'insertion sur la feuille de l'atelier. L'atelier ne présente pas les noms des entrées et des sorties directement à la sortie. Chaque entrée et sortie sont représentées par des carrés rouges.

Prenons un autre exemple d'un cas d'une chaîne à 7 modules. Cette expression est présentée à la figure 72. La chaîne combinatoire que nous avons trouvée dans notre étude est :

$O7 = C_{40}\ C_{40}\ C_{40}\ C_{40}\ C_{40}\ C_{40}\ B_{40}^0\ C_{32}\ C_{32}\ C_{32}\ C_{32}\ C_{32}\ C_{32}\ B_{32}^1\ C_{24}\ C_{24}\ C_{24}$
$C_{24}\ C_{24}\ B_{24}^1\ C_{17}\ C_{17}\ C_{17}\ C_{17}\ C_{17}\ B_{17}^2\ C_{10}\ C_{10}\ C_{10}\ C_{10}\ C_{10}\ B_{10}^0\ C_3\ C_3\ C_3\ B_3^3$
M7 M5 M6 M3 M4 M1 I8 I9 I10 I3 I4 I1

Elle est très lourde. Ce cas est un parmi tant d'autres. Imaginez une chaîne contenant plus de 100 modules. Pour un être humain, cela serait un vrai cauchemar. La plate-forme devra nous donner la chaîne combinatoire. Voici la chaîne présentée par l'atelier présenté à la figure 72.

$$C_{40}C_{40}C_{40}C_{40}C_{40}C_{40}B^{0}_{40}C_{32}C_{32}C_{32}C_{32}C_{32}C_{32}B^{1}_{32}C_{24}C_{24}C_{24}$$
$$C_{24}C_{24}B^{1}_{24}C_{17}C_{17}C_{17}C_{17}C_{17}B^{2}_{17}C_{10}C_{10}C_{10}C_{10}C_{10}B^{0}_{10}C_{3}\ C_{3}\ C_{3}\ B^{3}_{3}$$
$$\mathbf{M7\ M5\ M2\ M6\ M3\ M4\ M1\ I8\ I9\ I10\ I3\ I4\ I1}$$

Figure 72 : Représentation combinatoire d'une chaîne.

Nous pouvons remarquer que les deux chaînes sont complètements identiques. Les modules ont été ajoutés dans le même ordre qu'à la main. L'ordre des entrées est le même ainsi que l'ordre des modules. Les combinateurs sont identiques. En ce qui concerne l'aspect visuel de la chaîne, la figure 73 et la figure 74 sont presque identiques. Je dis presque parce que les entrées et les sorties ne portent aucun nom sur la figure 74. Cela facilite la lecture de la chaîne. Il y a une possibilité de voir le nom des entrées et de la sortie de la chaîne. Pour cela, l'utilisateur doit faire quelques manipulations.

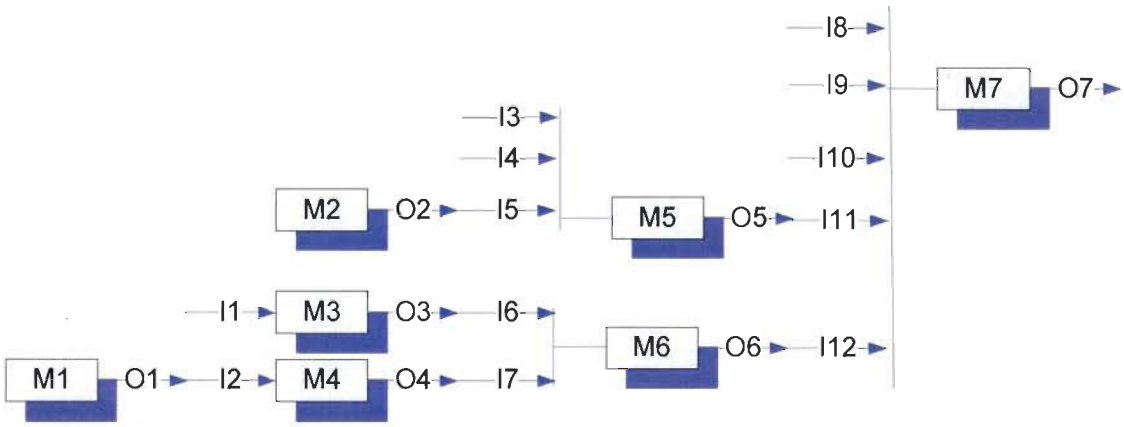


Figure 73 : Cas de 7 modules.

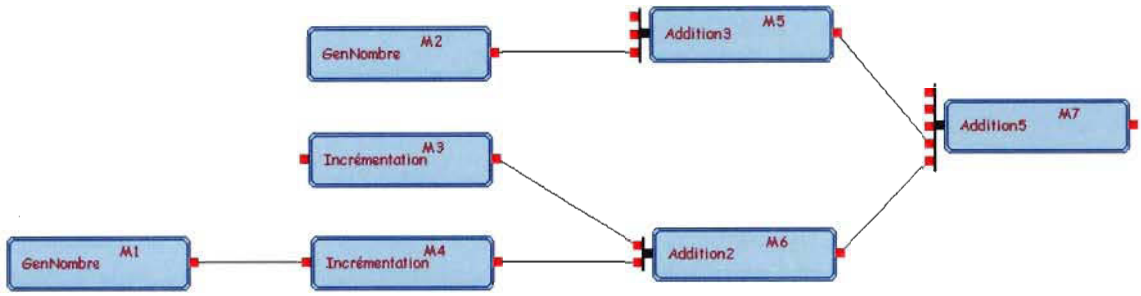


Figure 74 : Schéma de 7 modules combinés par l'Atelier.

L'Atelier permet à l'utilisateur de concevoir une chaîne de traitements avec son inspiration sans devoir se soucier des détails techniques. Si l'expert du domaine devait créer sa chaîne de traitement ainsi que de devoir écrire l'expression combinatoire, cela n'aurait aucun sens. Nous fournissons à l'utilisateur un bon outil de conception.

4.3.1.5 L'enregistrement de méta-module

Quand nous prenons une chaîne comme à la figure 74, cela nous donne un entrelacement de modules. Nous avons parlé au précédant chapitre des métas-modules. Ces métas-modules sont en fait des regroupements de modules contrôlés par un contrôleur. Rappelez-vous les règles pour qu'une chaîne de traitements existe?

1. La chaîne doit contenir au minimum 1 un module
2. Les modules doivent être syntaxiquement corrects
3. La chaîne doit posséder 0 ou plusieurs entrées
4. La chaîne doit posséder 0 ou 1 sortie

Est-ce que la chaîne de la figure 74 répond bien à ces critères? Elle contient 7 modules. Elle répond bien à la première règle. Est-ce que les éléments sont syntaxiquement corrects? Oui, les éléments sont syntaxiquement corrects puisqu'ils

sont connectés ensembles. La connexion est rendue possible par la vérification des types. L'Atelier procède automatiquement à la vérification des types. Il vérifie la sortie et l'entrée qui vont être connectés ensembles. La figure 75 présente un exemple de refus et la figure 76 présente un exemple d'acceptation.

Figure 75 : Affichage de refus de connexion entre deux modules.

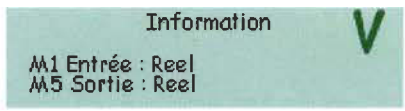


Figure 76 : Affichage d'acceptation de connexion entre deux modules.

Quand notre chaîne est syntaxiquement correcte, nous pouvons dire qu'elle respecte la seconde règle. La sémantique de la chaîne ne peut pas être vérifiée par l'Atelier. La sémantique relève de l'expert du domaine. L'atelier ne pourra pas dire si un traitement a échoué parce qu'il ne respecte pas les règles dictées par l'expert. L'expert devra faire des tests et vérifier les résultats afin de pouvoir s'assurer que la chaîne est conforme à la sémantique. La chaîne comprend plusieurs entrées. Elle est conforme à la troisième règle. La chaîne comprend une sortie. Elle respecte la quatrième règle. Nous avons donc un méta-module.

4.3.1.6 Exécution d'une chaîne

L'exécution de la chaîne est un processus en trois étapes :

1. Saisie des données pour les entrées libres
2. Exécution dans l'ordre des modules
3. Visualisation des données de la sortie

Deux des trois étapes font intervenir l'utilisateur dans le processus. Premièrement, c'est l'utilisateur qui doit remplir les entrées. Dans la figure 77, nous avons six entrées libres.

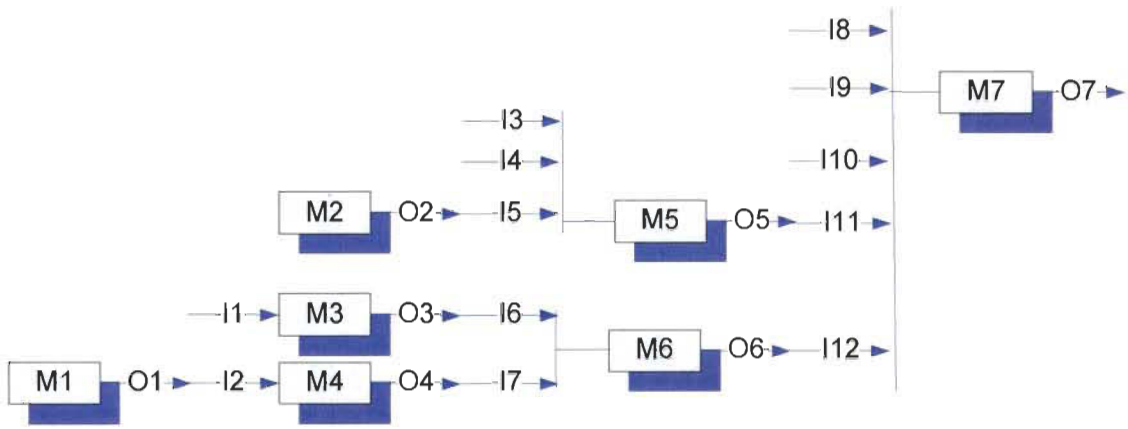


Figure 77 : Entrées d'une chaîne à 7 modules

Si les entrées ne sont pas comblées, c'est que les données sont manquantes et que personne d'autre à part l'utilisateur les a saisies. L'utilisateur devra, pour chaque entrée libre, saisir les données lui-même à l'aide de l'écran à la figure 78. L'écran demande seulement à l'utilisateur de saisir le bon type de données. Une vérification est faite avant la fermeture. Cela évite des erreurs sur les types. On accompagne l'utilisateur tout au long du processus.

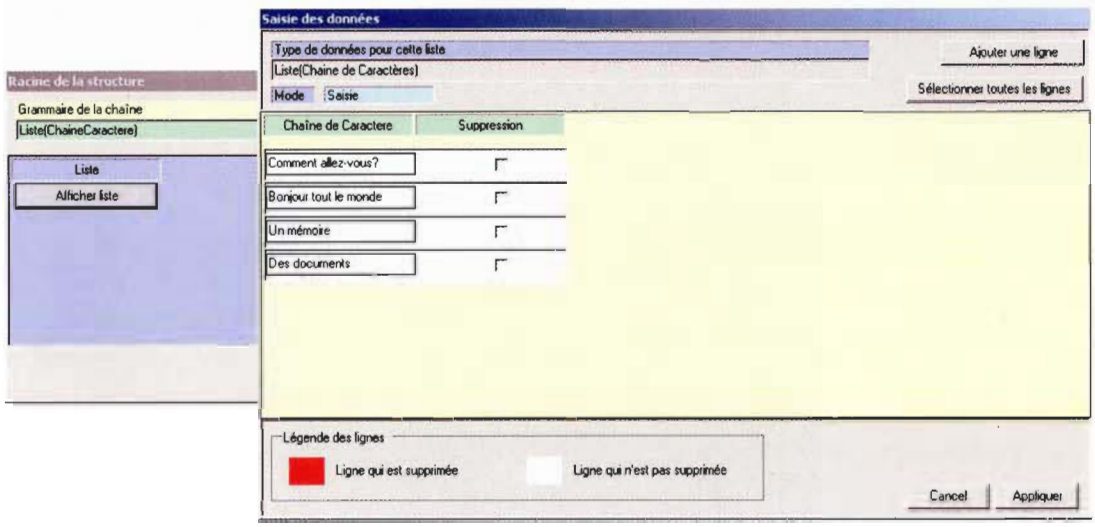


Figure 78 : Écran de la saisie des données.

Les données sont tous sauvegardées sous la forme d'un fichier XML. Ce langage descriptif est tout à fait approprié. La structure en arbre peut aider une personne à se repérer très facilement. La figure 79 représente une saisie d'un nombre entier.

```
<?xml version="1.0" encoding="utf-8" standalone="yes" ?>
- <Donnees type="e_d" typeGrammaire="Liste(ChaineCaractere)">
  - <de td="L">
    - <cel>
        <dl td="CC">Comment allez-vous?</dl>
      </cel>
    - <cel>
        <dl td="CC">Bonjour tout le monde</dl>
      </cel>
    - <cel>
        <dl td="CC">Un mémoire</dl>
      </cel>
    - <cel>
        <dl td="CC">Des documents</dl>
      </cel>
  </de>
</Donnees>
```

Figure 79 : Représentation des données.

Quand toutes les entrées libres sont saisies, nous pouvons passer à l'exécution. L'exécution de la chaîne va utiliser la partie des modules de la logique combinatoire. Rappelez-vous, au dernier chapitre la constitution d'une chaîne combinatoire comme la figure 80.

Figure 80 : Constitution d'une chaîne combinatoire.

La chaîne comprend la partie des modules. Nous avons parlé que les modules sont placés dans l'ordre lorsqu'ils sont combinés. Nous pouvons facilement retrouver l'ordre d'exécution. Prenons la figure 81. Nous avons 7 modules à exécuter. La partie modulaire de la chaine combinatoire est : M7 M5 M2 M6 M3 M4 M1. Est-ce que cet ordre est vraiment ordonné? Si nous la lisons de la droite vers la gauche, nous avons notre premier module qui est M1. Est-ce qu'un module doit être exécuté avant M1 ? Non, il est le premier et n'a pas de précédant. Ensuite vient M4, est-ce que ce dernier peut être exécuté ? Oui, il peut être exécuté puique le module M1 a déjà été exécuté. Il peut lui fournir des données. Prenons M3, peut-on l'exécuter ? Nous pouvons l'exécuter si son entrée libre a été comblée. Passons à M6. Peut-on exécuter M6 ? Oui, M6 peut être exécuté puisque que M3 a déjà été exécuté et que M4 est déjà exécuté. Nous passons maintenant à l'exécution de M2 si son entrée a été comblée. Nous exécutons M5 puisque M2 a déjà été exécuté. Nous devons combler les 2 entrées libres de M5 avant de l'exécuter. Il nous reste plus que M7. Ce dernier module, avant d'être exécuté, doit s'assurer d'avoir comblé toutes ses entrées libres. La figure 81 nous présente l'ordre visuellement.

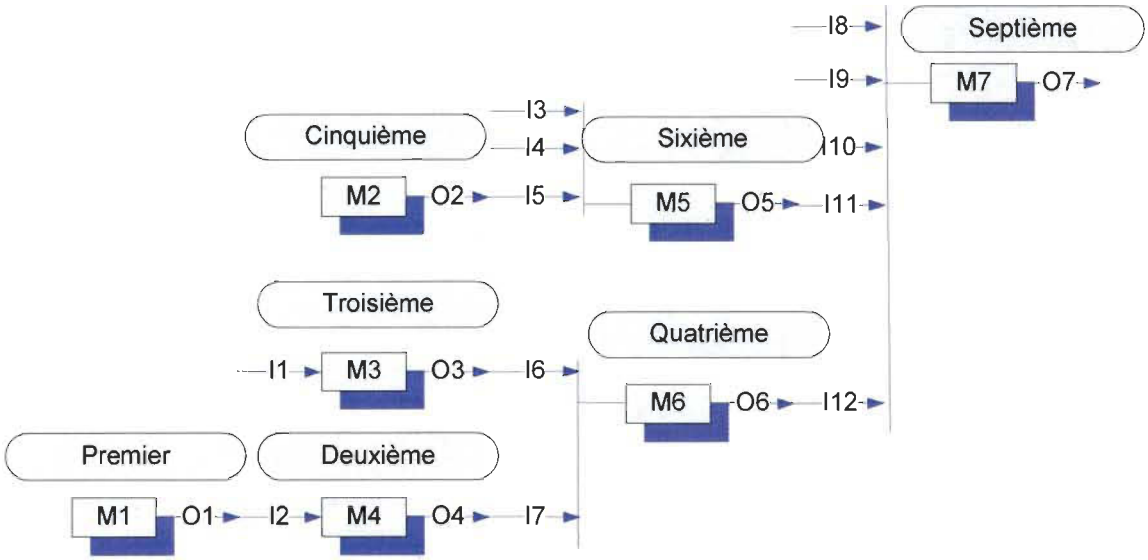


Figure 81 : Ordre d'exécution de la chaîne à 7 modules.

L'Atelier va procéder pour nous à l'exécution de chacun des modules. La figure 82 nous présente l'écran d'exécution des modules. Cet écran nous présente chacun des modules qui sera exécuté et affiche une barre de progression. Il est à remarquer que s'il y a une erreur dans un module, l'exécution de la chaîne s'arrête immédiatement.

Figure 82 : Écran de progression de l'exécution des modules.

Si lors de l'exécution un module est en fait une composition de modules lui-même, comment l'exécution se fera-t-il? Prenons un exemple fictif d'une chaîne qui est : M1 M2 M3 M4 M5 M6. Supposons que M3 est une composition de modules. Que se passera-t-il? Si nous suivons l'ordre d'exécution, nous avons M6, M5 et M4 qui seront exécutés. Quand nous arrivons à M3, le module démarrera et exécutera à son tour sa série de modules. Lorsque le dernier module aura été exécuté, nous allons

exécuter le module M2 ainsi que le M1. Tout le processus est transparent à l'utilisateur. Dans la fenêtre d'exécution, outre le temps un peu plus long pour l'exécution de M3, l'utilisateur n'aura probablement jamais connaissance que ce dernier est une composition de modules.

Lorsque l'exécution de la chaîne combinatoire est complétée, nous avons bien entendu des données de la sortie. Il faut que le module possède une sortie. Nous avons un rapport d'exécution nous montrant l'heure et la date de début ainsi que de la fin de chacun des modules. La figure 83 nous présente l'écran des rapports. Ce rapport peut être utilisé pour faire des comparaisons entre différentes chaînes.

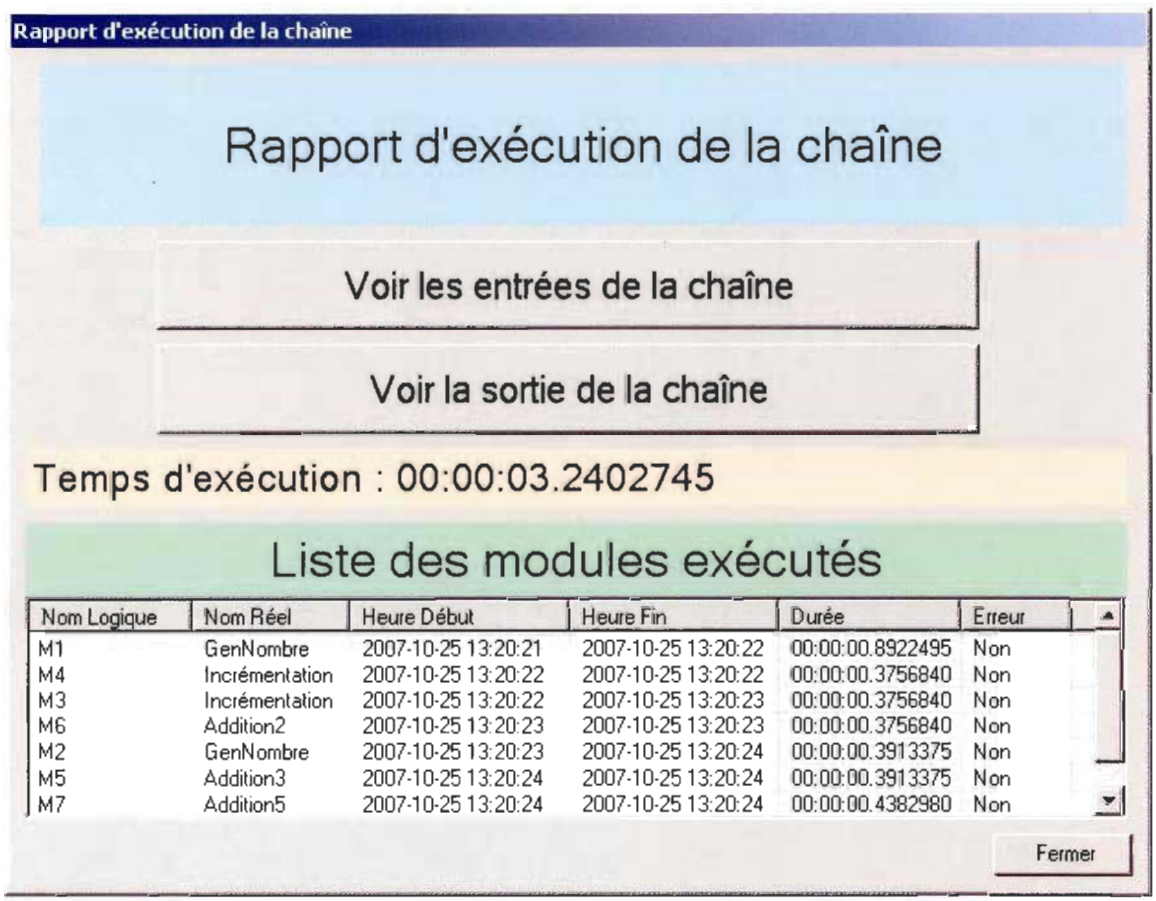


Figure 83 : Écran du rapport d'exécution d'une chaîne de traitement.

Quand l'exécution de la chaîne est terminée, il peut être intéressant de voir si les données de la sortie répondent bien à nos attentes. La figure 84 nous présente cet écran. C'est le même écran que pour la saisie à quelques détails près. Dans cet écran, nous ne permettons pas à l'utilisateur de pouvoir saisir des données. Il ne peut que les visualiser.

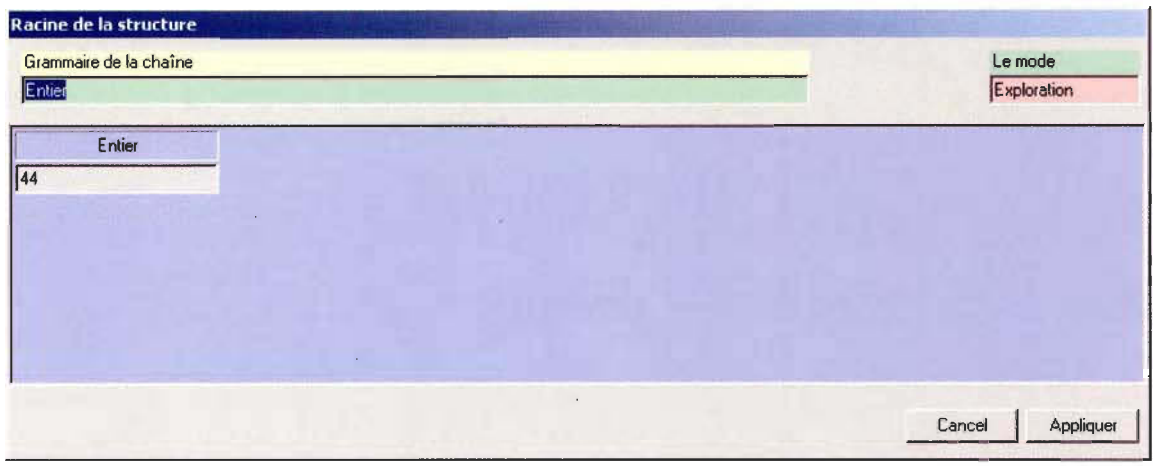


Figure 84 : Écran de visualisation des données.

4.3.2 Le concepteur de modules.

L'Atelier est une grosse boîte à outils nous permettant de gérer les modules et de créer des chaînes de traitements. Cette boîte doit être alimentée. En termes d'alimentation, je parle des modules. Si l'Atelier ne contient aucun module, cette plate-forme n'aura aucune raison d'exister. Ces modules ne tombent pas du ciel. Ils doivent être construits en utilisant des normes. Ces normes doivent être appliquées à la lettre. Un concepteur de modules a été créé afin de simplifier la création des modules. C'est probablement la seule partie de SATIM où un programmeur interviendra dans le processus des chaînes de traitement.

Le constructeur de modules simplifie beaucoup la vie du programmeur. Il construit toute l'enveloppe du module en laissant le soin au programmeur de s'occuper uniquement de la fonction du module. Tout le code est générer automatiquement en

ce qui concerne l'enveloppe. Chaque écran du pas à pas guide le programmeur. Il lui demande d'entrer les informations essentielles du module. Deux des informations les plus essentielles sont les entrées et la sortie. C'est là qu'est la grande partie de la génération du code. Ce qu'il faut comprendre, c'est que la communication passe par en une première étape qui est l'*xmlisation* des données. Pour procéder ainsi, nous devons suivre des normes bien établies pour que tous les modules puissent communiquer ensemble sans problème. Si le programmeur désire le faire de façon manuelle, il n'y a aucun problème. Il devra s'assurer que la grammaire est très bien portée en XML. Ce qui n'est pas une tâche facile si nous avons une grammaire complexe de ce style = { Liste(Entier, Reel, Liste(Entier)), Entier } . Cette grammaire n'est pas simple à mettre en place. L'assistant pas à pas va générer automatiquement le code XML pour la construire.

La figure 85 présente l'écran de développement de l'application. Elle ressemble beaucoup aux autres plates-formes pour le développement. Au démarrage, elle est vide puisqu'aucun projet n'est démarré.

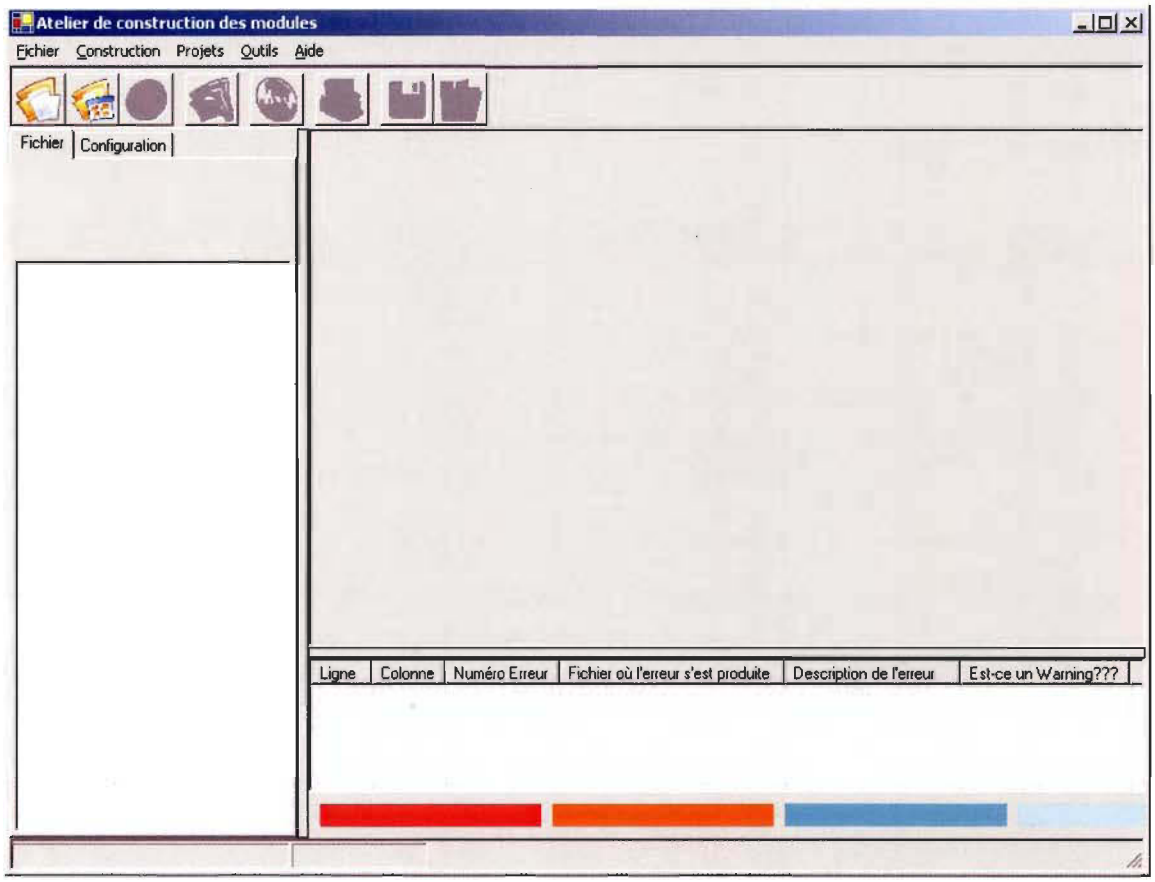


Figure 85 : Atelier de construction des modules.

Lorsque l'utilisateur démarre le pas à pas, il arrive dans un écran de bienvenue très basique. Il peut se déplacer dans le pas à pas sans problème pourvu qu'il remplisse au fur et à mesure les informations qui lui sont demandées. S'il ne le fait pas, des messages personnalisés lui disent de remplir les champs erronés. La figure 86 présente l'écran de démarrage du pas à pas.

Figure 86 : Écran de départ du mode pas à pas.

Lorsque l'utilisateur avance, il entre de plus en plus dans les détails du module. Il arrive à l'écran de la figure 87. Cet écran est celui où les informations vont le plus servir pour la génération du code. En fait, ce sont les entrées et la sortie comme nous le mentionnions. Ils sont d'une importance capitale pour la génération du code.

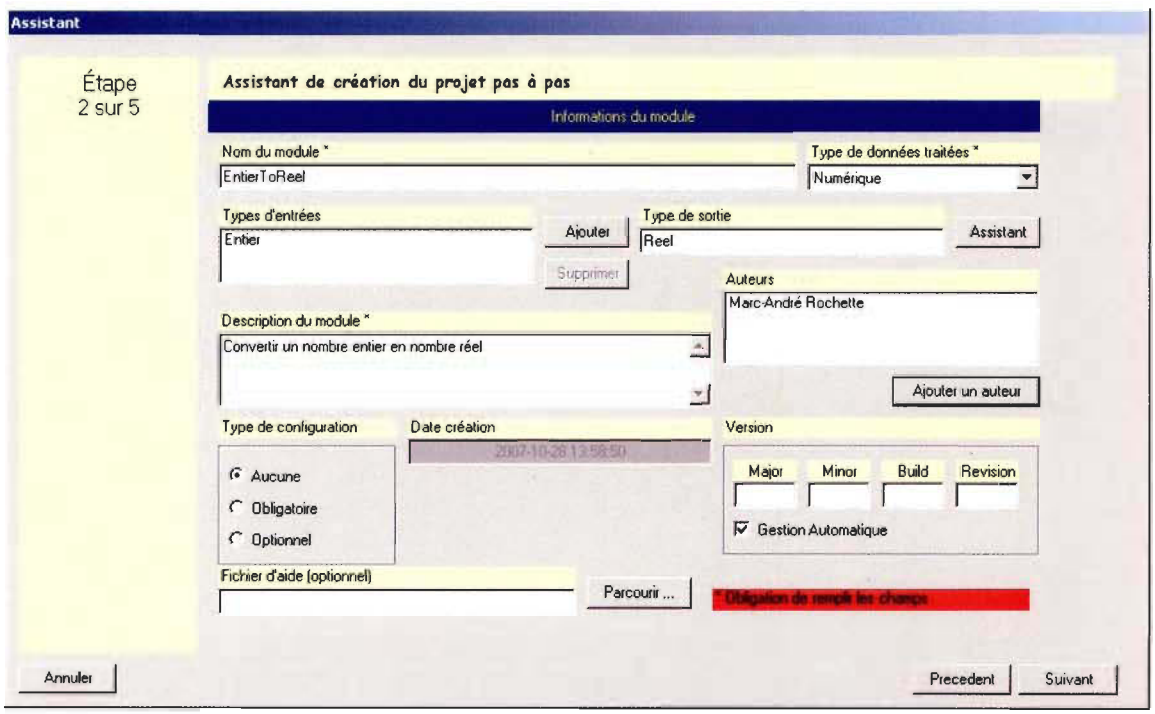


Figure 87 : Écran principal du mode pas à pas avec saisie.

La figure 88 présente l'écran de conception de la grammaire. Cette grammaire doit suivre des règles. Pour aider, l'écran assiste l'utilisateur pour la construire et l'informe si la grammaire est correcte où pas.

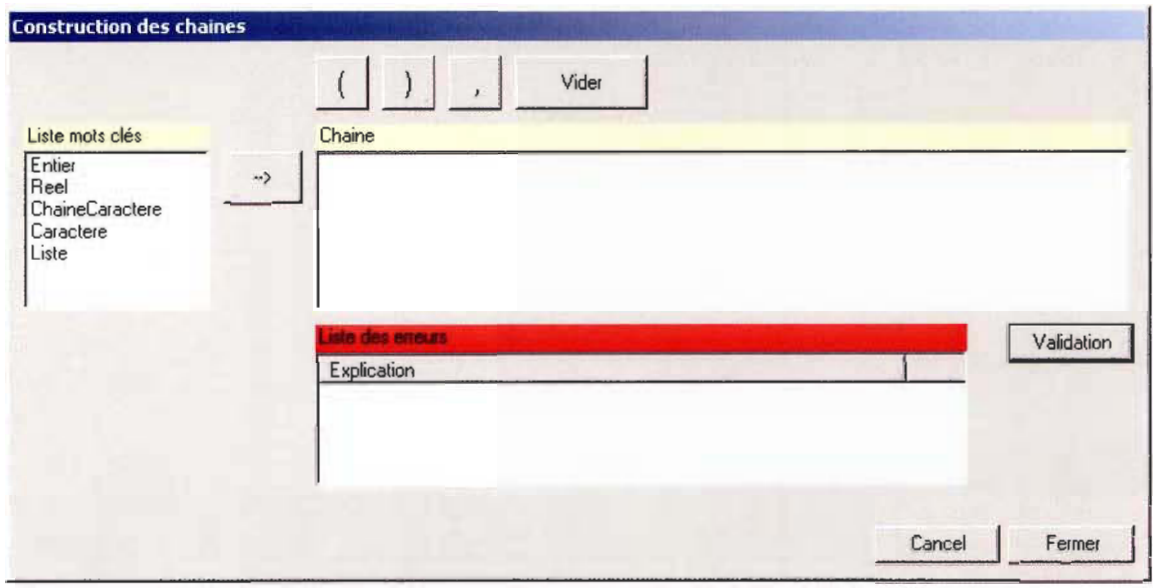


Figure 88 : Écran de conception de la grammaire.

L'utilisateur doit faire le choix du langage de programmation dans lequel il désire que son code soit généré. Comme le démontre l'écran de la figure 89, le seul langage actuellement disponible est le C#. Il faut comprendre que la conception du code de génération est une procédure complexe. Pour l'instant, seul le langage C# est disponible. À long terme, il sera possible d'intégrer d'autres types de modules produits dans d'autres types de langages tels que le Java. Comment cela va-t-il être possible? Par la magie de la communication réseau. La communication réseau rend les applications presque universellement compatibles ensembles. En exploitant cette fonction avec la plate-forme, nous aurons la possibilité de concevoir des modules dans d'autres langages.

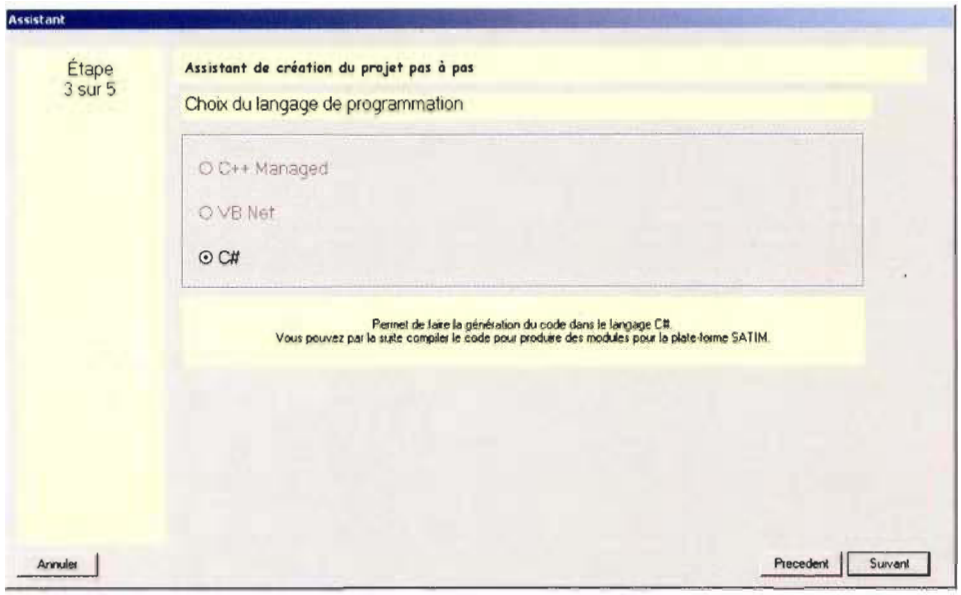


Figure 89 : Écran du type de langage.

Ce procédé viendra alourdir un peu l'architecture logicielle. Comme nous le démontre la figure 90, nous devons rajouter un élément entre la plate-forme et le module Java. Ce module permet de transformer les éléments du module Java en des éléments compréhensibles pour la plate-forme. La couche que nous rajoutons entre les deux se nomme : Module Greffon. Nous pourrions, implémenter un protocole. Cela nous éviterait de devoir passer par un intermédiaire. Cela nous permettrait de pouvoir exécuter des modules à distance plus facilement.

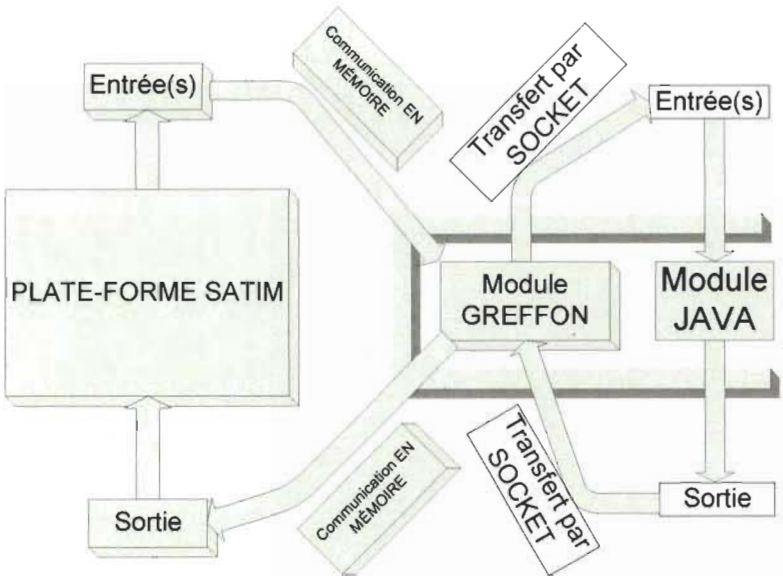


Figure 90 : Schéma de conception pour un autre langage de programmation.

Après les langages et la programmation, vient le temps où le programmeur peut entrer le code de son action. Bien que la figure 91 présente la partie où l'utilisateur peut entrer son code, il lui est possible de le saisir plus tard dans son fichier à l'aide de l'outil. Cette étape n'est pas obligatoire. Si le code n'est pas saisi, le module se compilera et pourra s'exécuter. Il ne produira cependant aucune action sur les données. Il n'y aura pas de retour de données.

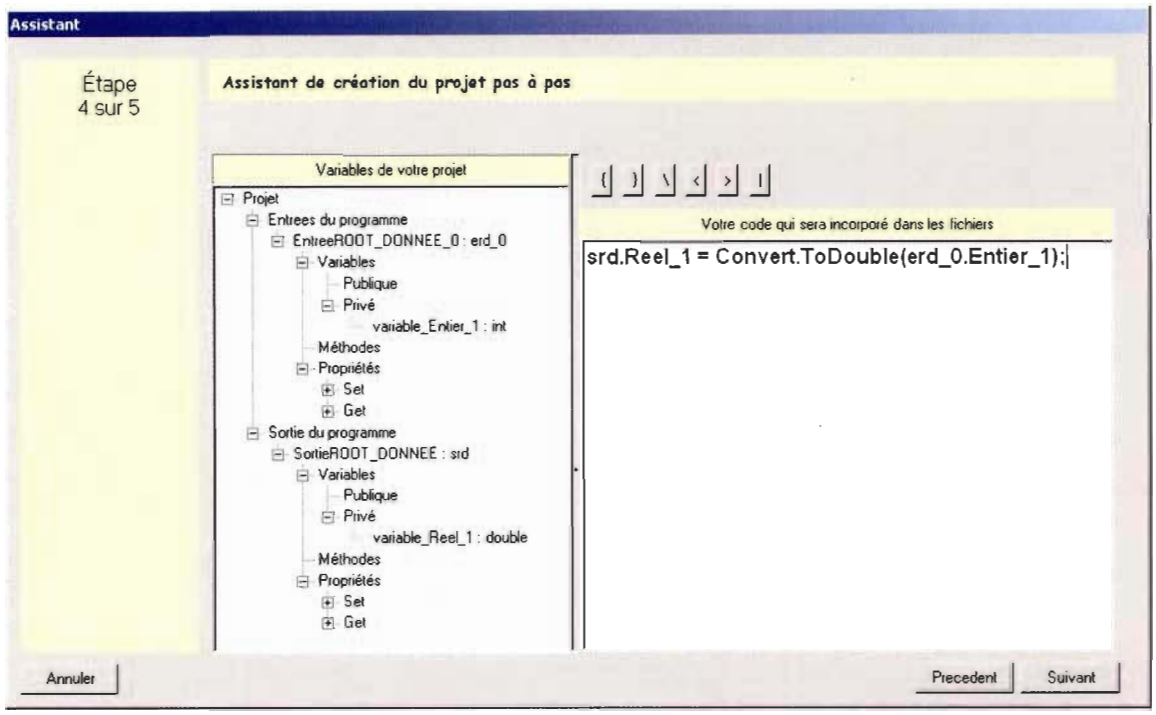


Figure 91 : Écran de saisie du code pas à pas.

Lorsque le programmeur aura terminé de saisir son code, le programme lui demandera s'il désire compiler son code automatiquement. Il pourra modifier son code à l'aide du concepteur de module. La figure 92 présente l'écran principal sur laquel nous pouvons se déplacer au travers des fichiers du projet.

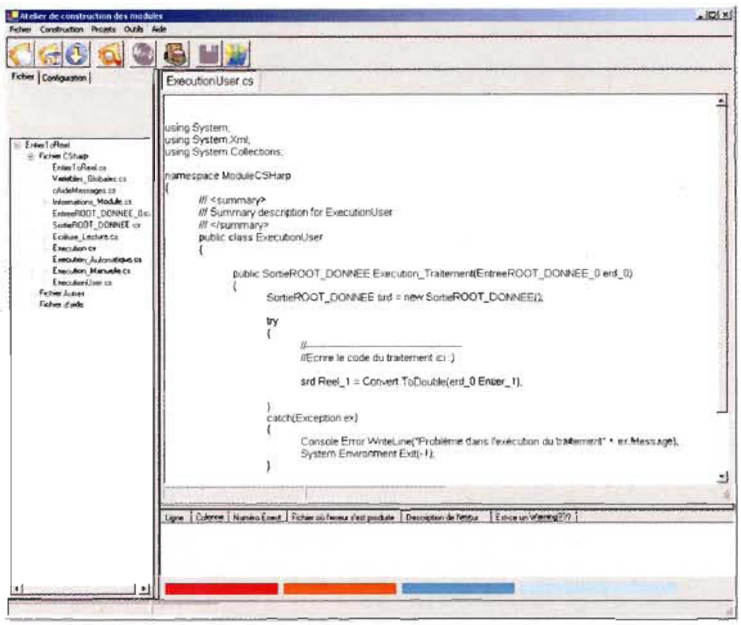


Figure 92 : Écran principal de conception avec code.

Il est important de mentionner que bien qu'il y a possibilité de modifier tous les fichiers du projet, il est recommandé de modifier le fichier de l'action uniquement. Si l'utilisateur désire modifier le reste des fichiers, il le fait à ses risques et périls. Il risque d'altérer le module. Cela le rendrait incompatible avec le reste des autres. Les modifications qui touchent aux fonctions vitales du module sont à faire avec une grande précaution.

Le programmeur a la chance de pouvoir tester son module dans un environnement de tests qui est produit par le concepteur de module. La figure 93 présente cet écran de test. Le programmeur n'a qu'à spécifier les données des entrées et cela va lui donner la sortie.

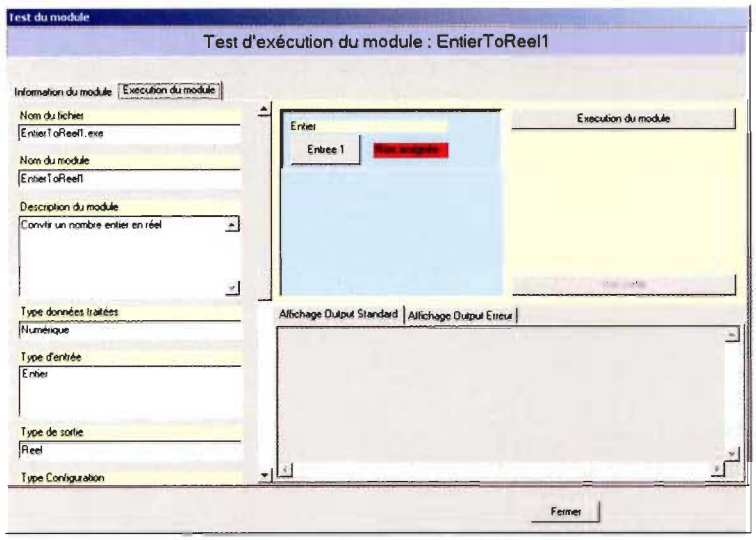


Figure 93 : Écran de test du module en conception.

La plate-forme est conçue afin de faciliter le plus possible la vie au programmeur. Elle est encore très basique. Il y a vraiment beaucoup de possibilité de le faire évoluer tout comme l'atelier.

L'Atelier et le Concepteur de Modules forment la plate-forme SATIM. Nous venons de voir SATIM. Cette plate-forme qui est composé de deux outils essentiels qui sont l'Atelier et le compositeur de Modules. Nous avons bien vu que l'Atelier compose la logique combinatoire pour nous. C'est elle qui fait tout le travail technique. Cette partie est surtout réservée au non initié. L'autre partie qui est le constructeur de module, cette partie est plus réservée au programmeur qui composera les modules pour ajouter à l'Atelier. L'Atelier au grand complet est visuel. L'utilisateur à que très rarement à entrer des paramètres. Cela va lui laisser tout le loisir de développer sa créativité pour concevoir sa chaîne. Nous avons aussi vu que l'exécution d'une chaîne est très simple. En fait l'utilisateur démarre le processus et entre des données si cela est nécessaire. Ensuite, il laisse l'exécution à l'Atelier qui affiche un rapport à la fin de l'exécution. Il reste encore du travail pour améliorer l'interface mais les bases sont jetées.

CONCLUSION

Si vous vous souvenez de l'introduction, je parlais de l'idée de JOHN MARKOFF de faire de la programmation comme on joue avec les briques Légos [Annexe D « Sorfware Out of There »]. Cette idée n'est pas si farfelue que cela et même que cela est faisable. Nous l'avons démontrée.

Les modules de SATIM représentent les briques Légos que nous pouvons emboiter. Nous pouvons les emboiter en respectant les règles. Ces règles sont comme celle des briques Légos. Ce ne sont pas tous les briques qui sont reliées ensemble au même titre que nos modules. Nous pouvons laisser à l'utilisateur la liberté de se créer des chaînes de traitements. Une chaîne de traitements est vue comme un programme en elle-même. Nous laissons vraiment à l'utilisateur le choix de construire la chaîne. Nous devons déployer beaucoup d'effort afin que l'utilisateur n'ait aucun souci en ce qui concerne les détails techniques de la programmation.

Nous avons vu que différents paradigmes nous viennent en aide. L'Orienté Objet garde toujours une place d'importance. On peut s'en servir pour concevoir les modules. Nous avons vu que la Méta-Programmation était particulièrement efficace pour effectuer l'exécution des différents modules de la chaîne. Pour permettre la communication entre les modules, nous avons le langage XML qui descend du paradigme descriptif. Le comportement des modules ainsi que leurs états découlent du paradigme multi-agent. Pour réussir à formuler une chaîne de traitements, nous utilisons la logique combinatoire. Comme nous venons de le voir, chacun des paradigmes a sa place. C'est en les réunissant ensembles que nous arrivons à donner la chance à l'utilisateur de faire fonctionner son imagination.

Comme avec les briques Legos, nous avons besoin d'une base. Quand vous jouez avec des briques, vous avez un point de départ, une plaque, une brique... Pour pouvoir utiliser notre chaîne de traitements, nous avons l'environnement SATIM qui

est notre base. La base est aussi importante que les modules puisqu'elle permet de construire la chaîne de traitements.

Comme JOHN MARKOFF le disait, l'Internet entre dans l'ère des Lego [Annexe D « Sorfware Out of There »]. Un concept pour enfant refait surface pour, peut être, apporter des solutions aux très gros logiciels dispendieux et difficiles à maintenir. En parlant d'argent, nous ne pouvons pas oublier cet aspect. Il faudrait peut-être songer à faire cohabiter le monde du système ouvert ainsi que celui de l'open source comme dans l'exemple cité par Éric Besson dans son article : « L'open source et le propriétaire doivent cohabiter plutôt que de s'affronter ». Il explique que les applications propriétaires et celles du logiciels libres doivent cohabiter et non se faire la guerre [Annexe D « L'open source et le propriétaire... »]. On a qu'à penser à la guerre que ce livre Linux et Windows. Est-ce que la cohabitation ne serait pas plus bénéfique à tout le monde. Comme le dit Hubert Guillaud, la programmation modulaire a été prise en otage par les systèmes propriétaires [Annexe D « L'internet des Légos »]. Aujourd'hui avec l'open-source, il y a un revirement de la situation.

Notre aspect permet d'innover dans la mesure où nous pouvons faire travailler plusieurs types de professionnels. Nous avons les professionnels des métiers ainsi que des informaticiens. Les informaticiens vont créer des modules selon les exigences des experts des domaines. Les experts des domaines vont valider ces modules et créer des chaînes de traitements. Les experts devront valider si leurs chaînes est conforme à ce qu'il demande. Si cette chaîne est conforme, on peut la donner à utilisateur finale qui a presqu'aucune connaissance en informatique ni dans le domaine pour lequel il utilise la chaîne.

Cette façon de procéder permet au non informaticiens de pouvoir créer des logiciels sans devoir avoir des bases en programmation et en création de logiciels. On permet à l'utilisateur d'utiliser sa créativité. Il est sûr que cette créativité est balisée mais on laisse quand même une grande place à l'expert ainsi qu'à l'utilisateur. Ils pourront exercer leurs pouvoirs pour la création.

Ce qui est intéressant, c'est que cette plate-forme peut très bien aller sur internet pour aller chercher des informations ou aller chercher des informations sur votre système. Elle peut aussi vous demander d'entrer des informations. Ensuite, quand les modules sont nécessaires, nous pouvons très bien opérer des traitements sur cette information. Nous pouvons penser aux traitements sur les textes. Chaque personne a sa propre vision de comment chercher des informations. Par exemple, une personne peut très bien choisir de faire la rechercher sur le texte brute sans au préalable effacer certains mots comme les déterminants. Une autre peut vouloir extraire certains mots du textes pour faire des statistiques. Bref, on peut faire beaucoup de traitements sur le texte. L'avantage de notre plate-forme est qu'une personne qui conçoit une chaîne, peut rajouter ou effacer des modules pour effectuer des traitements. Si vous faisiez cela en programmation traditionnelle, vous devriez chaque fois recompiler le tout. Ici, pas besoin, vous n'avez qu'à retirer ou ajouter le module et le connecter à votre chaîne. Notre plate-forme ne se limite pas uniquement aux textes. Nous pouvons très bien l'utiliser pour traiter des informations sur les images, le son, etc. Il n'y a pas de limite. On peut très bien traiter un fichier XML ou un fichier WORD. Il faut bien entendu avoir les modules pour les traiter. Certains modules pourront peut-être servir pour plusieurs types de données. C'est là que la créativité va entrer en jeux. Oui il est nécessaire d'avoir des programmeurs, mais il faut aussi des experts et des utilisateurs finals qui vont s'amuser avec les modules des programmeurs ou alors des métas-modules des experts.

Nous avons un immense jeu de briques. La seule limite pour la création des chaînes est la disponibilité des modules. Si un module n'est pas encore créer et que nous avons besoin pour continuer, là cela va causer un problème. Sinon, aucune limite n'existe dans la création des chaînes de traitements.

ANNEXE A
LANGAGE XML

Le langage XML fait partie des langages à balisage. Ce format, bien que relativement nouveau, est de plus en plus utilisé. Il permet de structurer universellement les données.

Les protocoles de communication utilisent la norme XML afin transférer les données. Le XML est une évolution du langage HTML qui était très peu structuré. La norme internationale du XML peut être trouvée sur le site du W3C (http://www.w3.org/XML/).

Les compagnies utilisent le XML pour régler autant les problèmes de communication que l'enregistrement des données. Le grand avantage du XML est qu'il est interopérable. Cela veut dire que vous n'avez pas à convertir le fichier ou le flux de données XML entre les différents systèmes. Ce qui est un très grand avantage. En plus de structurer les données, nous n'avons pas à nous préoccuper de sa compatibilité avec les différents systèmes qui l'utilisent. Le monde du logiciel libre (Open Source) s'est vite approprié cette technologie afin de créer une suite bureautique avec des fichiers structurés avec le langage XML. On pourrait penser à OpenOffice.

Vous créez un document avec un traitement de texte en Linux et vous pouvez aisément utiliser le même fichier avec un traitement de texte compatible en Windows. Ce qui peut être difficile à mettre en place, se sont les spécifications du format de fichier. Vous devez les respecter à la lettre. Pour plus d'information sur le format ODF (Open Source Format), référez-vous au site de l'organisation (http://www.odfalliance.org/). Ce format est un des plus respecté sur le marché présentement.

Microsoft a commencé à prendre part à ce développement en imposant son propre standard pour le format des fichiers. Le gros problème avec ce type de format, c'est que les compagnies élaborent leurs propres spécifications. Il est très difficile par la suite de tout mettre en œuvre pour qu'il y existe une compatibilité totale. Il faut de solide base pour cela. De plus, les gouvernements ont commencés à s'ingérer dans le débat des fichiers ouverts en voulant une norme. En faisant cela, les gouvernements s'assurent que les compagnies ne les prendront pas en otage avec leurs formats propriétaires. Si un jour il trouve qu'un logiciel de traitement de texte ne fait plus l'affaire, ils peuvent se tourner vers un autre logiciel sans devoir tout convertir tous les documents qu'ils possèdent.

Voici un exemple de fichier XML :

```
- <zoo>
  - <cages numero="1">
    - <especes_animals>
        <animal>Chien</animal>
        <animal>Lion</animal>
      </especes_animals>
    </cages>
  - <cages numero="2">
    - <especes_animals>
        <animal>Cheval</animal>
      </especes_animals>
    </cages>
  - <cages numero="3">
    - <especes_animals>
        <animal>Éléphant</animal>
        <animal>Poisson</animal>
        <animal>Crocodile</animal>
      </especes_animals>
    </cages>
  </zoo>
```

Figure 94 : Exemple d'un fichier XML

Comme nous pouvons le remarquer, tout est balisé dans ce fichier. C'est ce qui facilite la lecture. On sait qu'il y a pour chaque balise d'ouverture, une balise de fin.

Ce sont ces balises qui délimitent le fichier. Dans notre exemple, nous avons un zoo qui comprend 3 cages. Dans chacune des cages, on retrouve des espèces animales. Ces espèces sont identifiées une à une.

Un problème existe dans la création de ces fichiers. Les gens pensent que le XML va normaliser automatiquement les données pour eux. Il en est tout autre chose. XML ne normalise rien automatiquement. C'est au programmeur de s'asseoir et de créer une normalisation solide de son fichier XML au même titre qu'une base de données. Un fichier XML est une base de données. S'il n'est pas bien normalisé, le fichier ne sera pas cohérent. Il faut retenir que les cours de normalisation des bases de données sont très importants et qu'ils le seront toujours.

ANNEXE B
COMMUNICATION ENTRE LES PROCESSUS

Le transfert de la mémoire entre plusieurs processus est un sujet qui a toujours causé le plus de problèmes aux informaticiens. Aucun moyen simple ne semble exister. On ne peut pas faire comme dans un programme et bêtement partager une simple variable dans un espace global. Le processus est un peu plus complexe que cela. Pour résoudre ce problème, 2 grandes familles de méthodes existent. La première méthode est le transfert de mémoire sur disque et la seconde famille est l'échange de données en mémoire vive. Ce sont les grands mécanismes fournis par les systèmes d'exploitation pour la communication interprocessus. Que nous soyons dans le monde de Microsoft ou Linux, se sont les deux principaux modes pour transférer des données. Regardons les deux méthodes de plus près.

B.1 Transfert sur disque entre processus

Le transfert de données entre les processus à partir des fichiers sur les disques est le plus classique. Ce mode n'est pas l'un des mieux adapté aujourd'hui. Une des principales raison est que l'écriture sur le disque est très dispendieuse en temps. Les problèmes d'espaces ne sont plus d'actualité puisque les prix des supports physiques ont grandement chutés.

Quand on écrit sur le disque, que cela soit dans un fichier ou dans une base de données, cela est toujours lent. Nous ne parlons pas juste de petites valeurs numériques. Nous parlons de transfert d'énorme quantité de données. Regardons un peu comment se passe l'écriture sur le disque.

Lorsqu'on qu'on s'apprête à écrire sur le disque, nous devons positionner la tête de lecture dans un endroit où elle peut débuter son écriture. Cet endroit est définit par le système de fichier. Ce dernier est contrôlé par le système d'exploitation. L'écriture commence, mais doit se déplacer un peu de façon aléatoire. Ce que je veux dire par

aléatoire, c'est qu'un fichier est normalement séparé en plusieurs parties sur le disque. L'ensemble des fichiers n'est pas contigus. L'écriture ne se fait pas en ordonnancement sur le disque. Voici un schéma qui représente bien ce genre d'organisation sur le disque :

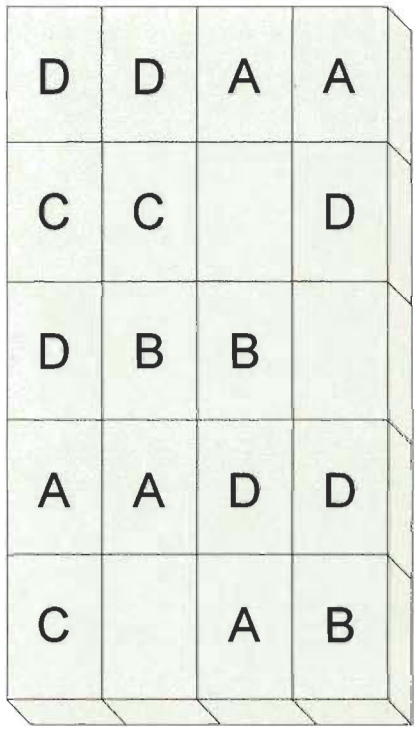


Figure 95 : Schéma d'une table d'allocation de fichiers non contigus (fragmenté).

La figure 95 présente 4 fichiers, soit le Fichier A, B C et D. Ces 4 fichiers ne sont pas contigus. On peut remarquer que le fichier A se retrouve à 3 endroits sur le disque, le B à 2 endroits, le C à 2 endroits et le D à 4 endroits. Présentement, nous allons présenter un exemple contenant seulement 4 petits fichiers. Imaginez ce que cela fait sur les systèmes réels de nos jours où les fichiers se comptent par millier. L'espace alloué à chaque case est définit par le système d'exploitation et le système de fichier. Pour lire le fichier D, la tête de lecture devra se déplacer au minimum 5 fois. Pourquoi 5 fois, nous devons pour commencer se positionner au début du fichier. Vous allez me dire, cela est bien facile, il se trouve au début du disque et la tête de lecture devrait commencer logiquement au début. Et bien non, la tête de lecture s'arrête au dernier endroit où elle a lue ou écrit. Pour lire le fichier D, on se

positionne au début, on lit les deux premiers blocs et on va se positionner sur les prochains blocs. On fait cela pour chacun des blocs. Bien entendu, vous allez dire, nous pouvons défragmenter le système de fichier. La défragmentation est une méthode utilisée pour repositionner chacune des cases du fichier. Ce processus est long et on ne le fait pas à chaque fois qu'on fait une petite écriture sur le disque.

Avec les disques qui doublent en capacité au 6 mois, nous pouvons vite imaginer qu'écrire sur de tels disques prend un temps énorme. Je parle de très grandes quantités de données. Les bases de données peuvent donner un léger avantage parce qu'elles n'écrivent pas toujours immédiatement les données sur le disque. Parfois, elles ont une mémoire tampon. À la fin, on revient quand même à écrire sur le disque. Ce que nous devons faire, c'est de transférer des quantités énormes de données entre deux processus. Donc, ce qu'il faut nous mettre en tête, c'est qu'un processus devra écrire dans un fichier et l'autre devra le lire. Logiquement, la tête de lecture fera le double du travail. Le temps nécessaire pour ce genre de solution n'est pas envisageable.

B.2 Transfert en mémoire

B.2.1 Gestion de la mémoire Win32

Regardons maintenant une autre avenue : la mémoire de Windows. L'espace mémoire d'un processus est entièrement protégée. Il est créé dans le *heap (tas)*. Un autre processus ne peut pas violer cet espace. Lors du lancement du processus, le système d'exploitation fait l'allocation des pages en mémoire. Le processus prend alors le contrôle de ces pages. Il s'exécute et à la fin, il libère la mémoire. La libération de la mémoire ne signifie pas que les données qui sont dans ces zones vont être effacées. Tant que personne ne touche à ces zones dans la mémoire, les données demeurent inchangées.

Deux processus ne peuvent pas partager la même zone de mémoire quand ils s'exécutent. Le partage de la mémoire ne se fait pas sans problème. Pour commencer, dans quelle zone de la mémoire les applications peuvent-elles partager des données? Cette zone est la mémoire virtuelle. On y accède par le *Memory Mapped File* (http://msdn.com). C'est un fichier que nous créons et qui contient des pages en mémoire pour stocker des données. Elles sont consultées à tour de rôle par les processus.

Le gestionnaire de mémoire prend le contrôle du fichier. Il va stocker les informations dans la *Mémoire*. S'il juge que la mémoire est trop saturée, il pourra écrire sur le disque pour simuler de la mémoire. Ce système ralentit un peu la gestion de la mémoire. Le programmeur ne touchera jamais à la gestion de la mémoire. Les fonctions de l'API cachent le tout.

L'allocation des tables en mémoire est réservée au gestionnaire de la mémoire du système d'exploitation. Néanmoins, dans certains langages de programmation, il faut créer un espace mémoire pour une variable. À la fin du processus, il faut absolument la détruire sinon elle demeure dans la mémoire.

Le système de *fichiers mappés* en mémoire permet aux programmes de partager la mémoire. Pendant qu'un écrit, l'autre doit attendre et vice versa. Il y a une synchronisation qui se produit entre les deux. Nous allons voir plus loin un schéma pour son utilisation.

Ce qu'il faut retenir, c'est que la mémoire est protégée par les applications. Il n'est pas facile de la partager entre deux processus.

B.2.1.2 La gestion de la mémoire pour le Framework .Net et Java

Avec le .Net ou le Java, la gestion de la mémoire se fait automatiquement. Nous n'avons pas à se soucier d'allouer l'espace pour faire la création de variables ou de la

destruction. La machine virtuelle qui fait rouler ces environnements se charge de vider la mémoire. Le robot s'appelle le *garbage collector* ou le *ramasse miette*. Le robot fait le tour de la mémoire et s'occupe de détruire les variables qui ne sont plus référencées par les applications. Ce procédé est automatique. Il est possible de lancer le *garbage collector* manuellement par code.

B.2.2 Communication Interprocessus

La communication entre les processus se fait par la mémoire de l'ordinateur ou par l'écriture sur un support physique. Nous nous attarderons à Microsoft Windows, par contre, il est important de dire que la plupart de ces mécanismes existent déjà sur d'autres Systèmes d'exploitation tels que Unix, Linux, … Nous avons 9 façons natives de l'API Win32 de transférer des données entre les processus (http://msdn.com). Nous allons voir 4 méthodes plus évoluées pour le partage (une fait partie du monde Java (http://java.sun.com)) et finalement, nous aborderons un petit peu le .Net Framework (http://msdn.com). Voyons ces 13 méthodes.

Tableau III

Les différentes méthodes de communication en mémoire

Clipboard (Presse-Papier)	COM/DCOM
Data Copy	DDE
File Mapping	Pipes
RPC	Windows Socket
Database	Remoting
RMI	CORBA
Mailslots	

À l'annexe C, nous allons parler de la communication inter processus par le *Framework .Net*. Expliquons chacune des méthodes du tableau III.

B.2.2.1 Clipboard (Le presse papier)

Ce système de gestion de la mémoire est natif. Il est intégré dans le système d'exploitation afin de transférer des quantités de données entre les applications. Les applications peuvent se trouver sur le même ordinateur ou sur deux ordinateurs distants à travers les réseaux.

Les applications qui s'échangent des données doivent est compatible. Par exemple, si je transfère du texte d'un traitement de texte à un logiciel de dessin, cela n'a aucun sens. Il n'y a pas de concordance. Il faut s'assurer que ces types de données soient compatibles entre les applications. Il n'est pas assuré que les données vont être là entre le temps où l'application va déposer et le temps où l'autre application va les chercher. Une autre application pourrait très bien y déposer quelques choses entre les deux opérations. On ne peut pas monopoliser le presse-papier juste pour 2 applications. À chaque nouvel ajout, on efface tout le contenu du presse-papier. Nous perdons toutes les données à chaque fois.

B.2.2.2 COM (Component Object Model)

C'est un modèle puissant qui est imposé dans le monde de la technologie Microsoft. Il est basé sur la technologie OLE (Object Linking and Embedding). Cela permet, par exemple, à un document de pouvoir recevoir des parties d'un autre logiciel. L'exemple typique est d'utiliser Microsoft Word pour créer un document texte et d'y insérer des parties d'un document Microsoft Excel. L'utilisateur peut démarrer directement Microsoft Excel à partir du document Microsoft Word et d'y insérer des données dans les cellules tout en ayant la sensation de toujours se trouver dans Microsoft Word. Il va voir très peu de différences à son document. La figure 96 présente l'exemple de ce document.

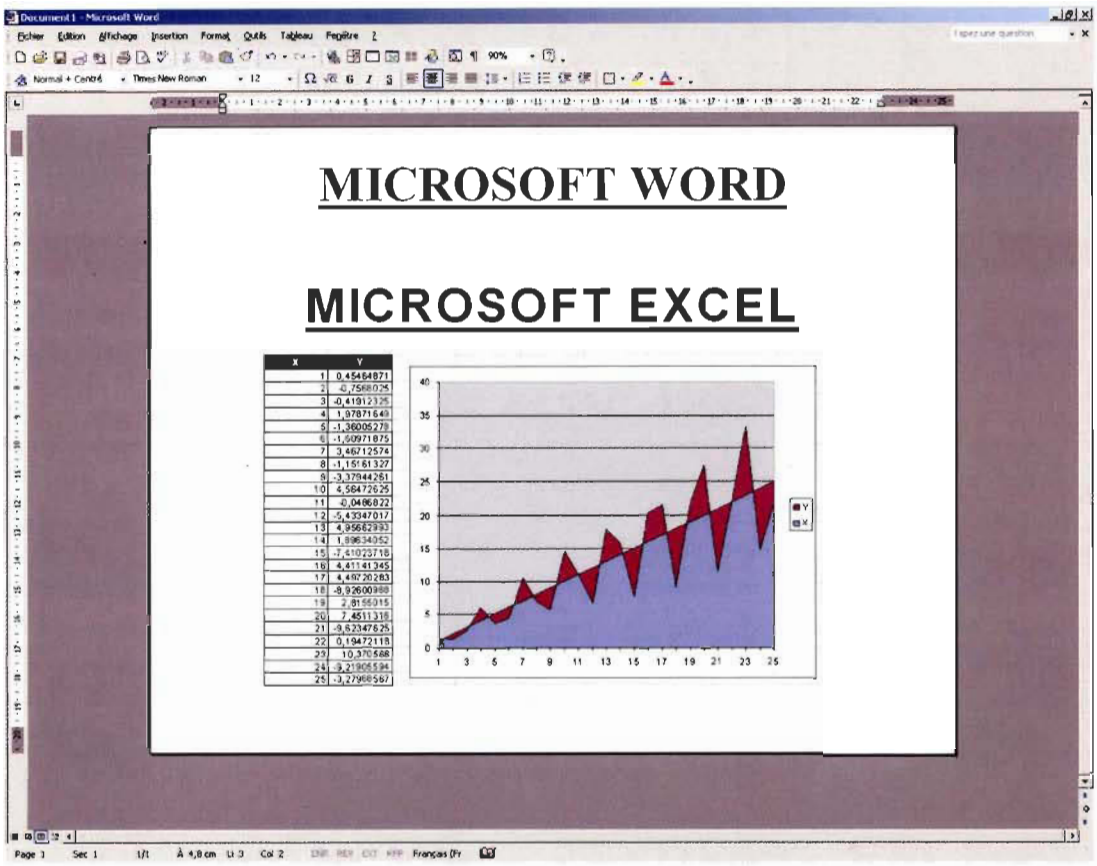


Figure 96 : Document Excel inséré dans un Document Word.

Les applications qui implémentent le modèle *COM* peuvent communiquer entre eux de la même façon que Word et Excel. La communication OLE va bien au-delà de cette simple forme de la communication. Nous pouvons nous servir de ce modèle pour la communication d'une application vers une base de données par exemple.

Les documents ActiveX sont très répandus sur le Web. Toute cette technologie est en mutation pour passer à la plate forme Microsoft .Net.

Intégrer un modèle *COM* n'est pas une mince affaire. Certains sites expliquent en détail ce qu'il faut faire. Le modèle est lourd. Il faut avoir une solide base en programmation avant de développer le modèle *COM*.

B.2.2.3 Data Copy

Ce principe utilise l'envoie de message entre des applications par l'entremise de la boucle des messages de l'application. En Win32 ainsi que dans la plateforme .Net, les applications sont construites avec au moins une boucle de messages. L'envoie de messages à travers une application se fait par la fonction de l'API (Application Programming Interface) *SendMessage* avec le message *WM_COPYDATA*.

Il doit y avoir un certain protocole qui est implémenté entre les deux processus pour recevoir et émettre. Il est important de dire que l'application qui envoie le message ne peut pas modifier les références en mémoires des variables qui lui sont envoyés. Pour pouvoir faire une modification, cette dernière doit faire une copie de la variable et la modifier. L'originale reste protégée en mémoire vu que la mémoire, dans laquelle elle est créée, est propriétaire de l'autre application.

Les deux applications doivent être exécutées en même temps. Il n'y a pas d'attente dans les messages. Ils ne sont pas stocker en mémoire. Ils attendent qu'un processus les appelle. On doit savoir le nom de l'application à qui est destiné le message. Généralement, on recommande ce transfert pour de petites quantités de données.

B.2.2.4 DDE (Dynamic Data Exchange)

Ce principe prend exemple sur le *Data Copy* ainsi que sur le *Clipboard*. Il permet de faire l'échange entre les applications en construisant un protocole de communication en ayant un émetteur et un récepteur. On peut se baser sur le modèle client serveur afin d'avoir plusieurs clients.

Le format de donnés est du même type que le *Clipboard*. Cela laisse aux applications une plus grande latitude face au format de données. L'échange des

données se fait normalement en synchronisant l'émetteur et le récepteur à l'aide du protocole qui est fournit par le DDE.

Il n'est pas nécessaire que les deux applications vivent en même temps pour pouvoir s'échanger des données. Cela dépend de la manière où on stock les données. Si on passe par le *Clipboard* ou un autre système. Pour répondre au protocole, nous utilisons ici la boucle des messages des applications Microsoft Windows. Ce système peut fonctionner entre les applications d'un seul ordinateur ou de plusieurs ordinateurs sur un réseau.

B.2.2.5 File Mapping (Fichier de trace de la mémoire)

Ce moyen de communication peut servir à deux concepts de communication. On peut se servir de ce mode pour communiquer à l'intérieur d'un même programme ou entre plusieurs programmes.

Le principe est de créer un fichier sur le disque qui fait référence à des blocs de mémoire. Il est important de dire que les applications, qui vont créer ces fichiers, n'ont pas de gérance sur le fichier en ce qui concerne l'allocation de la mémoire. C'est le gestionnaire de mémoire de Windows qui va gérer ce fichier. S'il trouve qu'il n'y a plus assez de mémoire ou s'il y a un dépassement, le gestionnaire va écrire des données sur le disque pour compenser.

Le principe de ce système, que cela soit dans un même processus ou entre deux ou plusieurs processus, est le même. Un des deux doit créer le fichier et y donner des droits d'accès. Le fichier va être accessible à tous. Nous pouvons attribuer des droits à ces fichiers. Il est important de synchroniser l'accès au fichier. Si nous ne synchronisons pas les *processus* ou les *threads*, nous allons avoir des erreurs de lectures et d'écritures.

L'inconvénient de ce système et que si nous dépassons l'espace alloué par le gestionnaire de la mémoire, ce dernier écrit sur le disque. Dès que nous écrivons sur le disque, nous perdons beaucoup de vitesse comme nous l'avons vu plus haut. Ce principe ne répond pas vraiment à nos besoins.

B.2.2.6 Mailslots (Boîte postale)

Et oui, on peut échanger du courrier entre les applications sous le même principe du système de courrier électronique ou postal que nous connaissons. Ce système est surtout utilisé afin d'envoyer de courts messages et non de grandes quantités d'informations.

Le principe est qu'un ordinateur poste un message par la *Boîte postal*. La *boîte postale* est un processus. Toutes les applications peuvent devenir serveur d'une *Boîte postale*. L'application crée la boîte sous un nom *X*. Les applications peuvent envoyer des messages sur cette boîte ou les lires. Microsoft recommande d'utiliser les *Pipes* ou les *Sockets* plutôt que d'utiliser la *boîte postale*.

B.2.2.7 Pipes (Tuyaux)

Les tuyaux de la communication. Ce principe est d'ouvrir un canal de communication qui relie deux processus ou des redirections de canaux. Il existe deux types de *PIPES* : les *PIPES Anonymes* et les *PIPES nommés*. Les *PIPES anonymes* servent pour rediriger les entrées standards. Par exemple, lorsque nous voulons rediriger l'entrée de l'écran. Au lieu que la fonction *printf* écrive des données à l'écran, nous allons rediriger la sortie *stdout* vers un *PIPE anonyme*. Avec ce moyen, nous allons pouvoir traiter les données qui étaient destinées à l'écran.

Les *PIPES nommés* sont utilisés pour la communication interprocessus. Un *PIPES* est unidirectionnelle. Cela veut dire que nous pouvons seulement écrire ou lire sur le *PIPE* à chacun des bouts. Un côté doit lire et l'autre écrire. Ils ne peuvent pas

échanger les rôles quand la communication est commencée. Par exemple, si l'application *A* désire envoyer un message à l'application *B*, elle ouvrira un tuyau de communication. *A* va écrire les données à *B* et ce dernier va les lires. Si jamais *B* désire écrire des messages à A, il devra ouvrir un *PIPE* afin d'envoyer des données à *A*. Les fonctions de lectures et de l'écriture sont bloquantes : pendant qu'un écrit, l'autre ne peut pas lire.

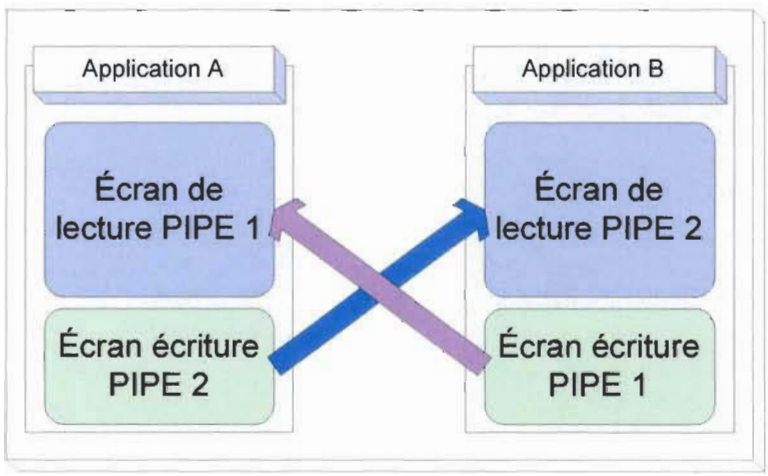


Figure 97 : Transfert de données par les PIPES (tuyaux) de communication.

Nous pouvons aussi utiliser les *PIPES* sur une même machine entre processus ou sur des machines distantes.

B.2.2.8 RPC (Remote Procedure Call)

Le *RPC* est un pour le transfert de données entre les applications. Ce système permet d'appeler des fonctions sur un réseau. La distance est relative, cela peut se trouver entre processus d'une même machine ou sur des machines distantes. Le *RPC* qui est distribué par Windows est totalement compatible avec les normes de *Open Software Fundation* qui est décrit par le *Distributed Computing Environnement*.

Le principe est d'avoir une architecture client / serveur. Le *RPC* cache toute la gestion des protocoles réseaux. Il le gère pour nous. Cela nous donne la possibilité de se concentrer uniquement sur notre problème.

Le *RPC* est au niveau des fonctions. Il y a la possibilité de faire des conversions automatiques. Nous verrons que la version du Microsoft .Net ainsi que le RMI de Java sont bien meilleures.

B.2.2.9 Windows Socket

Ce système est à la base de la communication réseau à travers les applications de Microsoft Windows. Il est tout à fait possible de créer des sockets entre deux processus sur une même machine et aussi de créer des sockets entre deux processus sur un réseau. Un grand avantage des sockets, c'est qu'ils nous permettent de communiquer, en implémentant des protocoles, sur des architectures différentes. Par exemple, faire communiquer un processus Windows avec un processus Linux. Il est aussi possible de faire l'échange de données entre des processus Linux et des processus Macintosh. Il s'agit d'avoir un protocole qui est standard sur les deux bouts de la *Socket*.

La communication entre les deux est bidirectionnelle (Figure 98). Nous pouvons soit envoyer ou recevoir par le port. Ces actions doivent être clairement décrites avec le protocole de communication. L'opération d'écriture et de lecture sur la *socket* est bloquante. Il faut alors utiliser les *threads*. Normalement, nous utilisons les *sockets* à travers une architecture client / serveur (Figure 99).

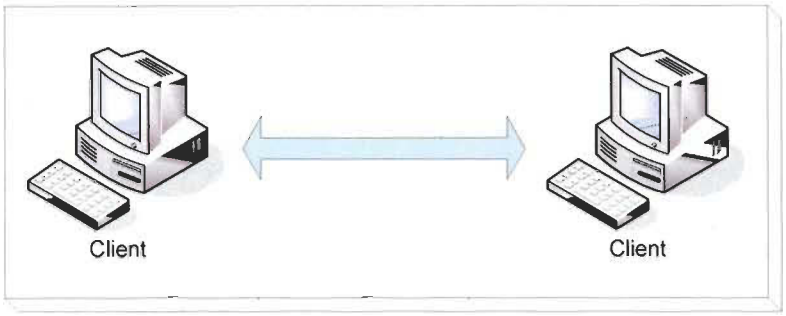


Figure 98 : Modèle bidirectionnelle.

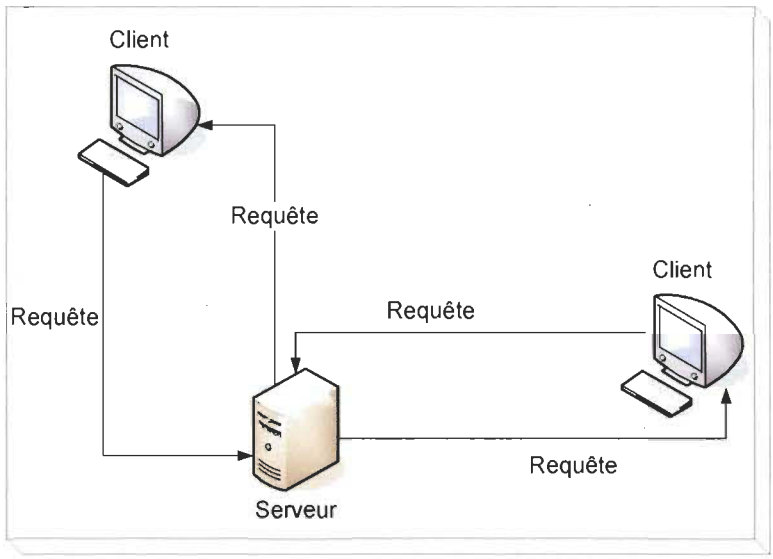


Figure 99 : Modèle Client Serveur.

B.2.2.10 Database (Base de données)

Ce principe a été abordé quand nous avons parlé de l'écriture sur disque. Il reste néanmoins que c'est une forme de communication qui existe entre les processus ou dans le même processus.

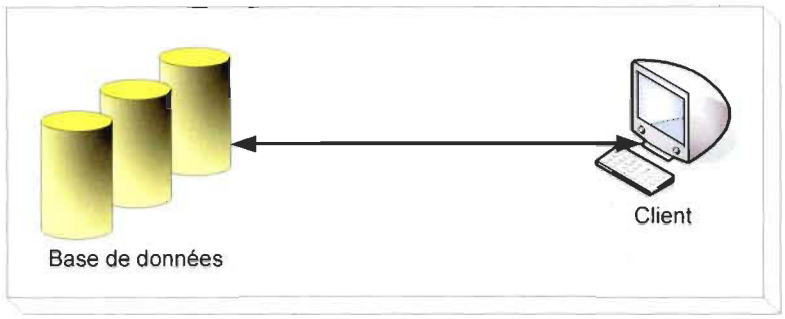


Figure 100 : Schéma d'un client avec une base de données.

Le principe est assez simple. Il s'agit d'ouvrir un canal de communication entre le processus et la base de données. Ensuite, il ne reste plus qu'à effectuer les transactions.

Si nous avons plusieurs clients se connectent à la même table sur la même base de données, il est important de faire une synchronisation entre les deux applications. Si les 2 processus accèdent en même temps aux données, cela risque de produire des pertes de données. Il faut mettre une gestion des transactions pour éviter la perte de données.

L'accès aux bases de données peut se faire sur la même machine et / ou à travers un réseau. Il est possible de faire un partage de données entre plusieurs processus où à l'intérieur d'un processus. Il s'agit, de faire une synchronisation entre *processus et thread*. Ce n'est pas la meilleure façon d'échanger des données entre les processus. Souvenez-vous de ce que nous avons parlé lors de l'écriture et la lecture sur disque. Si des applications doivent partager d'importantes quantités d'informations durant l'exécution, alors la base de données n'est pas le meilleur moyen.

B.2.2.11 Remoting .Net (Communication à distance)

Le principe du *.Net Remoting* repose sur la communication entre différents processus (Client / Serveur). Le plus grand avantage est la facilité d'utilisation de ce

système. Avant, quand nous utilisions la communication de types (Client / Serveur), le programmeur avait la lourde tâche d'implémenter le protocole et de gérer les entrées et sorties. Il devait les gérer du côté du serveur et du client. Cela avait comme tâche d'être assez difficiles. Nous avons qu'à penser au *multi-threading* côté serveur. L'avènement du *.Net Remoting* enlève beaucoup de charge au programmeur. On lui permet de se concentrer plus sur son application et on le dispense de gérer les *sockets* et les protocoles sur le réseau. Le *.Net* se distingue par sa gestion interne pour le serveur. Avant, quand nous avions plusieurs clients qui se connectaient à un serveur, il fallait créer un *thread* par client. Aujourd'hui, toute cette partie se fait à l'interne et le programmeur n'a pas besoin de s'en soucier.

Le *.Net Remoting* peut être utilisé soit sur le même ordinateur ou soit à distance sur un réseau. L'implémentation à la base est le même. La seule chose qui change est l'adresse de la communication.

On nous offre de pouvoir utiliser deux types d'objets dans le *.Net*. Soit les objets *SingleCall (Simple appel)* qui ont une seule durée de vie. Si l'application *A* appel l'objet et le modifie, l'application *B*, quand il va demander l'objet, n'aura pas les modifications, mais seulement un nouvel objet comme à l'appel initial de l'application *A*. Et nous avons les objets *Singleton (Seul)* qui permet aux applications de partager des données entre les applications. L'objet est créé une seule fois et il va vive jusqu'à temps qu'on le détruira.

Pour la communication, deux protocoles sont utilisé, le premier est le modèle *TCP* qui est rapide. Le second passe par le protocole *HTTP* qui utilise le format *SOAP (Simple Object Access Protocol)*. Ce dernier est un peu plus lent que le *TCP*.

B.2.2.12 RMI (Remote Method Invocation)

La communication par le protocole *RMI* est très semblable au *Framework .Net*. Pour pouvoir exécuter *RMI*, on doit absolument démarrer l'annuaire d'enregistrement des

objets que nous voulons partager. Un type de serveur global qui devra fonctionner pour une architecture client / serveur. Le serveur enregistre les données dans le *Registry (Annuaire)* et les clients y accèdent. Cela est totalement indépendant des protocoles HTTP, FTP ou autres.

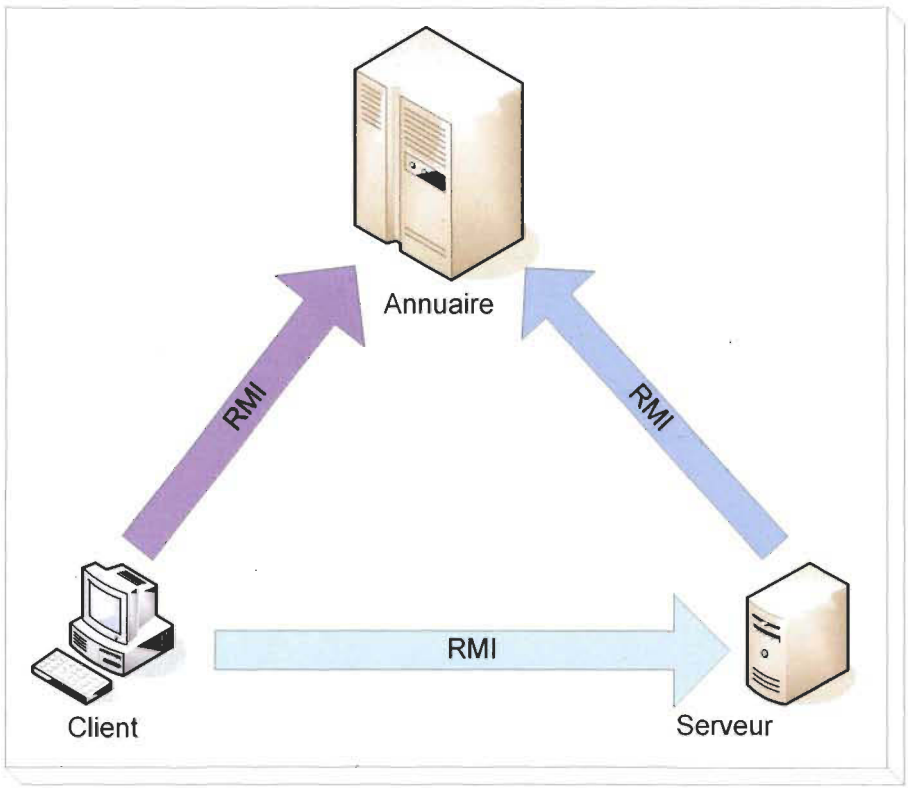


Figure 101 : Schéma de l'architecture RMI.

Selon des études, le *RMI* serait légèrement plus lent que le *Remoting .Net*. Cela pourrait être compréhensible parce que Java fonctionne sur une machine virtuelle. Il y a aussi le fait qu'il y a un serveur qui doit interagir avec le serveur d'annuaire. Le grand avantage est que le RMI peut tourner sous plusieurs plates-formes. Donc, on peut prendre un projet et le faire tourner soit sur Linux, Windows, …

B.2.2.13 Corba (Common Object Broker Architeture)

C'est une architecture qui a surtout été développé dans les années 1990. Corba est une norme pour faire communiquer des applications venant de différents langages entre elles. On peut les faire communiquer entre différents systèmes d'exploitation. Il est orienté pour partager des fonctions. Le principe de Corba repose essentiellement sur l'*ORB (Object Request Broker)*. C'est un mécanisme qui distribue des fonctions / objets. Pour faire communiquer les différents langages, on utilise une *IDL (Interface Definition Language)*. Il faut faire une correspondance entre les types. Avec l'*IDL*, on s'organise pour qu'un type entier soit la même chose pour tous les langages qui sont supportés par Corba. La figure 106 résume bien le tout.

Des personnes ont développées des outils qui sont similaires au modèle *Corba* afin de pouvoir faire communiquer le *Remonting .Net* ainsi que le *RMI.* Ce sont des approches expérimentales présentement.

C'est un système qui est très lourd. Cette architecture est très pratique pour les grandes applications distribuées. Il est aussi pratique si nous avons des applications qui fonctionnent sous différents systèmes d'exploitation et sous différents langages.

ANNEXE C
COMMUNICATION DANS LE FRAMEWORK.NET

Le *Framework .Net* est une innovation dans le monde de Microsoft. Ce système transforme l'univers de Microsoft Windows qui était généralement constitué uniquement d'appelles de fonctions et qui le transforme dans un univers totalement Objet. Sur les machines Windows, pour des raisons de compatibilités, les deux mondes cohabitent. En redéfinissant tout ce monde, Microsoft a dû répondre aux anciennes questions comme par exemple : comment faire communiquer des processus entre eux? Bien que l'architecture logicielle de Microsoft soit évoluée, sur les 8 méthodes proposées par Microsoft, 4 utilisent les fonctions natives de la plate forme *Win32*. Faisons un bref survol de ces méthodes.
(http://msdn.microsoft.com/fr-fr/netframework/default.aspx)

C.1 Named Event

C'est un principe qui est uniquement utilisé par un processus afin d'avertir un autre processus qu'un évènement vient de se produire. Il n'y a pas de transfert de messages ou de données. Il s'agit seulement d'un avertissement. Ce principe utilise les fonctions natives de *l'API Win32*.

C.2 Windows Messages

Très similaire au Named Event. Ce procédé utilise les fonctions natives de l'API Win32 pour envoyer des messages avec un petit bagage de données. Cela ressemble beaucoup à la méthode native *DATA COPY*.

C.3 Point-to-Point message queues

C'est le principe de créer une file d'attente afin que des clients puissent y déposer des messages et / ou des données. Le principe est une architecture client /

serveur. On ne peut pas transférer de larges quantités de données avec ce moyen. Ce procédé utilise des fonctions de *l'API Win32*.

C.4 TCP Sockets

Ce procédé est très semblable au *Sockets* de Windows. Étrangement, Microsoft n'utilise pas les fonctions natives de l'API Win32 pour accéder au réseau. Dû moins, il ne le dise pas sur le site web (msdn.com). Par contre, le principe est le même pour son utilisation *(Architecture Client / Serveur)*. On peut transférer des quantités de données très importantes. Il faut bien sûr implanter un protocole.

C.5 Memory Mapped Files

Ce principe est exactement comme le *File Mapping*. Il utilise des fonctions natives de *l'API Win32*. Du point de vue code, cela nous fait passer de la programmation fonctionnelle à la programmation orienté objet. C'est relativement un peu plus facile pour l'utilisation. Le principe est le même par contre.

C.6 Registry

Un autre moyen proposé par Microsoft est de transférer des données par l'intermédiaire de la base de registres de Microsoft Windows. L'implantation d'une base de registre dans Windows remonte à la version 3.1. Ce n'est qu'avec *Windows 95* que nous avons pris conscience de son importance. Le principe est d'entreposer à un endroit dans le registre, les données importantes du système d'exploitation et des applications. On peut aussi y accéder entre les applications. C'est peut-être une méthode qui est dangereuse, car, la base de registres est le centre nerveux de *Windows*. Si nous effaçons ou modifions des données importantes, nous risquons de faire des dégâts au système d'exploitation. Pour ce qui est de la sécurité, il serait préférable de ne pas utiliser cette méthode et même de ne jamais pratiquer la communication avec le procédé.

C.7 File System

Le système de fichier est un autre endroit qui est proposé par Microsoft. Il en parle plus spécifiquement dans le *Framework .Net*, car sur le système d'exploitation des ordinateurs de poches, le système de fichier n'est pas sur disque dur mais dans la mémoire. On peut développer des systèmes d'échanges de données entre les applications. Par contre, sur les ordinateurs ordinaires, le principe d'écriture et de lecture sur disque physique s'applique. Nous perdons un temps précieux à écrire sur le disque.

C.8 Database

Les bases de données que nous avons déjà parlées à l'annexe précédente. Dans la documentation, on dit que les protocoles seraient plus rapides pour transférer des données. Même avec l'optimisation des protocoles, nous devons toujours penser que la base de données va tout de même écrire sur le disque. Donc, que cela soit en *Win32* ou avec le *.Net* ou avec *Linux*, ce principe n'est pas trop envisageable pour des raisons de performance pour le partage des données.

ANNEXE D

LES ARTICLES

SOFTWARE OUT THERE

April 5, 2006

Software Out There

By JOHN MARKOFF

THE Internet is entering its Lego era.

Indeed, blocks of interchangeable software components are proliferating on the Web and developers are joining them together to create a potentially infinite array of useful new programs. This new software represents a marked departure from the inflexible, at times unwieldy, programs of the past, which were designed to run on individual computers.

As a result, computer industry innovation is rapidly becoming decentralized. In the place of large, intricate and self-contained programs like Microsoft Word, written and maintained by armies of programmers, smaller companies, with just a handful of developers, are now producing pioneering software and Web-based services.

These new services can be delivered directly to PC's or even to cellphones. Bigger companies are taking note.

For example, Google last month bought Writely, a Web-based word-processing program created by three Silicon Valley programmers. Eric Schmidt, the Google chief executive, said that Google did not buy the program to compete against Microsoft Word. Rather, he said, it viewed Writely as a key component in hundreds of products it is now developing.

These days, there are inexpensive or free software components speeding the process. Amazon recently introduced an online storage service called S3, which offers data storage for a monthly fee of 15 cents a gigabyte. That frees a programmer building a new application or service on the Internet from having to create a potentially costly data storage system.

Google now offers eight programmable components — elements that other programmers can turn into new Web services — including Web search, maps, chat and advertising. Yahoo offers a competing lineup of programmable services, including financial information and photo storage. Microsoft has followed quickly with its own offerings through its new Windows Live Web service.

Smaller companies are also beginning to share their technology with outside programmers to leverage their competitive positions. Salesforce.com, a fast-growing company that until recently simply offered a Web-based support application for sales personnel, published standards for interconnecting to its software not too long ago. That made it possible for developers inside and outside the

company to add powerful abilities to its core products and create new ones from scratch.

One result is that sales representatives using Salesforce's customer relationship management software to organize their workday can now make telephone calls using Skype, the popular Internet service, without leaving the Salesforce software.

The idea of modular software, where standard components can be easily linked together to build more elaborate systems, first emerged in Europe during the 1960's and spread to Silicon Valley in the 70's.

Despite its promise, however, modular software has generally been limited by corporate strategies that have held customers and other programmers hostage to proprietary systems.

Those limitations have eased almost overnight, mostly because of the open-source software movement, which promotes making information available to everyone.

The shift toward sharing, which in its grandest conception has been termed Web 2.0, has touched off a frenzy of software design and start-up activity not seen since the demise of the dot-com era six years ago.

"These tools are changing the basic core economics of software development," said Tim Bray, director of Web technologies at Sun Microsystems and one of the designers of a powerful set of Internet conventions known as Extensible Markup Language, or XML, which make it simple and efficient to exchange digital data over the Internet.

By lowering the cost of software development and thus the barriers to entering both existing and new markets, modular software is putting tremendous pressure on the corporations that have dominated the software industry.

It is also affecting Silicon Valley's venture capitalists. Start-ups have begun to bypass the venture capital firms, relying instead on individual investors, called "angels," or out-of-pocket financing, largely because new ventures are not as expensive.

In many cases, the start-ups do not even require the traditional Silicon Valley garage. The new companies are "virtual," and programmers work from home, relying on nothing more than a personal computer and a broadband Internet connection.

Early examples of the trend were tiny companies with significant ideas, like the consumer Internet software start-ups Flickr, a Web-based photo-sharing site, and Del .icio.us, which makes it possible for Web surfers to categorize and share things they find on the Internet. Both were acquired last year by Yahoo.

For some, the new era of lightweight, lightning-fast software design is akin to a guerrilla movement rattling the walls of stodgy corporate development organizations.

"They stole our revolution and now we're stealing it back and selling it to Yahoo," said Bruce Sterling, an author and Internet commentator.

Even more striking is the suggestion that a broad transformation of software development might reverse the trend of outsourcing to India, where highly skilled but low-paid programmers are plentiful.

"Transforming the economics of software development completely transforms the rationales for outsourcing," Michael Schrage, a Massachusetts Institute of Technology researcher, wrote in the current issue of CIO magazine.

The new economics of software development poses a fresh challenge to the dominant players in the industry. In 1995, when Microsoft realized that the Netscape Internet browser created a threat to its Windows operating system business, it responded by introducing its own free browser, Internet Explorer. By doing so, Microsoft, which already held a monopoly on desktop software, blunted Netscape's momentum.

Last November, Microsoft introduced a Web services portal called Windows Live and Office Live.

But as the world's largest software publisher, it still faces the delicate challenge of creating free Web services. Many of Microsoft's standard PC applications, in the new world of on-demand software, are migrating to the Internet.

At the Emerging Technologies Conference, held in San Diego last month, Ray Ozzie, one of Microsoft's three chief technical officers, showed a prototype effort that uses the Windows clipboard, which moves data among different desktop PC programs, to perform the same function for copying and transferring Web information.

Mr. Ozzie, who used the Firefox browser (an open-source rival to Internet Explorer) during his demonstration, said, "I'm pretty pumped up with the potential for R.S.S. to be the DNA for wiring the Web."

He was referring to Really Simple Syndication, an increasingly popular, free standard used for Internet publishing. Mr. Ozzie's statement was remarkable for a chief technical officer whose company has just spent years and hundreds of millions of dollars investing in a proprietary alternative referred to as .Net.

Moreover, the balance of power is shifting, Mr. Ozzie said. "For years, vendors like Microsoft have put huge resources into tools to build composite applications," he said. "With mash-ups, the real power becomes the people who can weave the applications together."

Microsoft is not the only company threatened by the simple tools of the Web 2.0 movement. Adobe Systems, which recently acquired Macromedia, publisher of the widely used Flash graphics standard, is under pressure from Ajax, or Asynchronous JavaScript and XML, a new development technique for creating interactive Web applications that look and function like desktop programs.

At the technology conference, Adobe showed a bridge between Ajax and Flash, making it possible for Ajax programmers to easily add Flash graphical abilities.

America Online has made a similar strategic shift by adding a set of "programmers' hooks" to its AOL Instant Messaging service to attract independent software developers to connect to its previously proprietary messaging platform.

Many technologists agree that as software development moves online, the risk will be particularly intense for large software

development organizations like I.B.M.'s Global Services, the consulting arm to the company, according to Mr. Bray of Sun.

I.B.M. is testing a faster development system based on Ajax, Web services and XML, said Rod Smith, the company's vice president for emerging technologies.

"We're testing it with customers now to see how disruptive it is," he said.

Mr. Smith acknowledged that the new software development trends present challenges. "Inside I.B.M., do-it-yourself software is an oxymoron," he said.

Another new idea comes from Amazon, whose Web Services group recently introduced a service called the Mechanical Turk, an homage to an 18th-century chess-playing machine that was actually governed by a hidden human chess player.

The idea behind the service is to find a simple way to organize and commercialize human brain power.

"You can see how this enables massively parallel human computing," said Felipe Cabrera, vice president for software development at Amazon Web Services.

One new start-up, Casting Words, is taking advantage of the Amazon service, known as Mturk, to offer automated transcription using human transcribers for less than half the cost of typical commercial online services.

Mturk allows vendors to post what it calls "human intelligence tasks," which may vary from simple transcription to identifying objects in photos.

Amazon takes a 10 percent commission above what a service like Casting Words pays a human transcriber. People who are willing to work as transcribers simply download audio files and then post text files when they have completed the transcription. Casting Words is currently charging 42 cents a minute for the service.

Other examples are also intriguing. A9, Amazon's search engine, is using Mturk to automate a system for determining the quality of photos, using human checkers. Other companies are using the Web service as a simple mechanism to build polling systems for market research.

The impact of modular software will certainly accelerate as the Internet becomes more accessible from wireless handsets.

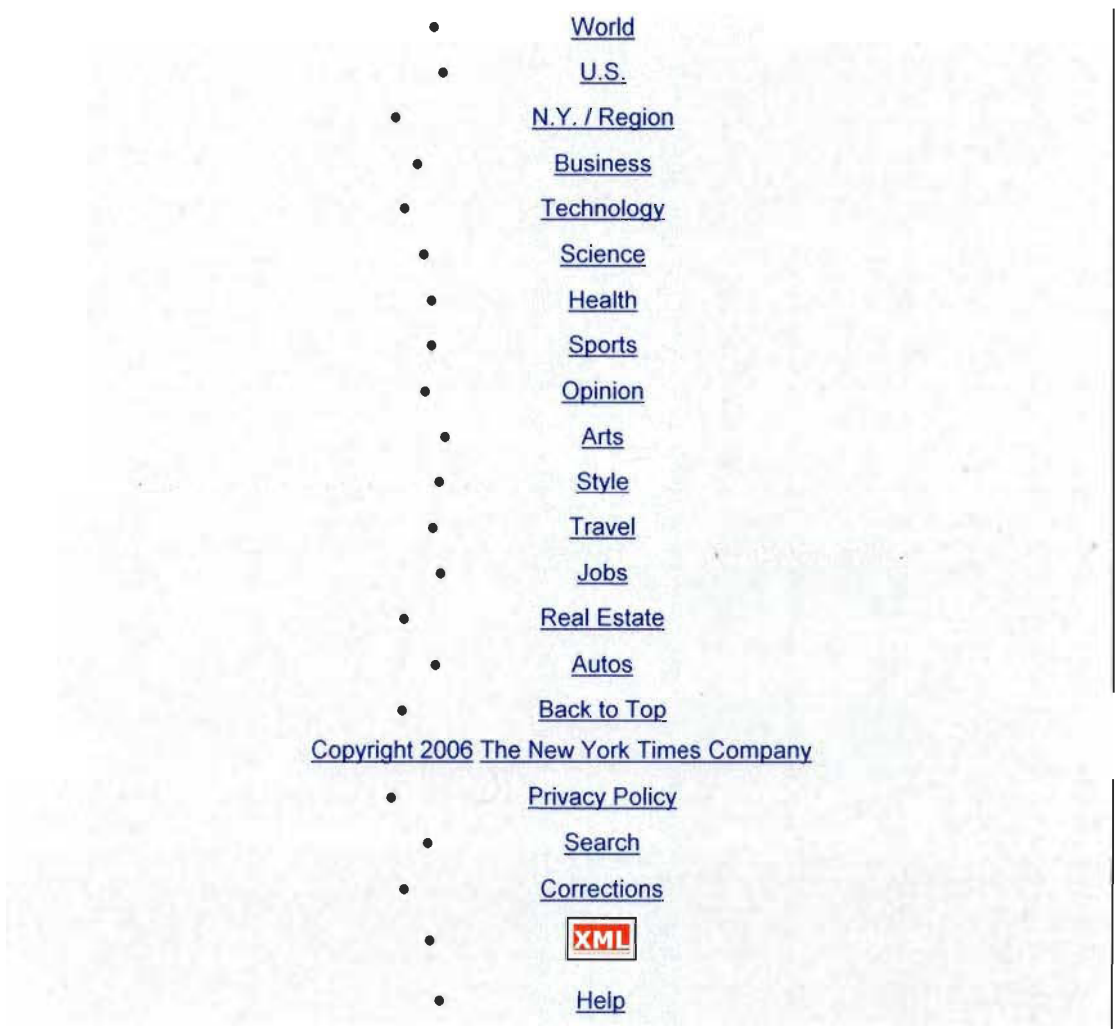
Scott Rafer, who was formerly the chief executive of Feedster, a Weblog search engine, has recently become chairman of Wireless Ink, a Web-based service that allows wireless users to quickly establish mobile Web sites from anywhere via Web-enabled cellphones.

Using modular software technologies, they have created a service called WINKsite, which makes it possible to use cellphones to chat, blog, read news and keep a personal calendar. These systems are typically used by young urban professionals who are tied together in loosely affiliated social networks. In London, where cellphone text

messaging is nearly ubiquitous, they are used to organize impromptu gatherings at nightclubs.

Recently, Wireless Ink struck a deal with Metroblogging, a wireless blogging service, to use its technology. Metroblogging, which already has blogs in 43 cities around the world, lets bloggers quickly post first-person accounts of news events like the July 2005 London bombings.

"Here are two tiny start-ups in California that care about Karachi and Islamabad," Mr. Rafer said. "It's weird, I'll grant you, but it is becoming increasingly common."



L'INTERNET DES LÉGOS

L'internet des Légos

Dans: Innovation, R&D, Economie et marchés, Enjeux, débats, prospective, Brèves - Par Hubert Guillaud le 14/04/2006

"L'internet entre dans l'ère des Légos. En effet, les blocs de composants logiciels interchangeables prolifèrent sur le web et les développeurs les ajustent ensemble pour créer un choix potentiellement infini de nouveaux programmes utiles. Ces nouveaux logiciels représentent un nouveau départ par rapport aux programmes du passé, peu flexibles, parfois difficiles à manier, conçus pour être installés sur chaque ordinateur", explique John Markoff du *New York Times* (enregistrement obligatoire). *"En conséquence, l'innovation du secteur est en train de devenir très décentralisée. Au lieu de grands programmes complexes et d'un seul bloc tels que Microsoft Word, écrits et maintenus par des armées de programmeurs, de plus petites compagnies, avec juste une poignée de développeurs, produisent maintenant des logiciels et des services web pionniers."* Né dans les années 60, malgré ses promesses, *"le concept de logiciel modulaire a longtemps été limité par les stratégies des entreprises qui ont pris en otage les consommateurs et les programmes avec leurs systèmes propriétaires."*

Cette transformation est en train de changer l'économie du développement logiciel, souligne Tim Bray, directeur des technologies web de Sun. Autre conséquence inattendue, indique Michael Schrage dans *CIO magazine*, il devient beaucoup moins intéressant de délocaliser la production de logiciels dans des pays à bas coût de main d'œuvre, puisque les logiciels ont cessé d'être d'énormes piles de lignes de code.

A moins, bien sûr, que les ingénieurs de ces pays ne s'adaptent rapidement à la nouvelle donne, ce qui paraît probable.

IS OFFSHORING CODING YESTERDAY'S FAD?

From: www.cio.com

Making IT Work - Is Offshoring Coding Yesterday's Fad?

– Michael Schrage, CIO

March 15, 2006

Over brunch in a cheap Brooklyn restaurant, a longtime MIT friend proudly demonstrated his latest startup's software. The idea is clever, and its beta implementation is sweet. I liked it; usually the stuff I see turns my stomach. So I'm pleased that Hans Peter Brondmo's Web-based "personal information organizer" has technical chops and global business potential.

Then again, I usually pay close attention to Brondmo's digital designs. He's not an uber-geek who'd rather write code than chat up prospects. A reasonably successful entrepreneur, he's a get-it-done pragmatist who won't coddle programming prima donnas. He wants to hit the market cheap, fast and hard with products that aren't hard to upgrade or maintain.

So when Brondmo told me his software, called Plum, was the first time he'd done serious coding in over a decade, I was taken aback. "I couldn't believe how much things have changed," he confided. "When my development teams wrote code 10 years ago, it took us three days to find and kill a bug. Today, it takes us only three hours."

What's more, he continued, whenever his (geographically distributed) development team runs into trouble, they can usually instant message their way into a just-in-time partnership that simultaneously solves the problem while alerting everyone to potential conflicts. "We do better real-time collaborative development and review now remotely then we did back at MIT when we were all in the same building," he notes.

Brondmo's favorite development discovery occurred when he was stuck for a few lines of code. He realized that by Googling he could see if anyone anywhere had posted something he could use. He and his team found quite a few virtual solutions this way. "But what about context?" I asked. After all, not everyone documents their C++ in English. He dismissively waved his hand: "Code is code. I found something that looked like what I needed in the middle of what looked like a bunch of Chinese. You paste it in and see what happens. It worked."

The ultimate result? He's never done a startup where the software development has been better, faster or cheaper. "In the past, I've had to raise lots of money to support the burn rate and the licenses necessary to develop real software over a couple of years; the costs are huge," he said. "You had to deal with the venture capitalists. They had the money.

"Development cost is still significant, but it's now focused on value creation, not infrastructure development," he added. "Open source and the availability of tools reduce our infrastructure cost. We don't have to pay for expensive software licenses and engineers to implement 'commodity' functions. So more money can be focused on innovation, not plumbing. We do more features faster. Development isn't really an obstacle."

Even allowing for hyperbole—perhaps Brondmo's "three days to three hours" time compression is really closer to "two days to five hours," we're still describing at least a fourfold productivity leap. That's impressive. Marry that to the evolving array of development-oriented communication, collaboration and search tools spilling into the global digi-sphere,

and the serious CIO might want to delay that Bangalore RFP. The new economics of software development may have rendered India and China yesterday's fad.

Plum's provenance may not be typical, but there's nothing extraordinary about it either. A savvy entrepreneur is exploiting technical innovation to cost-effectively generate technical innovation. The stuff works. This is where savvy CIOs need to sit up and take notice. The implementation implications are enormous.

I'm the last person to suggest that busy CIOs should immerse—or, God forbid, reimmerse—themselves in code. But any CIO preaching the gospel of productivity better know if his organization's methodologies discourage—or invite—healthy experimentation with these nascent development platforms. A CIO should know if he can now consistently get a year's worth of software development in 90 days. A CIO should know if 75 percent of a project portfolio can go to value-added features instead of infrastructure maintenance. This matters.

Transforming the economics of software development completely transforms the economic rationales for outsourcing. Reducing both the cost and time-to-market of new features and functionality completely transforms a company's economics of innovation. Ideally, CIOs should "own" these transformations. Do you?

Three clear implementation transformation scenarios emerge. The first scenario is the easiest and most obvious: These development economics create a new generation of Salesforce.coms and other ASPs that offer suites of mix-and-match business processes for enterprise consumption. For example, while Brondmo has given little thought to Plum as an enterprise "knowledge management" platform, it could easily be adapted to become one. With a little goosing, it could become an "account management" app too. More choice, less money.

Toward Value-Added Innovation

Scenario two has IT recommit to enterprise software development. These tools and technologies turn the internal economic equations for IT investment away from outsourcing and toward value-added innovation. IT becomes a better, faster and cheaper innovation partner for both key business units and core enterprise processes. ERP systems are goosed and spruced by customized Web apps instead of extended by packaged procurements like Siebel or PeopleSoft.

The third scenario has IT bypassed by ambitious business unit leaders who can't—or won't—wait for the CIO to get his act together. So they pursue scenario one and scenario two-type behaviors independent of whomever the CIO is and whatever the CIO wants. Like the rise of

the software spreadsheets more than 20 years ago, the rise of Plum-like digital platforms and processes proceeds without the need for central approval.

Scenario-three CIOs will have a hard choice: Either be seen as enablers and champions of creative enterprise interoperability or get used to losing a lot of fights.

My personal belief is that the variation IT's been witnessing since 2000 will accelerate: The "IT doesn't matter" crowd will continue to manage and invest in IT as a commodity, while the "strategic IT" companies will be exploiting these new development economics for better and faster differentiation, segmentation and innovation. These emerging economics will further fragment the CIO community. The rich will get richer; the smart smarter; and the not-so-rich and not-so-smart will find themselves struggling to remain "fast-followers."

When you look at the core economic dynamics driving software development and business competition, it seems painfully clear: There has never been a better time to be a smart CIO at an organization that wants to win.

L'OPEN SOURCE ET LE PROPRIÉTAIRE DOIVENT COHABITER PLUTÔT QUE DE S'AFFRONTER

ZDNet.fr

Éric Besson : « L'open source et le propriétaire doivent cohabiter plutôt que de s'affronter »

Le 30 juin, face à une assemblée d'éditeurs, en majorité propriétaires, le secrétaire d'État en charge de l'Économie numérique a joué la carte de la cohabitation des modèles open source et propriétaires. « Loin de s'affronter, ce sont deux mondes qui doivent coexister », a-t-il déclaré. Éric Besson s'exprimait, dans le cadre des Assises du logiciel, organisées par l'Association française des éditeurs de logiciels (Afdel).

Une déclaration faite par le ministre juste avant de quitter les lieux. Il s'en est suivi une levée de boucliers des éditeurs qui déploraient manifestement les multiples orientations de l'administration publique en faveur des solutions libres.

Rappelons notamment que les PC des députés tout comme ceux des gendarmes passent à Ubuntu, ou que le ministère de la Défense recommande Thunderbird et Postfix à ses fonctionnaires. Cela en attendant un futur référentiel général d'interopérabilité (RGI), qui

édictera un ensemble de règles que devront respecter toutes les administrations publiques ; il devrait faire la part belle à l'open source.

« L'État ne doit pas préconiser un choix de technologies »

En réaction à cette tendance favorable au libre, Pierre-Marie Lehucher, DG du groupe Berger Levrault qui édite des solutions pour les collectivités publiques, a rappelé que « notre métier c'est de créer des solutions, pas de rechercher des subventions ou de se conformer à un modèle imposé par l'État ».

Dans la même veine, Patrick Bertrand, président de l'Afdel et DG de Cegid, premier éditeur français de progiciels de gestion a déclaré : « Ce n'est pas à l'État à définir et à préconiser un choix de technologies, l'État est là pour définir un décor dans lequel nous allons évoluer. »

« Si l'interopérabilité est une bonne chose, elle s'imposera d'elle-même comme une évidence » a renchéri Jean Ferré, P-DG de Sinequa, éditeur de solution de recherche et de navigation pour les entreprises.

Seul éditeur open source à prendre la parole : Bertrand Diard, président de Talend, éditeur de logiciel pour l'intégration de données. Il a indiqué que la proposition de réserver une part des marchés publics à l'open source « relève d'un faux débat ».

L'Afdel représente 150 éditeurs dont seule une minorité possède des activités open source, mais se dit prête à accueillir d'autres éditeurs open source.

Article modifié le 03/07/2008

BIBLIOGRAPHIE

[Adler et Al. 1998] Adler Ronald B., Towne N., 1998, "Communication et interactions", 2ème édition, 2760706087

[Appleby et Al. 1991] Appleby D., Vanderkopple J., 1991 "Programming languages : paradigm and practice", 0-07-557904-9

[Aspray 1990] Aspray W., 1990, "John Von Neumann and the origins of modern computing" 0262011212

[Bergenti et Al. 2004] Bergenti F., Gleizes M.P., Zambonelli F., 2004, "Methodologies and software engineering for agent systems :the agent-oriented software engineering handbook", 1402080573

[Bergin et Al. 1996] Bergin T. J., Gibson G. R., "History of Programming Languages", 0201895021

[Biskri 1995] Biskri I., 1995, "La Grammaire Catégorielle Combinatoire Applicative dans le cadre de la Grammaire Applicative et Cognitive"

[Biskri et Al. 2002] Birkri I., Meunier J.-G., "SATIM : Système d'Analyse et de Traitement de l'Information Multidimensionnelle"

[Bjorner 2005] Bjorner D., 2005, "Software Engineering 1 : Abstraction And Modeling", 3540211497

[Blaess 2003] Blaess C., 2003, "Programmation système en C sous Linux" Signaux, processus, threads, IPC et sockets, 2212110545

[Booch et Al. 2000] Booch G., Rumbaugh J., "Guide de l'utilisateur UML", 2212091038

[Booch et Al. 1999] Booch G., Rumbaugh J., Jacobson I., 1999, "The unified modeling language user guide" 0201571684

[Bourbaki 2007] Bourbaki N., 2006, "Éléments d'histoire des mathématiques", 3540339388

[Bouzeghoub et Al. 1997] Bouzeghoub M., Gardarin G., Valduriez P., 1997, "Les Objets", 2212089570

[Bryan 1988] Bryan M., 1988, "Guide to the Standard Generalized Markup Language", 0201175355

[Bünning 2001] Bünning U. "ASP 3 Sites Web dynamiques" 2-7429-1712-8

[Cardone et Al. 2006] Felice Cardone, Hindley J. Roger, "History of Lambda-calculus and Combinatory Logic"

[Chaib-Draa 1999] Chaib-Draa B., Moulin B., Delisle B., 1999, "Analyse et simulation de conversations : de la théorie des actes des discours aux systèmes multiagents" 2907447238

[Chaib-Draa et Al. 2006] Chaib-Draa B., Muller Jorg. P., 2006, "Multiagent Based Supply Chain Management" 2907447238

[Chaib-Draa et Al. 2005] Chaib-Draa B., Labrie M.A., Bergeron M., Pasquier P., 2005, "DIAGAL: An Agent Communication Language based on Dialogue Games and Sustained by Social Commitments",

[Coplien 1999] Coplien James O., 1999, "Multi-Paradigm Design for C++", 0201824671

[Cox et Al. 1991] Cox J. B., Novobilski A. J., "Object-Oriented Programming", An Evolutionary Approach, 0201548341

[Cunningham et Al.] Cunningham H., Maynard D., Bontcheva K., Tablan V., "GATE: an Architecture for Development of Robust HLT Applications", Department of Computer Science, University of Sheffield

[Darlington et Al 1992] Darlington J., Dietrich R., 1992, "Declarative programming, Sasbachwalden 1991 :Phoenix Seminar and Workshop on Declarative Programming, Sasbachwalden, Black Forest, Germany, 18-22 November 1991" 3540197354

[Desfray 1997] Desfray P., 1997, "Modélisation par objets, la fin de la programmation" 222583119X

[Deitel 1999] Deitel 1999, "Comment programmer en C++ cours et exercices", 2893771580

[Deitel 1999] Deitel 1999, "Comment programmer en Java", 289377184X

[Desclés 1990] Desclés J.P., 1990, "Langages applicatifs, langues naturelles et cogniotion", 2866012275

[Epp 1995] Epp Susanna S., 1995, "Dicrete Mathematics with Applications", Second Edition, 0534944469

[Ferber 1995] Ferber J., 1995, "Les Système multi-agents", Vers une intelligence collective, 2-7296-0665-3

[Ferber 1997] Ferber J., 1997, "Les Système multi-agents", Vers une intelligence collective

[Filman 2005] Filman Robert E., 2005, "Aspect-Oriented software development", 0321219767

[Gamma et Al. 2005] Gamma E., Helm R., Johnson R., Vlissides J., "Design Patterns", 1405837309

[Gayton 2004] Gayton F., 2004, "Vers une nouvelle ingénierie de l'information"

[Gélinas 2005] Gélinas J.-F.., 2005, "Mesure de la cohesion dans les systems orientés aspect", 0494060573

[Gerhard 1999] Gerhard W., 1999, "A Modern Approach to Distributed Artificial Intelligence", 0-262-23203-0

[Godfrey et Al. 1993] Godfrey M.D., Hendry D.F., 1993, "The Computer as von Neumann Planned It", IEEE Annals of History of Computing, Vol. 15, No. 1, 1993

[Godin 2003] Godin R., 2003, "Systèmes de gestion de bases des données par l'exemple", 2921180480

[Gregor et Al. 1997] Gregor Kiczales, John Lamping, Anurag Mendhekar, Chris Maeda, Cristina Videira Lopes, Jean-Marc Loingtier, John Irwin, "Aspect-Oriented Programming",

[Gyorgy 1988] Revesz G., "Lambda-Calculus", Combinators and Functionnal Programming, 0521345898

[Hudak et Al. 2007] Hudak P., Hudges J., Peyton Jones S., Walder Philip., "A History of Haskell", Being Lazy With Class

[Ivar et Al. 2005] Ivar J., Pan-Wei Ng., 2005, "Aspect-Oriented software development with uses cases", 0321268881

[Jacobson 2000] Jacobson I., "Le Processus unifié de développement logiciel", 2212091427

[Kay 2000] Kay M. "XSLT 2ème édition" 2-7440-9004-2

[Kiczales et Al. 1997] Kiczales G., Lamping J., Mendhekar A., Maeda C., Videira Lopes C., Loingtier J.-M., Irwin J. "Aspect-Oriented Programming"

[Krivine 1990] Krivine J.-L., "Lambda-calcul, types et modèles", 2-225-82091-0

[Koch et Al. 2000] Koch D., Kürten O., Harms F., "HTML 4 XML-XHTML" 2-7429-1567-X

[Larman 2002] Larman C., 2002, "UML et les Design Patterns" 2844013013

[Leslie B. et Al 1993] Leslie B., Robert C., 1999 "Comparative Programming Language", 0201568853

[Leslie et Al. 2000] Leslie B. Wilson, Clark G. Robert, 2000, "Comparative Programming Languages", 0201710129

[Collin 1992] Collin Willian J., "Data Structures", An Object-Oriented Approach, 1st edition, 0201569531

[Mcgrath 1997] Mcgrath Sean, "Parseme. 1st: Sgml for Software Developers", 0134889673

[Meyer 2000] Meyer B., 2000`"Programmation Objet", 2212091113

[Michaelson 1989] Michaelson G., 1989, "An Introduction to Functional Programming Through Lambda Calculus", 0201178125

[Mitchell 1997] Mitchell T., 1997, "Machine Learning", 0071154671

[Mitchell 1996] Mitchell John C., 1996, "Foundations for Programming Languages", 0262133210

[Northrup 2006] Northrup T., Wildermuth S., Ryan B., 2006, "Les bases du développement d'application avec Microsoft .NET 2.0 Framework"

[Pasquier et Al. 2004] Pasquier P, Chaib-Draa B., 2004, "MODÈLES DES DIALOGUES ENTRE AGENTS", In Cognito Publications Vol. 1- No. 4 - 2004

[Ravi 1989] Ravi S, "Programming Languages", Concepts and Constructs 201-10365-6

[Ravi 1995] Ravi S, "Programming Languages 2ème édition", Concepts and Constructs 0-201-59065-4

[Schwarte 1996] Schwarte Joachim, "Le grand livre d'HTML", 2742906487

[Russel 1997] Russel S.J., 1997 "Rationality and Intelligence", Artificial Intelligence

[Sebesta 2001] Sebesta Robert W., 2001, "Concept of programming languages" 0201752956

[Sebesta 2004] Sebesta Robert W., 2004, "Concept of programming languages" 0201752956

[Sethi et Al. 1996] Sethi R., Stone T., 1996, "Programming Languages : Concepts and Constructs", 0201590654

[Siobhán 2005] Siobhán C., 2005, "Aspect-oriented analysis and design : the theme approach", 0321246748

[Sommerville 2004] Sommerville I., 2004 "Software Engineering", 0321210263

[Stansifier 1994] Stansifier R., 1994 "The Study of Programming Languages", 0137269366

[Stuart et Al. 1999] Stuart J. Russel, Norvig P., 1999, "Artificial Intelligence: A Modern Approach", 0137903952

[Tanenbaum 2003] Tanenbaum A., 2003, "Réseau", 2744070017

[Templeman et Al. 2002] Templeman J. Olsen A., 2002, "Microsoft Visual C++.NET, Étape par Étape", 2100065226

[Tischer 1997] Tischer M., 1997, "Bible PC Programmation Système", 2742915028

[Wegner 1988] Wegner, P. 1988 "Object-oriented concept hierarchies"

[Weiss 1999] Weiss G., 1999, "Multiagent System: A modern approach to Distributed Artificial Intelligence" 0262232030

[Weiss 2002] Weiss M.A., 2002 "Data Structures", Problem Solving using JAVA, 0-201-74835-5

[Wexelblat 1981] Wexeblat R. L., "History of Programming Languages", 0127450408

[Wielsch et Al. 2000] Wielsch M., Prahm J., Esser H.G., 2000, "Linux, La bible" Administration – Réseau TCP/IP, Intranet – Programmation

[Witten et Al. 2005] Witten I., Eibe F. "DATA MINING Practical machine Learning Tools and Techniques" 0-12-088407-0

[Wooldridge 2002] Wooldridge M., 2002, "An Introduction to MultiAgent System" 0201360489