الجمهورية الجزائرية الديمقراطية الشعبية
وزارة التعليم العالي و البحث العلمي
جامعة العقيد الحاج لخضر – باتنة

République Algérienne Démocratique et Populaire
Ministère de l'enseignement supérieur et de la recherche scientifique

Université Colonel Hadj Lakhdar - Batna
Faculté des Sciences de l'ingénieur كلية الهندسة
Département d'informatique قسم الإعلام الآلي

Une approche de modélisation des systèmes à évènements discrets utilisant les concept-maps

Thèse

Présentée et soutenue publiquement le :

20 Février 2011

Pour l'obtention du grade de :

Doctorat en science

(Spécialité : informatique)

Par

BOUROUIS Abdelhabib

Devant le jury composé de :

Président	: BENMOHAMMED	Mohamed	Professeur	Université de Constantine
Rapporteur	: BELATTAR	Brahim	M. C. A	Université de Batna
Examinateurs	: BILAMI	Azeddine	Professeur	Université de Batna
	: MAAMRI	Ramdane	M. C. A	Université de Constantine
	: ZIDANI	Abdelmadjid	M. C. A	Université de Batna
	: CHAOUI	Allaoua	M. C. A	Université de Constantine

Université de Batna - 2011 -

Remerciements

C'est difficile de remercier ici toutes les personnes grâce à qui j'ai pu mener à bien toutes ces années de recherche ; une période de rencontre et d'échange qui ne peut se résumer à une simple liste de noms.

Néanmoins je tiens tout d'abord à remercier **Dr BELATTAR Brahim** maître de conférences à l'université de Batna et mon directeur de thèse de doctorat, qui m'a initié à la recherche scientifique depuis 2001 et qui n'a jamais cessé de m'encourager et de m'inciter. Il m'a énormément appris pendant ces années, tant sur le plan scientifique que sur l'organisation de la vie d'un «chercheur», la manière d'écrire une publication, de valoriser son travail et de défendre ses idées.

Je remercie chaleureusement les membres du jury de thèse :

Dr BENMOHAMMED Mohamed Professeur en informatique à l'Université de Constantine pour m'avoir fait l'honneur de présider mon jury de thèse, ainsi que **Dr CHAOUI Allaoua** maître de conférences à l'Université de Constantine, **Dr BILAMI Azeddine** Professeur en informatique à l'Université de Batna, **Dr ZIDANI Abdelmadjid** maître de conférences à l'Université de Batna et **Dr MAAMRI Ramdane** maître de conférences à l'Université de Constantine pour avoir accepté de juger ce travail.

Bien sûr, je tiens à remercier ma famille, mes amis et mes collègues à la fois aux départements d'informatique de l'université de Batna et de l'université d'Oum el Bouaghi, pour leur présence dans les moments de joie et leur soutien dans les moments de peine.

Je tiens à remercier vivement **Mr KERKOUCHE El Hilali** pour les discussions scientifiques fructueuses, ainsi que **Dr BOUMEZBEUR Abderrahmane** et **Dr BOUTTOUT Farid** pour la relecture et la correction des articles que j'essayais tant bien que mal d'écrire dans un anglais correct.

Enfin, je remercie toute personne ayant contribué de loin ou de près à l'élaboration de ce travail.

BOUROUIS Abdelhabib.

Dédicace

Je dédie ce modeste travail à
Mes sacrés très chers parents
Ma femme ***Nedjoua***
Mes sœurs et frères
A ma nièce, la regrettée Ichrak
Ainsi qu'à tous ceux qui m'aiment.

Résumé-Abstract

Résumé

Les travaux réalisés dans ce manuscrit s'inscrivent dans le cadre de l'ingénierie dirigée par les modèles. En partant d'une investigation des possibilités d'utilisation des concept-maps dans les différentes étapes d'un projet de modélisation et de simulation, nous avons développé un formalisme visuel sous forme de concept-map pour la modélisation des systèmes à évènements discrets. Ce formalisme baptisé EQNM^{2}L s'inspire de la théorie des réseaux de files d'attente et s'avère extensible et expressif. Un format d'échange standard basé sur XML a été proposé pour favoriser l'interopérabilité entre les outils de simulation et d'analyse. Dans la perspective d'automatiser la génération du code de simulation (ou d'analyse), la transformation de modèles nous a servi de moyen pour obtenir un code Java complet selon la bibliothèque Japrosim développé aussi dans le cadre de ces recherches. La transformation de modèles a été réalisée utilisant XSLT.

Mots clés

Concept-map, Systèmes à Évènements Discrets, Langages Visuels, Explication Participative, Modélisation, IDM, DSML, Théorie des files d'attente, Évaluation de performances.

Abstract

Research accomplished in this thesis adhere to the model driven engineering. Starting with an investigation of possible uses of concept-maps in different stages of a modeling and simulation project, we developed a visual formalism as a concept map for modeling discrete event systems. This formalism called EQNM^{2}L based on the theory of queueing networks and is extensible and expressive. A XML-based standard exchange format has been proposed to promote interoperability between simulation and analysis tools. In the context of automated simulation (or analysis) code generation, model transformation has served as a means to obtain a complete Java code using the Japrosim library also developed through this research. The model transformation has been achieved using XSLT.

KeyWords

Concept-map, Discrete Event Systems, Visual Languages, Participatory Explanation, Modeling, MDE, DSML, Queueing Theory, Performance Evaluation.

Introduction générale

Problématique

La dimension sociale prend constamment de la place au sein des environnements de modélisation et de simulation (EMS), qui s'orientent récemment davantage vers un travail de groupe, en exploitant à fond les concepts de TCAO. La collaboration est un besoin inhérent dans les projets de simulation, durant ses différentes phases, quelque soit le domaine visé par l'étude. La pluridisciplinarité de la simulation fait que la nature du groupe impliqué dans un projet soit hétérogène, du chef de projet au modélisateur, en passant par le statisticien, le programmeur et effectivement le client de l'étude. Il est rare de trouver un cadre possédant toutes les connaissances nécessaires pour réaliser la totalité du travail.

Les intervenants dans un projet de simulation ont besoin de communiquer et de coopérer. Un formalisme de modélisation à la fois simple et expressif devient de plus en plus une nécessité. D'une part, le client qui n'a pas à priori des connaissances dans le domaine de la modélisation et de la simulation, cherche souvent à exprimer ses exigences et sa vision du problème rencontré aux moyens des techniques informelles, ou semi formelles. Un tel formalisme n'arrange pas les spécialistes du domaine de la simulation, parce qu'il nécessite un immense travail durant les premières étapes du projet de simulation. L'obtention d'un modèle conceptuel exploitable par la suite, n'est pas aisée. D'autre part, le modèle conceptuel obtenu, même sous sa forme graphique est souvent difficile à communiquer, soit avec le client ou bien avec les autres membres du groupe impliqués dans l'étude.

Les formalismes de modélisation actuels, utilisés pour les systèmes à évènements discrets, manquent aussi de :

1. L'aspect formel : les langages utilisés présentent des lacunes sur le plan formel (syntaxes et sémantiques). Ceci rend difficile l'apprentissage, l'évolution, la construction de compilateurs, interpréteurs, environnements d'édition, débogage, …etc.
2. L'interopérabilité : Le stockage et l'échange des modèles conçus se fait dans des formats propriétaires. Ceci n'encourage pas l'interopérabilité entre les différents outils de simulation et d'analyse.
3. La capacité de produire des scénarios explicables : pareil à ce qu'on trouve essentiellement dans le domaine de l'intelligence artificielle. Contrairement aux systèmes experts qui sont capables d'expliquer et de justifier les résultats qu'ils obtiennent, les simulateurs ne possèdent malheureusement pas cette faculté. L'étude que nous avons menée auparavant [Bourouis 03] montre bien que cette incapacité dépend essentiellement du formalisme de modélisation utilisé, ainsi que de l'absence, par mégarde, de ce besoin durant la conception et le développement des EMS.

Il s'avère donc très intéressant de définir un outil permettant d'aider à la modélisation des systèmes à évènements discrets à la fois visuel, simple, expressif, compréhensible et utilisable par les clients et les spécialistes. Il servira comme support à l'élaboration de modèles conceptuels, leur simulation et leur communication, ainsi qu'à la présentation des explications, tout en facilitant l'enseignement et l'apprentissage de la simulation.

Les concepts-maps représentent un candidat favori pour notre investigation. Ils sont une technique de représentation spatiale d'idées et des relations entre ces idées. La plupart du temps un concept-map est constitué de symboles et de traits qui les relient. Les symboles contiennent des textes courts ou des mots clés. Les traits peuvent aussi être nommés pour mieux qualifier les relations entre les symboles qu'ils relient. Un concept-map

peut aussi utiliser une métaphore géographique comme continent, pays, ou structurelle comme bâtiment.

Les concepts-maps sont surtout utilisés pour la communication de concepts et idées à l'intérieur d'un groupe de personnes. Les ingénieurs informaticiens, par exemple, connaissent bien les formes spécialisées de concept maps que sont les notations utilisées dans le génie logiciel tel qu'UML, OML, Booch...etc. Dans le domaine éducatif, les concepts map sont largement utilisés dans beaucoup de programmes.

Objectifs spécifiques

Dans cette thèse, nous nous intéressons à l'étude des possibilités d'utilisation des Concept-maps dans la modélisation des systèmes à évènements discrets. Il s'agit de réfléchir sur la possibilité de les adapter au domaine de la simulation pouvant aussi bien aider les concepteurs de modèles que les utilisateurs de ceux-ci. La validation des recherches devra nécessairement passer par la réalisation d'un environnement logiciel supportant le formalisme proposé. Cette solution doit impérativement exploiter les nouvelles technologies du Web et de l'internet. Nous prenons en compte la dimension de l'explication dite participative (Participatory Explanation) qui a très bien montré son adéquation avec les concepts-maps. Ce choix nous obligerait par conséquent à focaliser nos recherches sur les techniques de simulation dite participative (Participatory Simulation).

Le formalisme proposé devra répondre aux critères suivants :

1. Faire partie des langages visuels : une notation graphique proche des concept-maps qui s'inscrit dans le cadre des langages diagrammatiques.
2. Avoir une base formelle solide : tant sur le plan syntaxique que sémantique, avec en particulier une syntaxe abstraite bien établie.
3. Étendre le formalisme de réseaux de files d'attente de façon à améliorer l'expressivité. Cela permet de prendre en compte un éventail encore plus large des systèmes à évènements discrets modélisables.
4. Suivre le cycle de développement des langages de modélisation spécifiques au domaine (DSMLs).
5. Suivre un développement basé sur la méta-modélisation.
6. Avoir un format d'échange basé sur XML pour favoriser l'interopérabilité.

Organisation de la thèse et du travail

Ce manuscrit est organisé en deux parties, la première traite les aspects théoriques et la seconde ceux liés à la conception et à l'implémentation. Le premier chapitre explore les fondements des Concept-maps en abordant les aspects psychologiques, épistémologiques et pédagogiques avec un survol des variations rencontrées dans la pratique. Le second chapitre aborde le domaine des langages visuels et en particulier le sous ensemble des langages spécifiques au domaine (langages dédiés). Parmi ces derniers, les langages de modélisation sont étudiés sur le plan architectural et de conception. Le troisième chapitre est consacré à l'ingénierie dirigée par les modèles et le développement des DSLs. Le quatrième chapitre s'intéresse aux différentes étapes d'un projet de simulation en essayant de proposer dans chacune une variante de concept-maps adéquate. Le cinquième chapitre s'intéresse à la conception et à l'implémentation d'un noyau de simulation à évènements discrets qui représente le plus bas niveau d'abstraction dans une architecture DSM (code) nommé Japrosim. Le dernier chapitre est consacré au formalisme visuel proposé qui représente le haut niveau d'abstraction permettant de créer des modèles conceptuels. Il aborde sa conception selon une méthodologie bien définie en tant que DSML. En plus, le développement d'un environnement de modélisation et graphique et la génération automatique du code de simulation sont traités en détail .

Table des matières

Table des figures

Liste des tableaux

Première partie

Concept-maps et langages visuels

Chapitre 1

Fondements des concept-maps

Sommaire

1.1 Introduction

Les travaux en psychologie sur la mémoire et la représentation des connaissances émergèrent durant les années 60 et vers la fin de cette décennie. Plusieurs techniques sont proposées, parmi lesquelles les mind-maps et les concept-maps. Dans ce chapitre nous allons aborder l'historique et l'origine des concept-maps, étudier leurs fondements théoriques et explorer leurs domaines d'application. Nous faisons une comparaison des concept-maps avec les autres techniques similaires afin de pouvoir les identifier sans ambigüité.

1.2 Origines

L'invention des concept-maps remonte à 1972. A l'époque, le professeur Novak et son équipe de recherche travaillaient sur un projet dans le domaine de l'éducation à l'université Cornell aux états unis. L'objectif était de développer un outil permettant de décrire les changements explicites de la compréhension conceptuelle chez les enfants.

«In our discussions, the idea developed to translate interview transcripts into a hierarchical structure of concepts and relationships between concepts, i.e., propositions. The ideas developed into the invention of a tool in 1972 we now call the concept map ». [Novak et Canas 06].

Les concepts sont arrangés de façon hiérarchique, du plus général, plus inclusif en haut, au plus spécifique et moins général en bas. Un concept est généralement représenté par un ou deux mots dans un nœud. Les concepts sont reliés par des lignes pour créer des affirmations ou des propositions. Les Propositions sont des déclarations concernant des évènements ou des objets, montrant une relation entre deux concepts ou plus. Les concept-maps sont basés sur une psychopédagogie cognitive explicite et une épistémologie constructiviste. Cette idée est basée sur la théorie d'assimilation constructiviste d'Ausubel qui dit :

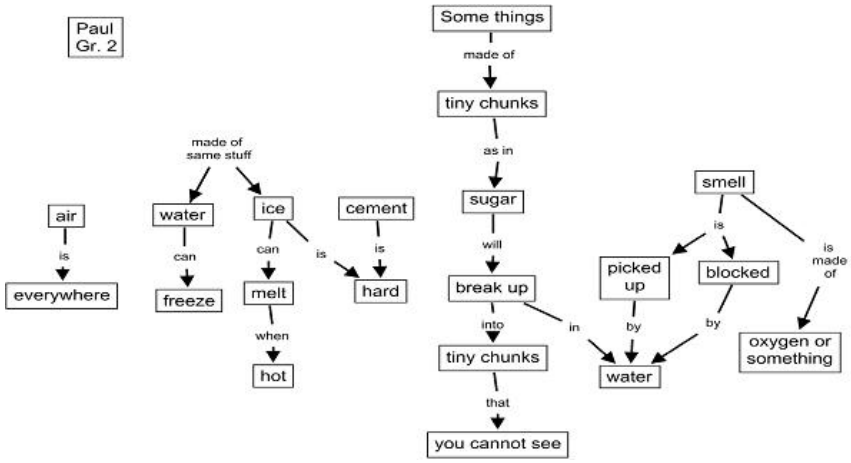


Figure 1.1: *Concept-map construit depuis des interviews avec un étudiant excellent de grade 2 [Novak et Canas 06].*

«The most important single factor influencing learning is what the learner already knows. Ascertain this and teach accordingly.» [Ausubel 63].

Novak et son équipe utilisaient alors ce nouvel outil pour extraire depuis les interviews bien structurés, le niveau du savoir acquis par un étudiant à n'importe quel point de son parcours pédagogique, ce que la figure 1.1 illustre clairement. Les changements dans les structures de connaissances de l'étudiant au fur et à mesure qu'il progresse sont devenues claires. Les concept-maps des apprenants peuvent être comparées avec ceux élaborés par des spécialistes du domaine, permettant de vérifier la validité des notions et concepts acquises. A cette époque, l'élaboration de concept-map était manuelle utilisant un crayon et une feuille blanche.

Novak a donné un cours pendant 20 ans à l'Université Cornell, qui a fait l'objet de son livre Apprendre comment apprendre (Learning How to Learn), en se servant des concept-maps, pour un apprentissage significatif et compréhensible. Il a été invité en 1987 par Bruce Dunn à l'université West Florida et leur coopération a permis en 1990 d'adopter les concept-maps comme outil pour l'élicitation des connaissances à l'IHMC (Institute for Human and Machine Cognition[1]) ainsi que comme composant d'explication dans les systèmes experts. La technologie web a permis également de développer le logiciel client-serveur nommé Cmap-Tools et l'émergence des concept-maps numériques. CmapTools Network constitue actuellement un réseau de serveurs et de clients permettant le travail coopératif synchrone/asynchrone et le partage de concept-maps.La NASA a mis en ligne une grande quantité de documents et d'informations concernant le projet *Mars Exploration*[2] illustré par la figure 1.2 tirée de [Briggs et al 04]. Sun a élaboré aussi un concept-map qui décrit toute la technologie Java[3] destinée aux débutants pour avoir une idée claire sur la puissance de ce langage.

1.3 Fondements psychologiques

Dès la naissance, l'être humain commence à acquérir, par observation des régularités des objets et évènements qui l'environnent, des concepts à chaque fois nouveaux. A partir de l'âge de 03 ans il peut acquérir en utilisant le langage (apprentissage par réception : reception learning). Les nouveaux concepts sont assimilés en posant des questions et en obtenant des clarifications qui mettent en valeur le lien entre ceux-ci et ceux déjà acquis. Ceci constitue un savoir transmis ou réceptif contrairement au savoir issu de la découverte autonome ou apprentissage heuristique [Ausubel 68] et [Macnamara 82].

La mémoire de l'être humain ne constitue pas un seul bloc à remplir, mais plutôt un ensemble complexe de systèmes mémoire interconnectés. Les informations perçues sont organisées et traitées par la mémoire de travail qui interagit avec les connaissances la mémoire long-terme. La mémoire de travail est caractérisée par

1. Florida Institute for Human and Machine Cognition : http://www.ihmc.us/
2. Mars Exploration Concept Map Contents : http://cmex.ihmc.us/cmex/table.html
3. Java Technology Concept Map : http://java.sun.com/new2java/javamap/intro.html

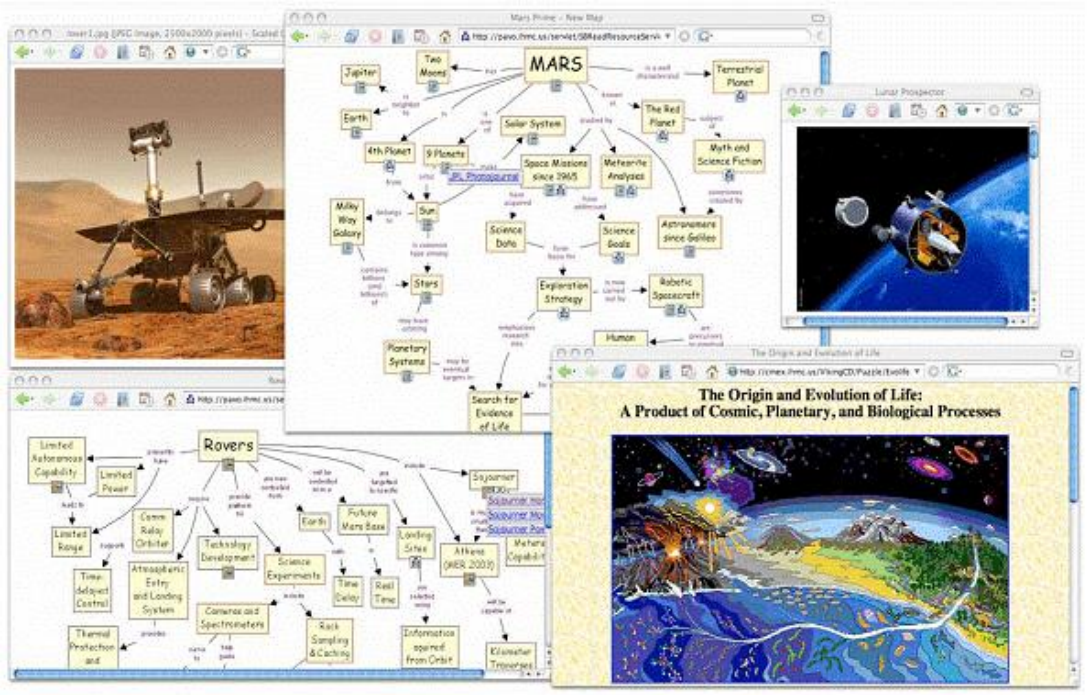
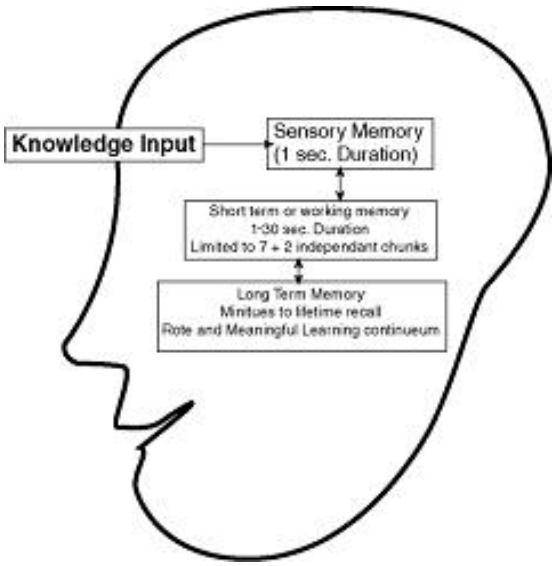


FIGURE 1.2: *Concept-map pour NASA Mars Exploration [Novak et Canas 06].*

une capacité limite de traitement à un moment donnée (5 à 9 unités psychologiques) [Novak et Canas 08]. Les processus-mémoires et la façon dont les connaissances sont incorporées dans le cerveau humain sont mal connus. Ce qui est évident, c'est l'organisation hiérarchique des concepts dans nos cerveaux et les techniques d'apprentissage doivent donc prendre ce constat en considération, voir la figure 1.3.

L'apprentissage peut être significatif ou bien par répétition (mécanique ou machinal) à ne pas confondre avec les techniques d'enseignement. Un apprentissage significatif ne s'obtient pas systématiquement par un apprenant qui pose des questions et demande plus d'information, sauf si celui-ci possède déjà les concepts de base nécessaires. La figure suivante illustre bien ces notions.

Les deux façons d'apprendre permettent le stockage des informations dans la mémoire long-terme. Contrairement à l'apprentissage significatif, les informations n'auront pas de liens avec les connaissances et les concepts déjà acquis dans l'apprentissage machinal, ce qui peut provoquer leur perte. La structure cognitive des connaissances de l'apprenant ne s'améliore pas.



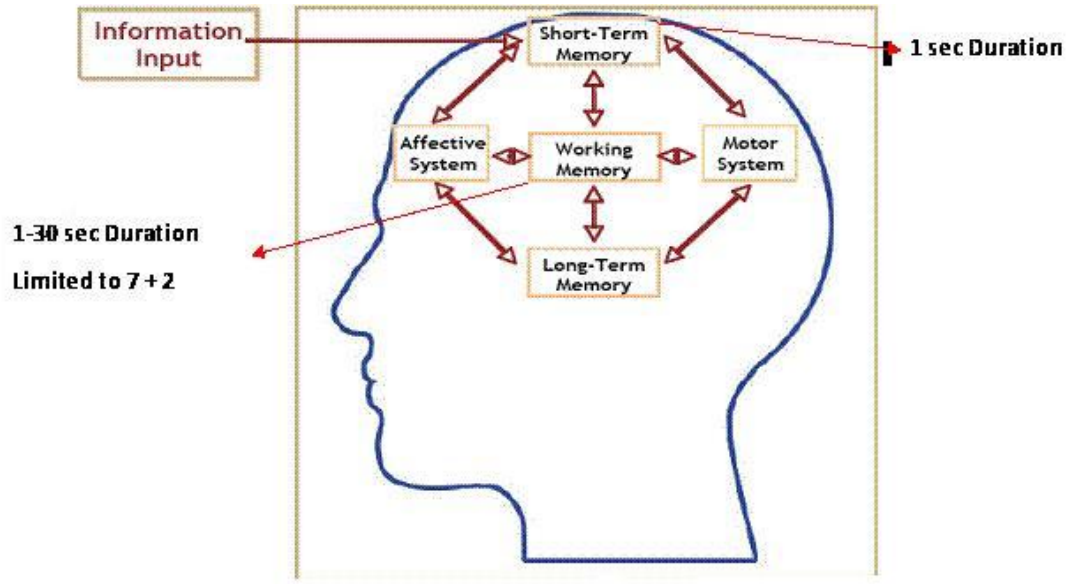


FIGURE 1.3: *Les parties intéressantes du cerveau humain qui interagissent lors de l'apprentissage [Novak et Canas 08].*

Beaucoup de recherches dans le domaine de la psychologie ont montré que les concepts sous forme audio et/ou visuelle sont bien assimilés et mémorisés. Les concept-maps facilitent l'apprentissage significatif où les nouveaux concepts introduits sous différentes formes (textuelle, symbolique, sonore, ...etc) sont assimilés l'un après l'autre de façon constructiviste, progressive et hiérarchique. Ils offrent ainsi un moyen efficace de bâtir un savoir et de le conserver de façon durable [Novak et Wandersee 91].

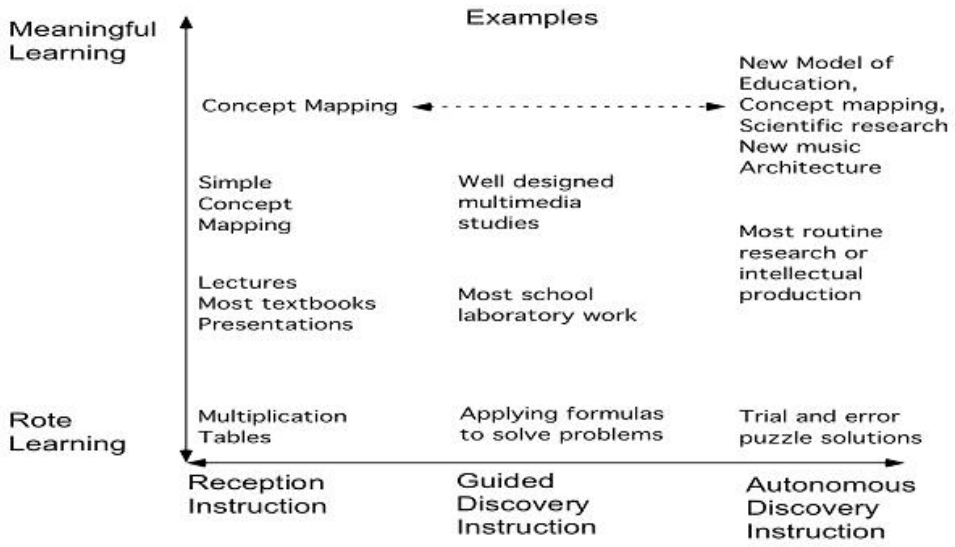


FIGURE 1.4: *Relation entre apprentissage et enseignement [Novak et Canas 08].*

1.4 Fondements épistémologiques

L'épistémologie est une branche de la philosophie qui traite la nature des connaissances et leur création. Actuellement, il y a un consensus entre philosophes et épistémologues en ce qui concerne la création de nouvelles connaissances. C'est un processus constructif, qui implique nos connaissances déjà acquises et nos émotions pour créer de nouvelles notions ainsi que les manières de les représenter.

La création de la méthode concept-mapping a conduit à de nouvelles opportunités dans l'étude du processus

d'apprentissage et à la création de nouvelles connaissances en incitant la créativité.

1.5 Définition

Le concept-mapping est un terme générique qui cache derrière une panoplie de langages visuels. C'est un outil de représentation et de visualisation de connaissances. Il est avant tout un graphe composé de nœuds étiquetés et d'arcs qui peuvent être aussi étiquetés. En faisant une analogie avec la théorie des graphes, le terme nœud en concept mapping est appelé sommet (en anglais vertex) dans la théorie des graphes. De même, le terme arc est appelé arête (en anglais edge).

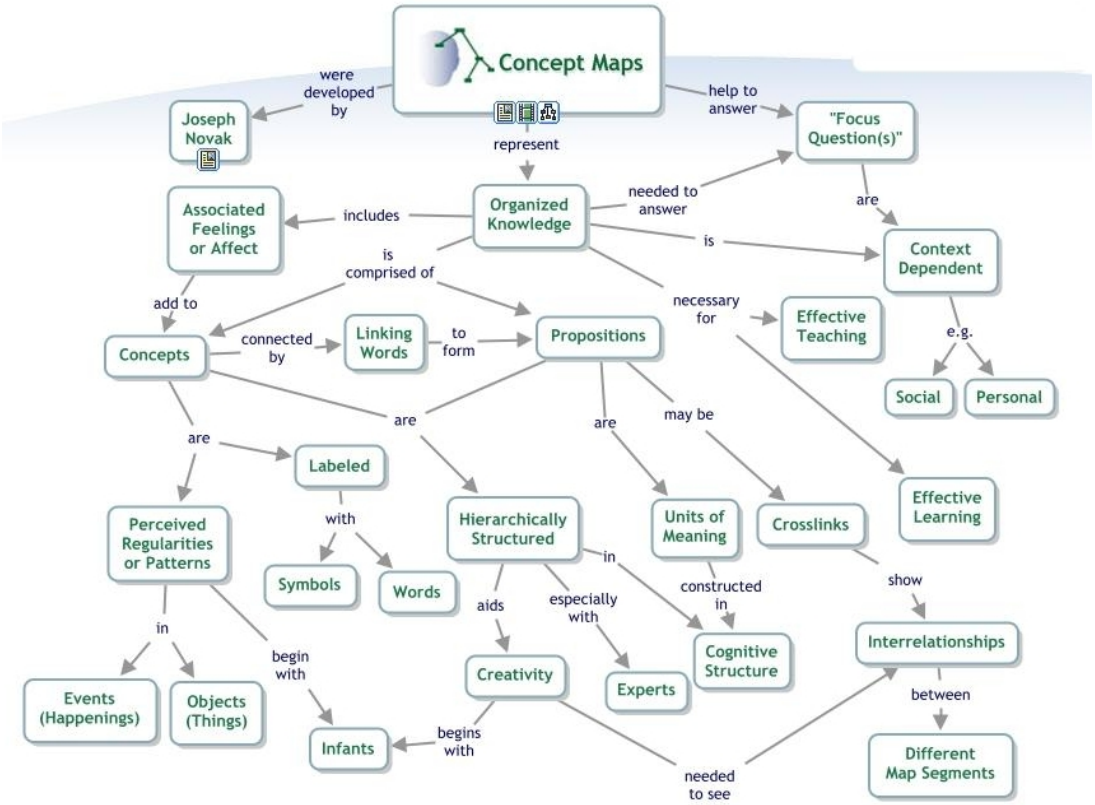


FIGURE 1.5: *Un concept-map sur les concept-maps du site http ://cmap.ihmc.us*

Les concept-maps ne sont pas tous informels. D'une part, leurs catégories informelles sont obtenues lors du manque de contraintes. Elles sont utilisées par exemple dans le domaine de l'éducation, l'apprentissage, brainstorming, acquisition de connaissances et la prise de notes dans les réunions. D'autre part, les catégories formelles subissent de fortes contraintes et sont élaborés par des experts. Elles se prêtent bien au traitement automatique par ordinateur et sont utilisées essentiellement dans pour la création de systèmes experts [Gaines 91], voire mêmes en tant que langages de programmation [Lukose 93].

Le passage du formel à l'informel n'est pas discret. Il est continu en passant par le semi-formel qui est maniable par l'humain et la machine à la fois, comme par exemple la notation Booch utilisée dans la conception orienté-objet.

1.6 Création de concept-maps

Il existe plusieurs méthodes pour la création d'un concept-map. Pour ce faire, on peut servir d'un crayon et d'un bout de papier, ou bien se servir d'un logiciel approprié. La méthode à suivre dépend aussi du domaine d'application. Celle décrite dans [Novak et Gowin 84] se présente sous forme d'une séquence d'étapes :

1. Définir une question sur un sujet particulier. Les concept-maps qui tentent de couvrir plus d'une question peuvent devenir difficiles à gérer.

2. Une fois le sujet principal est bien défini. Identifier une liste de concepts généraux, les plus importants, associés au sujet. Souvent 15 à 20 concepts suffisent.

3. Les concepts sont ordonnés de haut en bas, allant du plus général au plus spécifique. Ce qui favorise l'aspect hiérarchique (intéressant dans les concept-maps).

4. Les liens sont rajoutés par la suite pour lier les concepts et obtenir une ébauche.

5. Les liens sont annotés pour décrire les relations entre concepts.

6. Identifier les liens croisés entre concepts. Ils relient les concepts issus de champs différents ou des sous-domaines.

7. Finalement, le concept-map est révisé afin d'améliorer sa structure ou son contenu. Généralement plus de trois révisions mènent à un concept-map de bonne qualité.

1.7 Domaines d'application

Les concept-maps tels qu'introduits par Novak ont beaucoup servis dans le domaine de l'apprentissage et l'éducation. Néanmoins, d'autres domaines ont pu exploiter cet outil pour d'autres tâches plus spécifiques. Nous aborderons dans le chapitre suivant, lorsque cette technique sera étudiée dans le contexte plus générique des langages visuels, d'autres champs d'application. D'après [Novak et Gowin 84], les concept-maps sont utiles particulièrement pour :

- ✍ Faciliter l'apprentissage significatif : favorisé par le processus de construction d'une carte conceptuelle qui est si puissant.
- ✍ Aider à organiser les connaissances et à les structurer. La difficulté à construire des cartes conceptuelles et à les utiliser apparait essentiellement comme le résultat de plusieurs années de pratique d'un apprentissage machinal.
- ✍ Faciliter l'apprentissage coopératif : les recherches ont montré que lorsque les étudiants travaillent en petits groupes et coopèrent en essayant d'apprendre une matière, des résultats cognitivement et affectivement positifs en découlent. Le même constat peut être obtenu lorsque l'apprentissage vise à élaborer un concept-map de façon coopérative.
- ✍ Améliorer la qualité de l'enseignement : ils peuvent être également utilisés par les enseignants dans le but de construire leurs programmes d'enseignement. En faisant apparaitre les concepts principaux à enseigner, un concept-map incite l'enseignant à sélectionner les informations essentielles, à les structurer et par conséquent à organiser la manière dont elles doivent être transmises. Ce type d'organisation préalable, présentée régulièrement aux étudiants, favoriserait un apprentissage structuré.
- ✍ Faciliter l'évaluation de l'apprenant : Lorsque les concept-maps sont utilisés dans l'enseignement, ils peuvent également être utilisés pour l'évaluation. Ici, l'apprenant doit avoir un minimum d'expérience dans l'élaboration de concept-maps avant d'être évalué utilisant cet outil.
- ✍ Faciliter la communication : en favorisant l'échange des idées et des connaissances sous forme visuelle.
- ✍ Brainstorming (remue-méninges) : recherche des idées employées dans un domaine. Réunion où chacun fournit ses suggestions pour résoudre un problème.
- ✍ Aider à la représentation visuelle des connaissances : souvent préféré à une autre forme de représentation qui peut être destinée à un traitement automatique par machine et sera mal assimilée par les humains.
- ✍ Aider à la gestion de connaissances : en présentant les connaissances sous une forme organisée et hiérarchique avec les relations qui les attachent.

1.8 Techniques similaires

1.8.1 Mind Map

Il y a souvent confusion entre « mind map » et « concept map ». Les deux termes sont utilisés de façon interchangeable. Cependant ils présentent des différences essentielles :

1. A l'encontre d'un concept map, un mind map possède un concept central (concept clé).
2. Un mind map est utilisé essentiellement pour la prise de notes, tandis qu'un concept map se prête bien à l'éducation.

Bien que l'utilisation des mind-maps date depuis des siècles, le psychologue Tony Buzan a développé le mind-mapping et a obtenu son copyright [Buzan 89], [Buzan 91] et [Buzan 94]. Il le définit comme « un mot ou concept central, autour duquel sont dessinés de 5 à 10 sous concept. Chacun des sous concepts est à son tour relié à un groupe de 5 à 10 autres concepts. Le même processus peut se poursuivre de la même manière.».

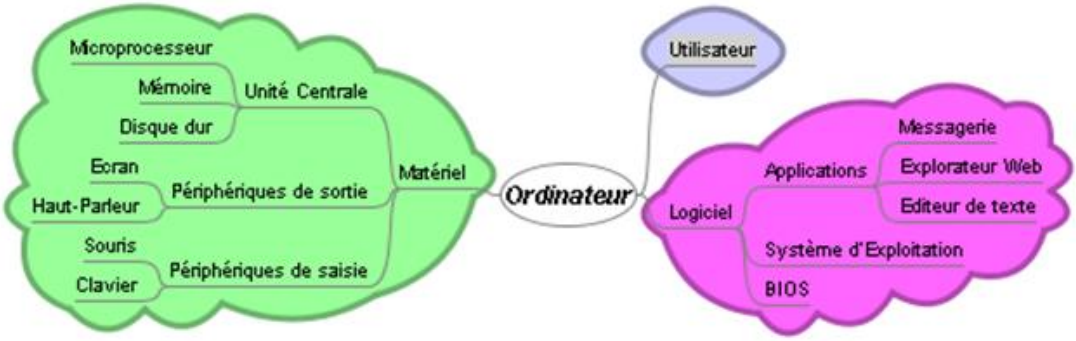


FIGURE 1.6: *Exemple de Mind Map sur l'ordinateur.*

Il est clair que le mind mapping est un dérivé du concept mapping qui se pratique de façon intuitive et informelle. Il est supporté par une gamme variée d'outils logiciels comme MindManager[4], Inspiration[5] et Axon Idea Processor[6].

1.8.2 Cluster Map

Le Clustering et le webbing sont des techniques qui capturent les associations entre les idées. Le Cluster mapping est développé comme une technique créative d'écriture par [?]. L'idée vient du fait que les clusters sont produits par un accès aux fonctions naturelles de la partie droite du cerveau et ses prédilections pour les images et les métaphores, suivi par une collaboration consciente avec la partie gauche du cerveau pour la syntaxe et la logique [Ambron 88]. Cette technique est utilisée dans le domaine éducatif. Les étudiants travaillent en collaboration pour élaborer un cluster map. Elle permet une participation active des étudiants et réduit leur anxiété.

1.8.3 Argument-Map

La théorie de l'argumentation est un domaine de recherche interdisciplinaire très riche. Déjà très répandue dans les domaines de la philosophie, la communication, les linguistiques et la psychologie. Elle trouve aussi son utilité dans un large éventail d'applications théoriques et pratiques de l'IA [Van Emeren et al 96], [Prakken et Vreeswijk 02], [Chesñevar et al 00] et [Carbogim et al 00]. L'Argument Mapping est une technique permettant d'obtenir une image du raisonnement et de l'argumentation. Sous forme d'un arbre ayant

4. MindManager : http://www.mindjet.com/
5. Inspiration : http://www.inspiration.com
6. Axon Idea Processor : http://web.singnet.com.sg/~axon2000/

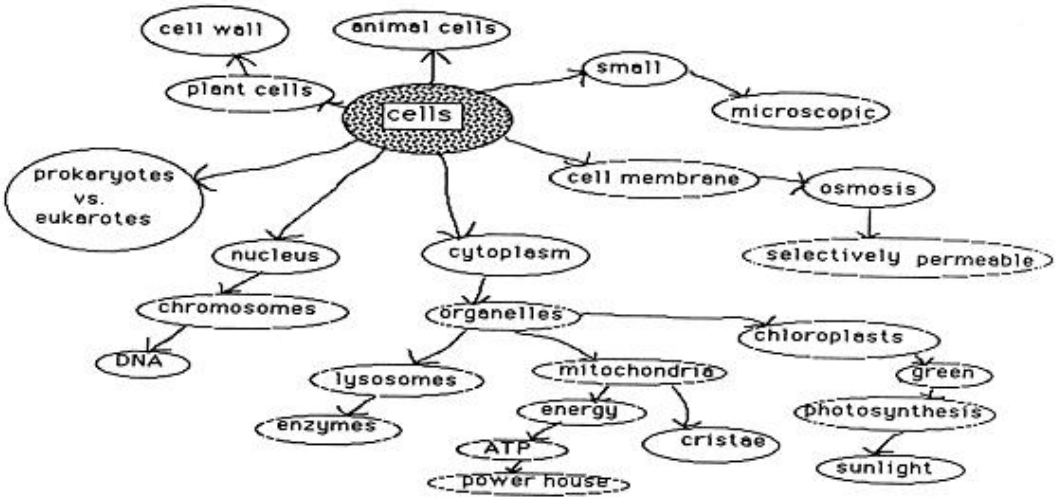


FIGURE 1.7: *Un cluster map du domaine de la biologie [Ambron 88].*

comme racine une conclusion reliée à des arguments en faveur et/ou en dépit. C'est une variante des Concept Maps où les concepts ne sont que des arguments, des déclarations, des propositions et des conclusions.

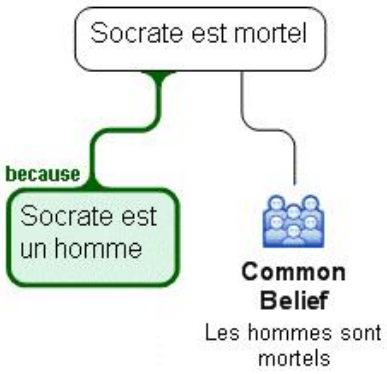


FIGURE 1.8: *Exemple d'Argument Map concernant la mortalité de Socrate.*

Argument-mapping est issu des travaux de Charles Wigmore, qui a conçu des Argument-maps pour des argumentations légales complexes. En 1958, le philosophe Stephen Toulmin a publié son livre *« The Uses of Argument »*, qui décrit un schéma simple pour l'argument-mapping. L'Argument-mapping est devenu un outil d'aide à l'apprentissage collaboratif [Van Gelder 03]. Cette technique est supportée par les outils logiciels de Concept Mapping, mais il existe certains outils dédiés à celle-ci comme Rationale[7], Araucaria[8], iLogos [9], Athena Standard[10],Causality Lab [11] et ArguMed[12].

1.8.4 Topic Map

Topic Map (TM) est un standard ISO pour la représentation et l'échange de connaissances. Il est connu depuis 1999 sous le nom ISO/IEC 13250. Un TM représente l'information utilisant des 'topics' ou 'sujets' (concepts), des associations (relations entre sujets) et des occurrences (représentent des relations entre les sujets et les ressources d'informations de ceux-ci). Similaire aux réseaux sémantiques, concept maps et mind maps

7. Rationale : http://rationale.austhink.com/reasonable
8. Araucaria : http://www.computing.dundee.ac.uk/staff/creed/araucaria/
9. iLogos : http://www.phil.cmu.edu/projects/argument_mapping/
10. Athena Standard : http://www.athenasoft.org/sub/software.htm
11. Causality Lab : http://www.phil.cmu.edu/projects/causality-lab/
12. ArguMed : http://www.ai.rug.nl/~verheij/aaa/index.htm

à l'exception de sa standardisation et son aspect formel. XML Topic Maps (XTM) et le langage d'échange normalisé depuis 2001 [Pepper et Moore 01]. TMAPI (Topic Map Application Programming Interface) est une API standard de facto. Un langage d'interrogation est en cours développement par l'ISO. Il existe aussi une forme linaire des TM (forme textuelle équivalente).

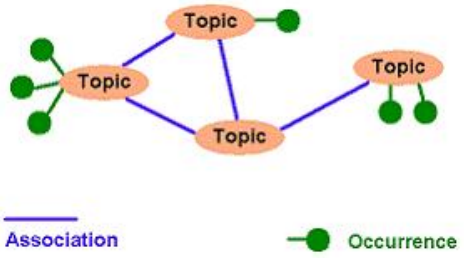


FIGURE 1.9: *Structure générale d'un Topic Map.*

Le concept central des Topic Maps est le Topic. Il représente un sujet unique et clairement identifié dans le contexte et est une instance d'au moins une classe. La propriété caractéristique de ladite classe définit un type du Topic. Un Topic est décrit par son (ses) nom(s), occurrences et rôle(s) dans les associations. Le sujet est ce que le Topic essaie de représenter formellement. Sans restreindre la nature de sa substance, les Topic Maps exigent qu'il soit identifié de façon unique et non ambigüe. L'identification du sujet est problématique. Les sujets ressources adressables (par exemple documents sur le Web, bases de données, etc.) sont identifiées de manière non ambigüe et optimale par leurs URI. L'URI de cette ressource permet l'accès à une définition écrite, sonore ou visuelle dudit sujet. Une occurrence est un lien vers une ressource sur le sujet du Topic. Les occurrences sont classifiables par type : document texte, image, statistiques etc. les occurrences sont valides dans un contexte.

Le modèle des TM est issu d'un travail initialisé en 1993 par le GCA Research Institute. Son origine en faisait un formalisme nouveau à la disposition des documentalistes. De manière intuitive, on peut considérer les TM comme une structure qui regroupe en un seul formalisme des représentations de type index, thésaurus, table des matières et glossaire.

1.8.5 Text Graph

Selon [Nuutila et Törmä 04], il s'agit d'un type particulier de Concept maps qui a été utilisé avec succès dans l'enseignement de l'informatique. L'avantage du Text Graph est la précision de la signification, qui le permet d'être plus approprié pour la représentation d'un contenu technique. Un nœud comporte un fragment de texte pouvant contenir des ancres. Une ancre constitue un support auquel s'accrochent d'autres fragments. Les arêtes permettent de relier les nœuds et les ancres entre eux.

TGE[13] est un outil qui supporte cette technique. Implémenté en Java et utilise Kawa [Bothner 03] pour un traitement sous forme de schémas.

1.9 Conclusion

Dans ce chapitre nous avons exploré les fondements théoriques des concept-maps. Le domaine de la psychopédagogie s'intéresse au processus d'apprentissage, d'enseignement, de représentation de connaissances et de leur mémorisation. Les travaux de recherches ont aboutit à plusieurs techniques de représentation de connaissances qui aide à l'enseignement et l'apprentissage qui sont aussi variées que les études

13. TGE (Text Graph Editor) : http://tge.cs.hut.fi/Software/TGE/

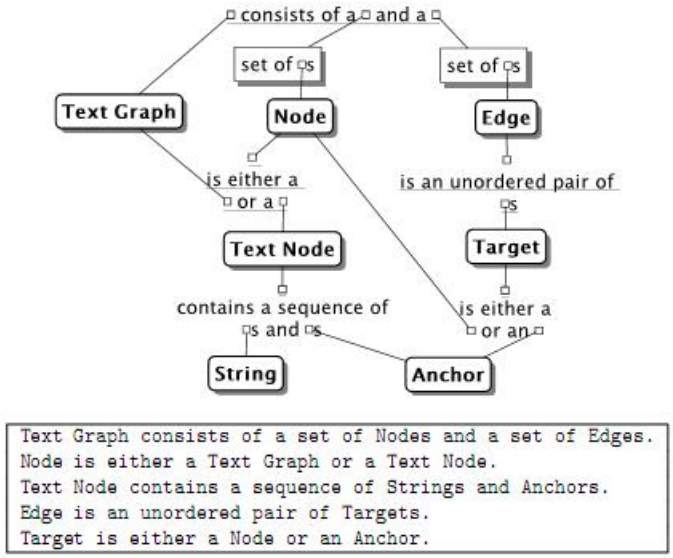


FIGURE 1.10: *Text graph défini en termes d'un Text graph avec une version textuelle*

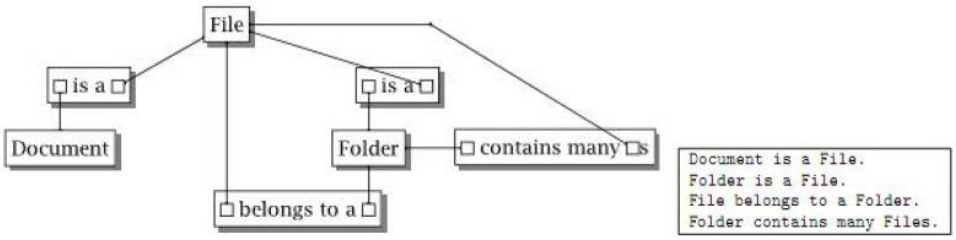


FIGURE 1.11: *Un Text Graph concernant les concepts liés au système de fichiers et l'ensemble des phrases correspondantes.*

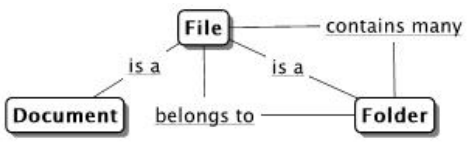


FIGURE 1.12: *Un Concept-map du système de fichiers.*

menées dans ce sens. Du fait qu'un domaine (ou thème) n'est qu'un ensemble de concepts liées entre eux par différentes relations, les connaissances forment alors une carte de concepts. Les concepts représentés sous forme sonore sont bien assimilés, mieux encore pour une forme visuelle graphique.

Plusieurs variantes des concept-maps ont été proposées et chacune peut être mieux adaptées à un domaine particulier. Les différences entre ces formes sont essentiellement la nature des relations qui relient les concepts ainsi que la façon d'organiser les concepts. L'édition des concept-maps ne nécessite pas un matériel spécifique sauf si cette tâche est prévue pour un traitement automatique par ordinateur. Pour pouvoir les exploiter dans le domaine de l'informatique il faut expliciter leur sémantique de façon formelle. De même, les concept-maps font partie des langages visuels ce qui nécessite d'aborder ces aspects sous cet angle.

Chapitre 2

Langages visuels et spécifiques au domaine

Sommaire

2.1 Introduction

Que ce soit en linguistique (langage naturel) ou en informatique (langage de programmation ou de modélisation), il est depuis longtemps établi qu'un langage est caractérisé par sa syntaxe et sa sémantique. La syntaxe décrit les différentes constructions du langage et les règles d'agencement de ces constructions également appelées *context condition*. La sémantique désigne le lien entre un signifiant (un programme, un modèle, …etc.), et un signifié (un objet mathématique) afin de donner un sens à chacune des constructions du langage. Il y a donc entre la sémantique et la syntaxe le même rapport qu'entre le fond et la forme.

Pour les humains, l'utilisation des images était et reste toujours une façon simple d'expression et de communication. Dès sa petite enfance, l'humain apprend la vie par les images et durant toute sa vie il pense en formant des images dans son esprit. Les recherches sur les langages visuels considèrent cette perception visuelle humaine pour concevoir des systèmes et des langages de programmation/modélisation qui utilisent des images ou des symboles graphiques. Ces recherches sont favorisées par le progrès technologique des écrans graphiques.

Les unités structurelles d'un langage visuel sont caractérisées par les lignes, la forme, la couleur, la texture, le motif, la direction, l'orientation, l'échelle, la proportion, le mouvement, la position relative et absolue, …etc.

Plusieurs chercheurs ont empiriquement établi l'avantage cognitif des méthodologies graphiques par rapport aux méthodologies textuelles. Le domaine des langages visuels regroupe des recherches sur les systèmes graphiques, les langages de programmation/modélisation et les interactions homme-machine. Les études sont faites sur les représentations graphiques, sur les paradigmes, sur les caractéristiques du langage et sur son utilisation. Ces différents points permettent, en effet, de classifier les langages visuels.

Lorsqu'on s'intéresse à un domaine d'application bien précis, un langage à usage général (General Purpose Language : GPL) n'offre pas le niveau d'expressivité et la souplesse d'utilisation nécessaires contrairement à un langage spécifique au domaine (Domain Specific Language : DSL). En plus, si le DSL est orienté modélisation

on parle plutôt de DSML[1] (Domain Specific Modeling Language), qui n'est pas forcément visuel de la même manière qu'un langage visuel n'est pas forcément spécifique à un domaine particulier.

Nous allons nous intéresser dans ce chapitre à un ensemble de langages visuels spécifiques aux domaines. Nous étudieront leurs fondements, leurs classifications et surtout les techniques de leur spécification formelle.

2.2 Les graphes

Un graphe permet de représenter la structure, les connexions d'un ensemble complexe en exprimant les relations entre ses éléments : réseau de communication, réseaux routiers, interaction de diverses espèces animales, circuits électriques, …etc. Les graphes constituent donc une méthode de pensée qui permet de modéliser une grande variété de problèmes en se ramenant à l'étude de sommets et d'arcs.

Les derniers travaux en théorie des graphes, du fait de l'importance qu'y revêt l'aspect algorithmique, sont souvent effectués par des informaticiens. Effectivement, il s'agit essentiellement de modéliser des problèmes. Nous exprimons le problème en termes de graphes et ensuite il devient un problème de la théorie des graphes. Les solutions de problèmes de graphes peuvent être faciles et efficaces (le temps nécessaires pour les traiter informatiquement est raisonnable car il dépend polynomialement du nombre de sommets du graphe) ou difficiles (car le temps de traitement est exponentiel) dans les cas où nous utilisons une heuristique, c'est-à-dire un processus de recherche d'une solution. La manipulation des langages visuels se trouve facilitée en recourant à la théorie des graphes. Les programmes et les modèles visuels représentés sous forme de graphes (et d'hypergraphes) se prêtent bien à des transformations (de graphes) permettant ainsi de réaliser des translations, de compilations, de génération de codes ou de changement de formalisme. Il est même possible actuellement de spécifier formellement les langages visuels en s'appuyant sur des structures de graphes.

2.3 Les langages visuels

Un langage visuel constitue toute forme de communication basée sur des notations graphiques 2D ou 3D contrairement à celle linéaire basé sur du texte[2]. Les langages textuels purs sont parfois soutenus par des propriétés visuelles comme l'indentation, la coloration, l'accentuation, la mise en gras …etc.

Le domaine des langages visuels est essentiellement le fruit de trois disciplines : l'infographie, la théorie des langages et l'IHM. PYGMALION [Smith 75] est un langage visuel de programmation, qualifié d'iconique (basée sur des icônes) qui constitue un déclic pour de nombreuses recherches dans le domaine de langages visuels de programmation.

Bien que les langages visuels exploitent pleinement les images, l'utilisation du texte est restreinte. Le rôle du texte est souvent limité aux étiquettes et annotations.

Comme tout autre langage, un langage visuel ne peut être interprété correctement que s'il possède une syntaxe et une sémantique formelles. La plupart des langages textuels (de programmation) sont intelligible grâce aux efforts prodigués pour investiguer leur sémantique formelle. Leurs homologues visuels n'ont pas eu cette chance et leur sémantique demeure cachée derrière leurs formes graphiques.

2.4 Classification des langages visuels

Nous somme tous convaincus qu'une image vaut mille mots. Beaucoup de recherches ont montré que les langages visuels peuvent être plus compréhensibles que leurs équivalents textuels. Il existe une multitude de langages visuels qu'on peut classifier selon plusieurs critères. Une des fameuses taxonomies dans le domaine de la programmation visuelle et la visualisation des programmes est certainement celle de Myers [Myers 90].

1. Parfois les DSLs sont dits langages dédiés et les DSMLs sont dits langages de modélisation dédiés.
2. Les langages textuels sont considérés comme monodimentionnels 1D et les compilateurs ou interpréteurs les traitent en flots unidimensionnels

2.4.1 Selon les systèmes de programmation

On identifie selon le système de programmation les catégories suivantes :

1. Programmation visuelle/programmation textuelle : Ici on distingue les langages de programmation purement visuelle de ceux de la programmation classique purement textuelle. La distinction n'est pas totale, car d'après [Erwig et Meyer 95] il existe des langages hétérogènes.

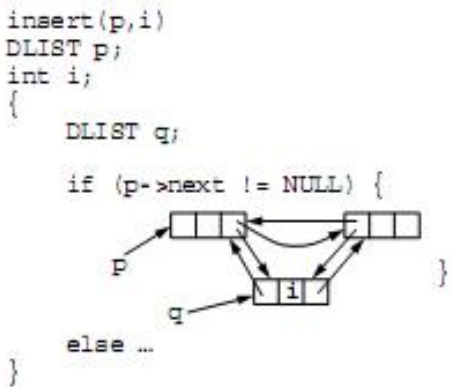


FIGURE 2.1: *Programme C hétérogène pour l'insertion dans une liste [Erwig et Meyer 95].*

2. Programmation par l'exemple/Programmation par spécification explicite : ici la distinction est faite entre les langages qui infèrent les programmes à partir d'un ou de plusieurs exemples de ceux nécessitant une spécification explicite des programmes.
3. Langage interprété/Langage compilé : pour distinguer les langages directement exécutables de ceux nécessitant une translation intermédiaire avant l'exécution.

2.4.2 Selon la technique de spécification

La deuxième taxonomie de Myers est plus intéressante. Elle permet de classifier les systèmes selon la technique de spécification en 14 catégories. Six (06) catégories appartenant aux Concept-maps ont été identifié dans [Kremer 97].

La dernière taxonomie de Myers concerne les langages de visualisation de programmes selon qu'ils représentent les données, le code ou les algorithmes.

2.4.3 Selon le domaine d'application

Si on considère le domaine d'application, nous pouvons distinguer des langages visuels pour la programmation [Boshernitsan et Downes 97], pour la représentation de connaissances [Gaines 91], pour la prise de décision [Conklin et Begeman 87], pour la visualisation de programmes [Myers 90], pour l'interrogation des bases de données et des SIGs [Calcinelli et Mainguenaud 94] …etc. Ils sont aussi variés que les domaines d'application.

2.4.4 Selon les relations spatiales

Une classification des « maps sémantiques multi-relationnelles », comportant 04 catégories, présentée dans [Lambiotte et al 84] :

1. Systèmes d'agencement et de localisation spatiale : basés sur la notion de proximité, de relations spatiales (comme en-dessous, en-dessus) et de lignes de frontières pour transférer les relations spatiales entre les éléments. Connus aussi sous le nom de *systèmes iconiques*.

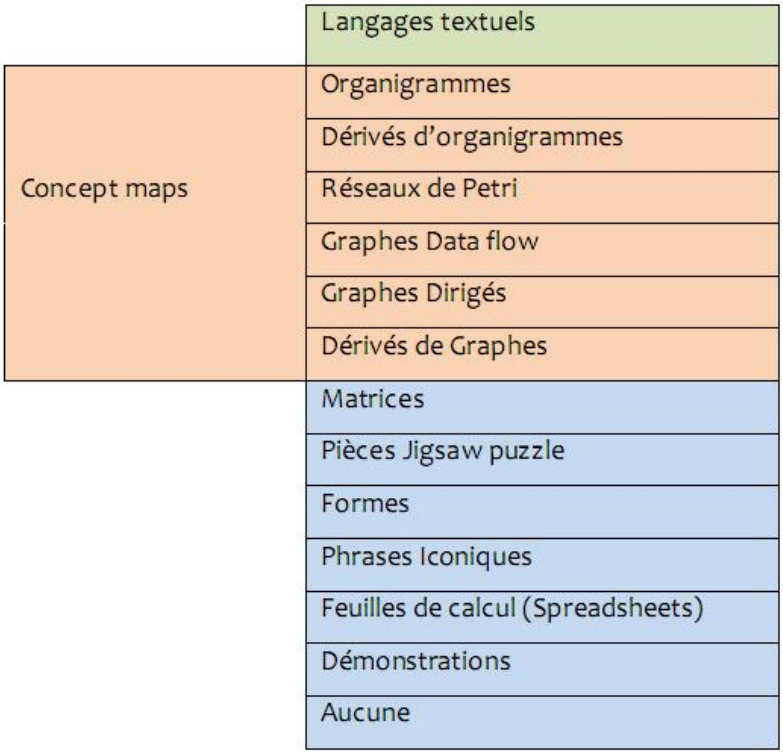


FIGURE 2.2: *Concept maps dans la classification de Myers [Kremer 97]*

2. Systèmes de nœuds : basés essentiellement sur les attributs de nœuds comme le style, la forme et la couleur pour pouvoir distinguer les différents types de nœuds. S'ils font usage des arcs, leurs types peuvent être inférés des types de nœuds reliés.
3. Systèmes de liens : basés sur un transfert d'information à travers les arcs qui peuvent être libellés et/ou possédant des attributs pour la distinction tel que le style, l'épaisseur et la couleur. Plusieurs types d'arcs sont ainsi identifiables. S'ils font usage de nœuds, ces derniers auront la même apparence et contiennent des données.
4. Systèmes Hybrides : combinent deux techniques ou plus et permettent ainsi de transférer un ensemble plus riche d'informations sémantiques.

Il est à noter que la première catégorie n'est pas considérée comme Concept map. Les autres catégories sont typiquement des concept maps et incluent les systèmes dits *diagrammatiques*.

Un *langage diagrammatique* utilise des composants graphiques (images, graphiques, ..etc) qui peuvent être reliés entre eux. Il est possible de rencontrer des langages hybrides qui utilisent à la fois une représentation iconique et diagrammatique.

2.5 Les langages spécifiques au domaine

Les langages spécifiques au domaines DSLs sont des langages qui se focalisent sur un domaine particulier[3]. Dans la littérature, ils sont désignés aussi par les termes *orientés-application*, *spécialisés*, *orientés-problème*, *spécifiques à la tâche* ou *langages d'application*[4]. Il faut noter ici la distinction entre les DSLs et les langages adaptés au domaine (Domain Adapted Languages : DALs). Un DSL n'est utilisable (ou peu utilisable) en dehors de

3. Les DSLs sont dans la plupart des cas déclaratifs du fait que la solution est exprimée dans un haut niveau d'abstraction.
4. En anglais les termes correspondants sont respectivement *application-oriented*, *special purpose/specialized*, *problem-oriented*, *task-specific* et *application languages*.

son domaine d'application, tandis qu'un DAL est à l'origine un GPL augmenté d'une bibliothèque spécifique à un domaine donné [Backhouse 00]. Dans [Taha 09], la définition d'un DSL doit s'appuyer sur quatre points essentiels :

1. Un domaine bien défini et central : la définition du domaine peut être mathématique comme l'algèbre linéaire ou les équations différentielles, comme elle peut être rattachée à une activité humaine comme les assurances.
2. Une notation claire : les éléments (textuels ou graphique de bases échangés sont clairement définies. Une syntaxe bien claire, tiré du jargon des experts du domaine facile à coder et à traiter.
3. Un sens informel concis : la signification de chaque élément du langage doit être claire et unique dans le domaine considéré. Par exemple une note de musique possède une seule signification à travers le monde pour tous les musiciens.
4. Un sens formel clair et implémentable : chaque élément du langage possède un sens formel issu d'un mapping du sens du domaine vers un sens concret qui aide à l'implémentation. par exemple une note de musique doit être vu dans ce cas comme un triplet de fréquence, volume et durée.

Les DSLs ne sont pas nouveaux dans le monde informatique. le tableau 2.1 illustre quelques uns :

DSL	Domaine d'application
BNF	Spécification de la syntaxe
Langage Macro d'Excel	Tableur
HTML	Pages web
LaTeX	Traitement de texte
SQL	Interrogation de Bases de données
VHDL	Conception Hardware

Table 2.1: *Langages spécifiques au domaine les plus utilisés. Tiré de [Mernik et al 03].*

La modélisation spécifique au domaine (Domain Specific Modeling : DSM) est une approche de développement logiciel qui s'intéresse à exprimer la solution d'un problème spécifique à un domaine donné, dans un niveau d'abstraction plus élevé que la programmation directe, en utilisant les concepts et les règles propre au domaine du problème. La solution est exprimée dans un langage dédié au domaine du problème et noté DSML (Domain Specific Modeling Language). En plus du DSML défini, il y aura toujours un besoin de générer automatiquement un code exécutable dans un langage de programmation général pour le modèle conçu, du fait que concepts utilisés sont dans un haut niveau d'abstraction. Le DSML obtenu facilite la tâche de ses utilisateurs du fait qu'il utilise les concepts de leur domaine et permet ainsi de réduire la complexité du développement, favoriser la productivité, augmenter la fiabilité, faciliter la maintenabilité et la portabilité. Néanmoins, le développement des DSLs nécessite à la fois une expertise dans le domaine d'application visé et des connaissances dans le domaine du développement des langages [Kelly et Tolvanen 08].

Il est à signaler que le mot spécifique est un peu subjectif à notre avis, parce que les domaines d'application s'interfèrent voire même s'imbriquent les uns dans les autres !. Nous pouvons parler ici du niveau ou degré de spécificité du DSL.

Enfin, la terminologie et les concepts utilisés dans le domaine des DSLs sont illustrés par le concept-map de la figure 2.3.

2.5.1 Le développement des DSLs

Plusieurs méthodologies de développement on été proposées notamment dans les travaux de [Hudak 98], [Spinellis 01], [Kelly et Tolvanen 08] et [Cook et al 07]. Une étude mené dans [Mernik et al 03] a identifié clairement et en détail le processus de développement des DSLs indépendamment des outils et des langages utilisés. Nous pouvons résumer les différentes phase de développement en :

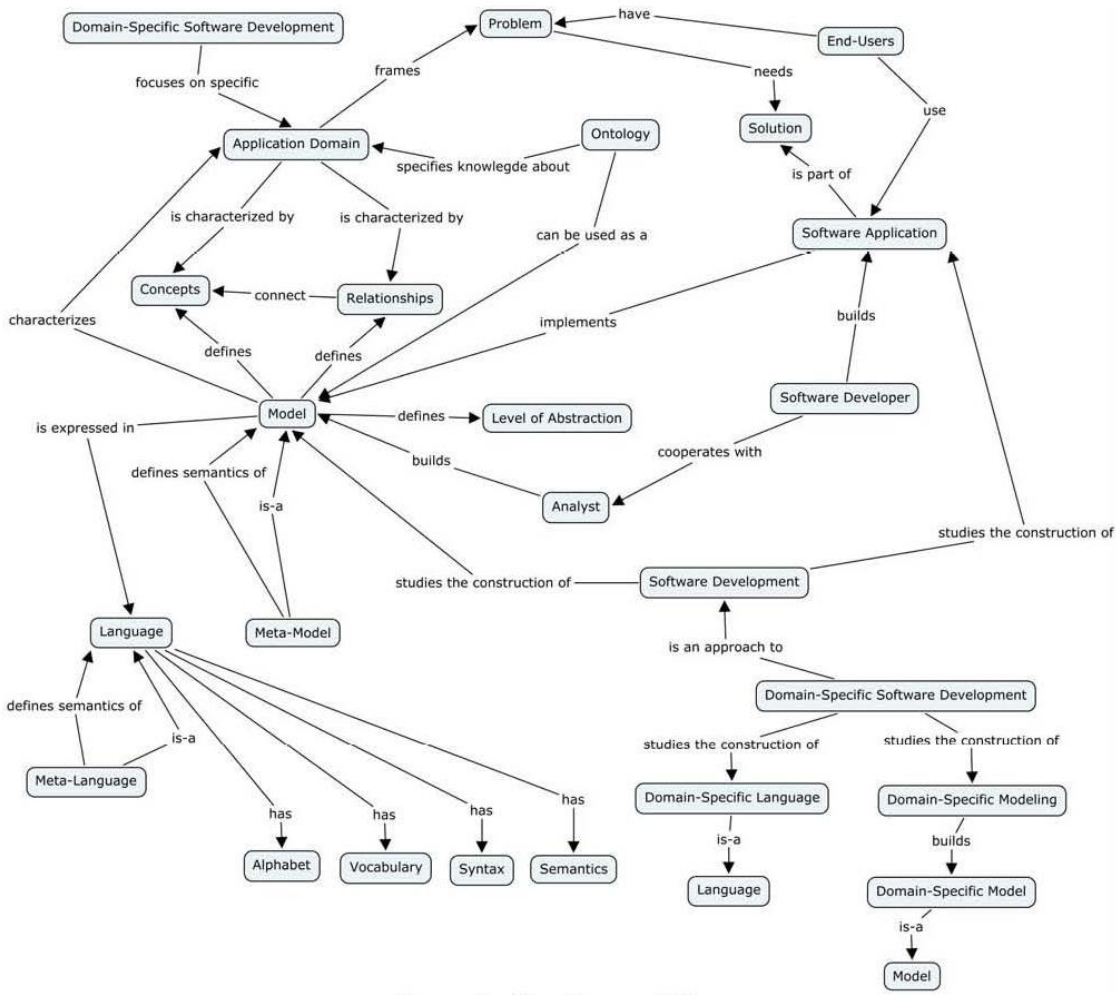


FIGURE 2.3: *Concept-map des concepts du domine des DSLs [Sanchez-Ruiz et al 07].*

1. **Décision** : l'adoption d'un DSL existant est certes la meilleure solution et la décision d'en créer un nouveau doit être justifiée. Le développent d'un nouveau DSL aura un impact économique à considérer (outils utilisés, temps dépensé, effort de développent, apprentissage, ...etc.).
2. **Analyse** : Dans cette phase, le domaine est identifié clairement et les concepts de bases sont collectés. Cela nécessite la coopération des experts du domaine et des informaticiens. L'analyse du domaine se fait souvent de façon informelle, mais des méthodologies claires ont été proposées et peuvent servir efficacement comme DARE [Frakes et al 98], DSSA [Taylor et al 95] et FODA [Kang et al 90]. L'ingénierie des connaissance se trouve au cœur de cette phase avec la capture des connaissances, leur représentation ainsi que le développement des ontologies.
3. **Conception** : deux éléments essentiels dans cette phase sont à considérer : la relation du nouveau DSL avec un langage existant et le degré de formalité de la description de la conception.

   Pour le premier, nous pouvons nous inspirer d'un langage existant pour dégager les points de ressemblances ce qui facilitera la tâche des futurs utilisateurs s'il connaissent déjà le langage source d'inspiration. L'alternative d'un DAL est toujours vérifiée si le langage visé est extensible dans ce sens par le biais d'une bibliothèque spécifique au domaine. A l'extrémité, nous trouvons les DSLs qui n'ont pratiquement aucun lien avec les langages existants, difficiles à développer.

   Le second élément est la spécification de la conception. Une fois l'image du DSL est claire, celle-ci peut être faite de façon informelle utilisant le langage naturel par exemple, facile à produire mais le plus souvent imprécise, ou bien de façon formelle, utilisant les techniques connues comme les expressions régulières et les grammaires formelles pour spécifier la syntaxe et les systèmes de réécriture, les machines à états finis et les grammaires attribuées pour spécifier la sémantique …etc[5].
4. **Implémentation** : la mise en œuvre de la conception est subordonnée aux choix retenus quant à la nature d'exécution du DSL. Une liste non exhaustive des alternatives comprend l'interprétation, la compilation, le pré-processeur, l'embarcation et l'extension (sous forme de bibliothèque de modifications faites au compilateur/interpréteur du langage hôte ...etc.).
5. **Déploiement**. l'effort consacré au déploiement du DSL et de son acceptation est facilité par le succès des phases précédentes. En plus, la facilité d'utilisation, l'adaptation au domaine visé, l'efficacité, l'expressivité, la facilité d'accès au DSL, l'implication d'une large communauté pour les tests et l'impact sur la productivité sont des points clés à prendre en compte.

2.5.2 Outils de développement des DSLs

Plusieurs outils sont proposés pour le développement des DSLs. Ils se partagent tous la même idée de produire un DSL à partir d'une description du langage. Le produit obtenu varie selon l'outil utilisé d'un simple interpréteur à un environnement de développement intégré. En plus, certains outils possèdent une méthodologie de développement fixe tandis que d'autres semble avoir un caractère indépendant d'une méthodologie bien claire. Quant à la prise en compte des phases cités avant, la majorité des outils n'offrent plus de support à la décision, n'intègrent pas l'analyse, supportent mal la conception et s'offrent pleinement à l'implémentation [Mernik et al 03].

Le développement d'un DSL nécessite au moins un compilateur, un interpréteur ou un simulateur. L'objectif idéal est d'obtenir un environnement complet permettant l'édition, compilation/interprétation/translation/simulation, débogage, …etc. Plusieurs recherches se sont concentrés durant les années 70 et 80 sur la génération automatique d'environnements de développement spécifiques à certains langages de programmation. Les résultats ont permis d'obtenir un ensemble plus ou moins complet d'outils (éditeur, compilateur/interpréteur, débogueur, ….) comme CPS [Teitelbaum et Reps 81] et Centaur [Klint 93]. Ils s'agissait des langages textuels que leurs syntaxes ont été spécifiées utilisant des métalangages (meta-syntactic formalism pour CPS) offrant des éditeurs qualifiés de *dirigés par la syntaxe* (Syntax Directed Edition). Ainsi l'arbre syntaxique est crée à fur et à mesure sans avoir le besoin de faire le balayage du programme (parsing). Ce style d'édition restreint

5. En respect à la nature du DSL qui n'est pas forcément un langage de programmation !

l'ensemble des modèles valides. Actuellement, les éditeurs offrent la possibilité de créer des modèles erronés et qualifiés *de main libre* (Hand-Free Edition). Ainsi, la validation du modèle est une tâche à part entière.

2.5.3 Architecture d'une DSM

Pour augmenter la productivité et la qualité ainsi que réduire et cacher la complexité, l'architecture d'une DSM est composée de trois couches au dessus de l'environnement cible [Kelly et Tolvanen 08]. La figure 2.4 illustre cette architecture.

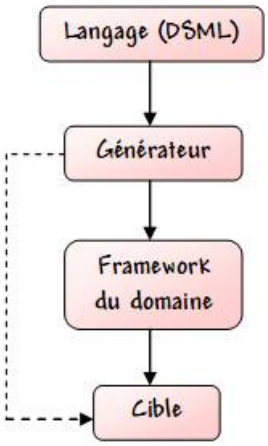


FIGURE 2.4: *Architecture d'une DSM inspirée de [Kelly et Tolvanen 08].*

La Modélisation spécifique au domaine cache la complexité en offrant un DSL (dans ce cas c'est un DSML) contenant tous les concepts et les règles du domaine, et non ceux d'un langage de programmation. L'utilisateur travaille directement avec les concepts du domaine. Ainsi, ces derniers sont projetés directement en objets du DSML ou en propriétés, relations, sous modèles d'objets, ….

Chaque composant du modèle peut produire un fragment de code. En plus chacune de ses propriétés, ses liens avec les autres peut également produire du code. Le rôle du générateur ici est de spécifier comment balayer le modèle, extraire les informations et produire le code correspondant. Le point fort de la DSM est que l'utilisateur n'a pas à modifier ni compléter le code produit. Le générateur est invisible aux utilisateur et joue un rôle semblable aux compilateurs. Comme tout autre langage, un DSML est spécifié par une syntaxe et une sémantique. La syntaxe sous sa forme abstraite parvient directement des concepts liés au domaine sans se soucier de l'implémentation. Elle est spécifiée souvent sous forme d'un *métamodèle* conforme à un *méta-métamodèle*[6]. La composition de plusieurs métamodèles spécifiques à d'autres DSL favorise la réutilisabilité. Les règles du domaine sont codées dans le métamodèle ce qui met des contraintes sur les modèles construits et favorise l'évitement prématuré des erreurs. La syntaxe concrète s'occupe des symboles utilisés dans la notation et leur représentation. Les DSMLs sont souvent visuels et combine parfois le graphisme et le texte. Néanmoins, il existe des DSMLs qui utilisent les matrices, les tableaux, les formulaires, voire même purement textuels. Il y'a tendance dans la communauté DSM à favoriser une notation proche de celle utilisée par les experts du domaine pour réduire encore l'écart cognitif. Des règles de correspondance entre la syntaxe abstraite et celle concrète permettent d'aller dans les deux directions. Le style aussi est lié à la syntaxe concrète (impératif ou déclaratif par exemple).

Pour la sémantique, il existe plusieurs façons de la spécifier. D'après [Kelly et Tolvanen 08], la sémantique qui compte le plus est celle du domaine où chaque élément du DSML possède un sens bien clair selon le domaine visé. Quant à la sémantique statique au sens formel, elle peut être exprimée dans le métamodèle. La sémantique dynamique est exprimée par translation du fait qu'il y aura génération de code dans un GPL.

6. Une grammaire sans contexte par exemple peut aussi servir.

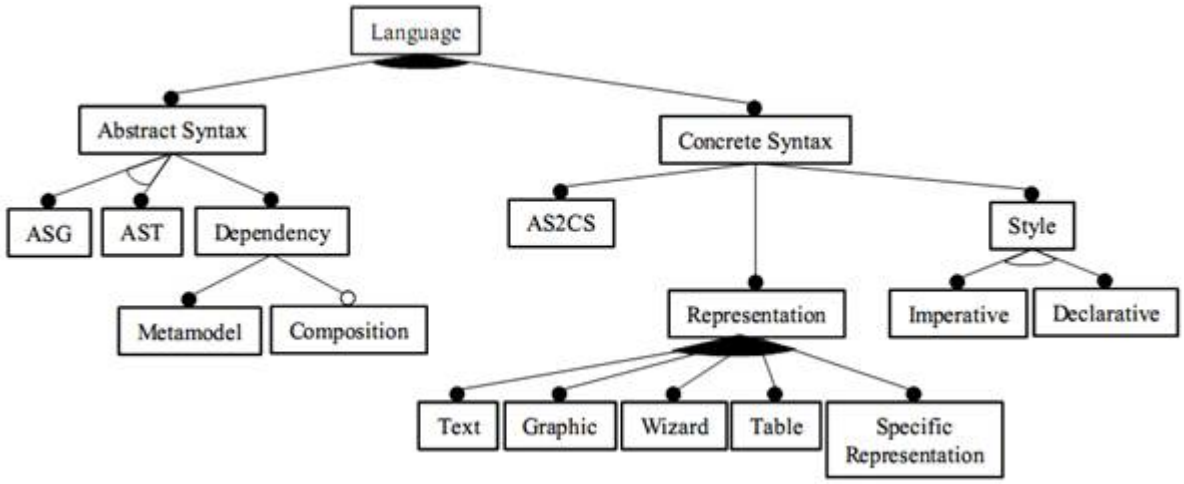


Figure 2.5: *Aspect syntaxique d'un DSML selon la notation FODA [Langlois et al 07]*

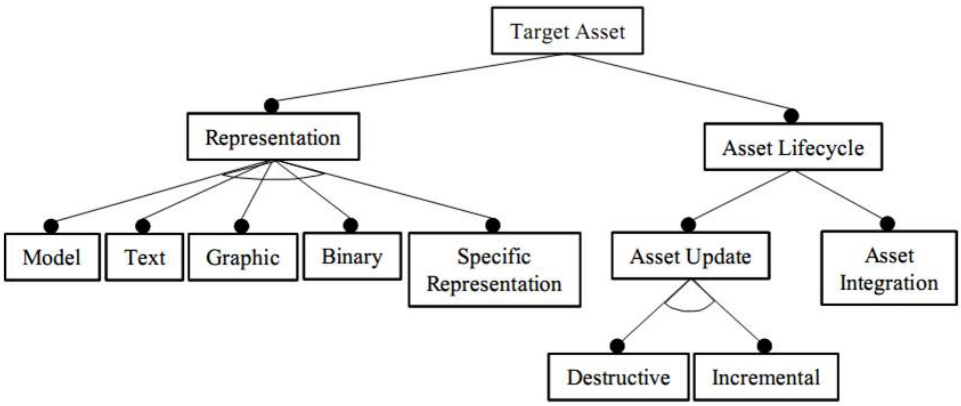


Figure 2.6: *Types de résultats de la transformation [Langlois et al 07].*

Le framework du domaine représente une interface entre le code généré et la plateforme cible. Bien qu'on peut parfois se passer de ce composant si la plateforme fournit suffisamment de services. Il fournit du code et des composants qui facilite le processus de génération de code.

La plateforme n'est pas forcément un ordinateur conventionnel et peut s'agir d'un système embarqué ou autre. La plateforme cible varie ainsi selon le domaine. Le produit issu de la transformation ou la génération doit avoir une présentation (texte, graphique, binaire …etc.) et possède un cycle de vie puisqu'il sera crée, mis à jours intégré à d'autres résultats pour être déployé et voire même détruit, voir la figure 2.6.

Dans la DSM, le langage est exprimé sous forme de métamodèle et les applications sont des instance de celui-ci. Ainsi les modèles ne peuvent exprimer que ce que permet le métamodèle.

2.6 Conclusion

Les langages visuels en informatique se présentent en multi-dimension mais sont aussi variés que leurs objectifs (programmation/modélisation/visualisation de programme …etc.), domaines d'application, techniques de spécification, types de relations spatiales et méthodologie de développement.

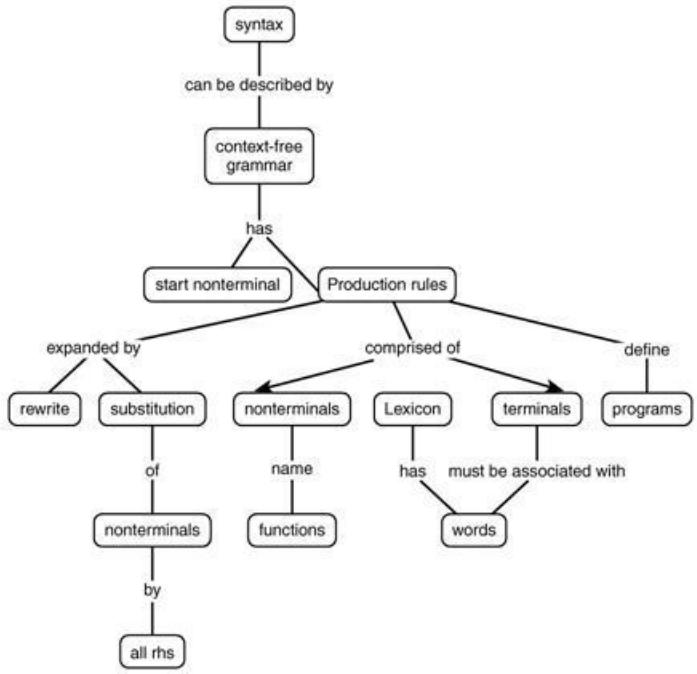
Les concept-maps forment une famille des langages visuels qui englobe les organigrammes et leurs dérivés, les graphes (dirigés ou non) et leurs dérivés ainsi que les variantes des réseaux de Petri. Autrement dit, ce sont des langages diagrammatiques où l'agencement spatial n'a pas d'intérêt, mais les nœuds et les arcs peuvent posséder des attributs pour véhiculer des informations. Les informations peuvent aussi être transférées entre les nœuds par les arcs.

La majorité des DSML son visuels et sont dédiés à la modélisation des problèmes d'un domaine donné. Ils facilitent la tâche de leurs utilisateur en permettant de spécifier les solutions dans un niveau d'abstraction élevé. La génération du code est automatisée de façon transparente à l'utilisateur qui permet ainsi de gagner en productivité. Dans notre étude, il s'agit de proposer un formalisme spécifique au domaine de la modélisation des systèmes à évènements discrets, en plus de son aspect visuel. Sa conception et son architecture doit s'adhérer aux DSMLs.

Chapitre 3

Ingénierie dirigée par les modèles et spécification formelle des DSMLs visuels

Sommaire



3.1 Introduction

La spécification formelle des langages visuels, en particulier celle de la syntaxe, est influencée par les travaux de la théorie des langages formels. L'exploration des résultats obtenus dans le domaine des langages textuels (mono dimensionnel) s'avère nécessaire afin juger la possibilité de généralisation pour les langages visuels (multidimensionnels). De nouvelles formes de grammaires sont utilisées actuellement pour prendre en considération l'aspect multidimensionnel, mais des techniques récentes proposent plutôt des approches logiques [Helm et Marriott 90] et algébriques [Uskudarli et Dinesh 95] et [Bardohl et Taentzer 97]. La plus efficace demeure celle basée sur l'ingénierie dirigée par les modèles grâce à sa productivité. En plus, les DSMLs sont au cœur de cette dernière approche.

3.2 Ingénierie dirigée par les modèles

3.2.1 Principes

L'ingénierie dirigée par les modèles (IDM)[1] est une approche de développement logiciel dans laquelle le modèle est le point autour duquel le processus de développement est centré. Elle part d'un haut niveau d'abstraction pour que dans un premier temps produire automatiquement ou semi-automatiquement des outils de création de modèle pour ensuite permettre de les manipuler en s'appuyant sur les transformations successives. Elle adopte le principe que *tout est modèle* en analogie avec la vision de l'orientation objet qui considère que *tout est objet*. Cette approche prend tout son sens dans le cadre des architectures logicielles dirigées par les modèles utilisant des standards tels que l'initiative MDA (Model-Driven Architecture) proposé par l'OMG [Soley et al 00], Usines à Logiciels (Software Factories) proposé par Microsoft [Greenfield et Short 04], MIC (Model Integrated Computing) proposé dans [Balasubramanian et al 04] ou bien d'autres standards, voir le tableau 3.1 et la figure 3.1. Cette approche, très axée sur l'instrumentation des modèles, permet de passer d'un mode *contemplatif*, où les modèles ne servent qu'à de la documentation à un mode *productif* où les modèles permettent d'être exploités en vue d'analyses et générations de code. Il s'agit d'une forme d'ingénierie *générative* dans laquelle tout ou partie d'une application est engendrée à partir de modèles. Les technologies de l'IDM regroupent les DSLs, les machines de transformation et les générateurs.

IDM			
MDA OMG	Software Factories Microsoft	MIC	Autres
Eclipse EMF : Eclipse Modeling Framework GEF : Graphical Editing Framework GMF : Graphical Modeling framework	DSL Tools	GME Generic Modeling Environment	Autres

TABLE 3.1: *IDM : Sdandards et outils.*

L'IDM ne se limite pas à un jeu de standards spécifiques à un organisme particulier, et désire proposer autour de la notion de modèle et de métamodèle une vision unificatrice permettant de rassembler selon [Caron 07] :

1. Différents courants de modélisation, méthodes de modélisation, langages de modélisation dédiés, techniques de génération de code (via assistant, ou fichier de configuration …etc.).
2. Différentes communautés informatiques : Génie logiciel, Web Sémantique, Ingénierie des système interactifs, Ingénierie des systèmes d'information …etc.
3. Différentes disciplines plus anciennes comme la sémiotique et les linguistiques pouvant apporter un plus à l'ingénierie dirigée par les modèles, des recherches abouties mettant en œuvre des solutions

1. En anglais MDE : Model Driven Engeneering

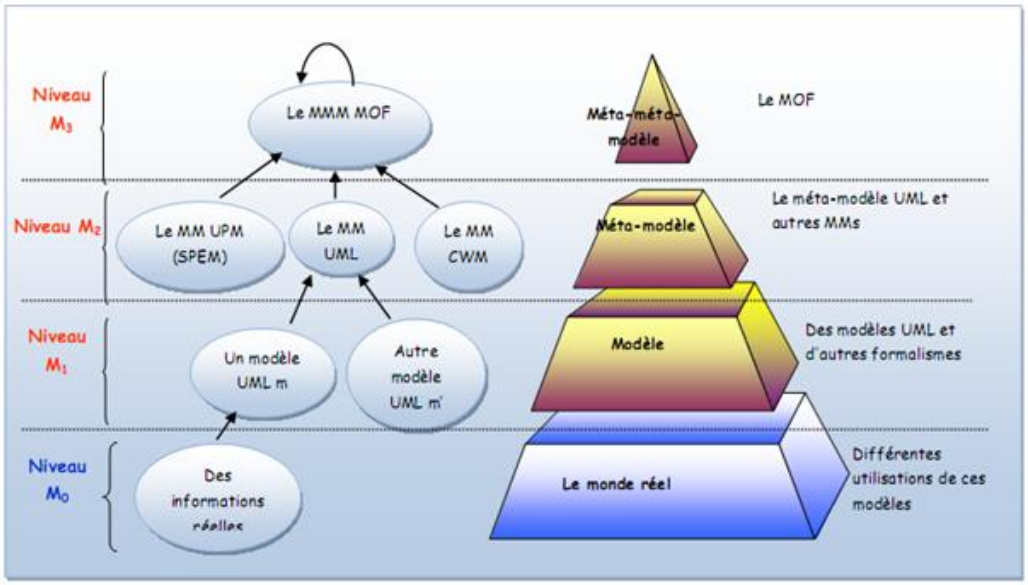


FIGURE 3.1: *Niveaux d'abstraction selon l'OMG.*

différentes de celles que l'IDM propose.

3.2.2 Modèle, métamodèle et mégamodèle

Pour un observateur A, M est un modèle de l'objet O si M aide A à répondre aux questions qu'il se pose sur O [Minsky 68]. Ainsi, un modèle est une abstraction et une simplification d'un système qu'il représente qui est suffisante pour comprendre le système modélisé et répondre aux questions que l'on se pose sur lui.

Le modèle est exprimé dans un langage ou formalisme. Le métamodèle représente (modélise) les entités d'un domaine, leurs relations, leurs contraintes, ainsi que leurs comportements. Autrement dit, le métamodèle n'est pas un modèle du modèle, mais une spécification de la syntaxe abstraite du *langage de modélisation* (formalisme) dans lequel est décrit le modèle.

Un système peut être décrit par différents modèles liés les uns aux autres. Ces modèles sont généralement exprimés dans différents langages (DSMLs) ayant des relations entre eux, restreignant ainsi l'ensemble des combinaisons valides des modèles conformes à ces différents métamodèles. Il est ainsi récemment apparu la nécessité de représenter ces différents DSMLs et les relations entre eux. Une réponse logique à ce besoin a été de proposer des modèles dont les éléments de base sont d'une part les différents DSML et d'autre part les liens exprimant leurs dépendances. Ce type de modèles est appelé *mégamodèle*.

Un mégamodèle est un modèle dont les éléments représentent des métamodèles ou d'autres artéfacts (comme des DSMLs, des outils, des services, etc.). Notons par ailleurs que le concept de DSML a donné lieu à la création de nombreux langages qu'il est maintenant urgent de maitriser (documentation, hiérarchie, etc.) et de pouvoir manipuler facilement (combiner, transformer, etc.). Pour cela, certains zoos[2] proposent un recensement, une documentation et une classification de ces DSMLs et offrent certaines manipulations, comme de pouvoir les transformer vers différents espaces techniques.

Le métamodèle à son tour est exprimé dans un langage (de métamodélisation) spécifié par le méta-métamodèle. Le langage utilisé au niveau du méta-métamodèle doit être suffisamment puissant pour spécifier sa propre syntaxe abstraite et ce niveau d'abstraction demeure largement suffisant (méta-circulaire). Chaque élément du modèle est une *instance* d'un élément du métamodèle (d'un méta-élément).

Un modèle est dit *conforme à* un métamodèle et constitue une *représentation* d'un système existant ou imagi-

2. Un zoo est une vue d'un mégamodèle où tous les métamodèles qui le compose ont le même méta-métamodèle. Par exemple : http ://www.zooomm.org/ et The ATLAS MegaModel Management (AM3) Project http ://www.eclipse.org/gmt/am3 .

naire. La relation entre un méta-métamodèle et un métamodèle est analogue à la relation entre un métamodèle et un modèle. Cette relation est illustrée dans la figure 3.2.

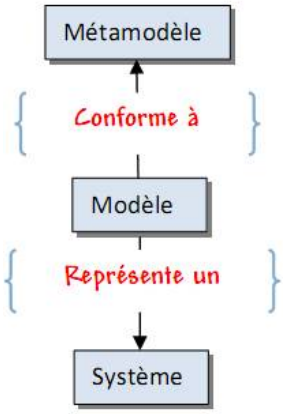


FIGURE 3.2: *Relation entre système, modèle et métamodèle*

L'IDM repose sur une architecture à plusieurs niveaux d'abstraction. Comme proposé par MDA, il existe quatre niveaux (voir le tableau 3.2).

Niveau d'abstraction	Dénomination	Exemples
Méta-métamodèle	M3	MOF, eCore, E-A, EBNF, Graphe conceptuel...
métamodèle	M2	Schéma-XML, RdP, UML, grammaire de Java ...
Modèle	M1	Fichier XML, Programme en Java, ...
Monde réel	M0	Entreprise, Engin, Exécution d'un programme...

TABLE 3.2: *Architecture à quatre niveaux d'abstraction.*

L'IDM a permis de mettre en évidence l'importance du concept de métamodèle pour la définition des langages de modélisation de systèmes mais aussi pour la capitalisation métier dans les activités industrielles. En effet, l'identification et la formalisation des concepts utilisés dans un domaine métier peuvent grandement tirer profit de l'exploitation de métamodèles. Le développement de métamodèles est une activité fondamentale qui garantit la conformité et la réutilisabilité des composants développés. Actuellement, les langages de méta-modélisation orientés-objets comme MOF (Meta-Objet Facilities) avec ses variantes EMOF et CMOF de l'OMG [Soley et al 00] ainsi que ECore d'IBM [Budinsky et al 03] gagnent de plus en plus du terrain.

3.2.3 Transformation de modèles

De la même manière qu'il existe un recueil de modèles ayant diverses propriétés, différentes opérations sur ces modèles peuvent avoir lieu. Toutefois, il n'y a pas de consensus sur une liste définitive de ces opérations. Les tableaux 3.3 et 3.4 présentent une classification des principales opérations.

Les opérations qui s'appliquent sur un seul modèle sont appelées *monadiques*. Par analogie, les opérations portant sur deux modèles sont dites *dyadiques*. Les opérations s'appliquant sur plus de deux modèles sont rares. Les opérations se servent explicitement ou implicitement d'un métamodèle.

Une transformation est une opération qui prend un modèle source en entrée et fournit un modèle cible en sortie. Elle est qualifiée d'*endogène* si les modèles source et cible sont conformes au même métamodèle (source et cible sont dans le même espace technologique), sinon elle est dite *exogène* et se fait entre deux métamodèles différents (source et cible sont dans des espaces technologiques différents).

Les techniques de transformation de modèles permettent de traduire des modèles en d'autres modèles et éventuellement de changer de domaine sémantique. Les transformations doivent être structurées et décrites dans

Affichage/Présentation	Traite la présentation physique des modèles. L'OMG a fait un RFP pour la représentation des modèles au sein d'UML 2.0, où la présentation et le contenu devront être définis dans deux métamodèles séparés.
Capture/Récupération	Construction de modèles initiaux à partir d'un certain existant.
Sérialisation/Transport	Le processus de conversion de l'état d'un objet en une forme enregistrable et transportable. Le standard XMI permet cette sérialisation.
Stockage	Il doit être persistant. Deux formats de stockage existent : objet (CORBA via le mapping MOF IDL et Java via JMI) et fichier (propriétaires via les outils CASE, XML via XMI). HUTN (Human-Usable Textual Notation) est un standard en cours qui permet de créer et modifier des modèles au format texte avec une syntaxe lisible pour l'homme.
Versionnement	C'est la capacité de gérer les versions de métamodèles et de modèles ayant la même version de métamodèle ou non ; grâce à des techniques de représentation différentielle (ex. CVS, ...).
Spécialisation	Spécialiser un modèle pour un domaine particulier en utilisant les profils UML.
Extension	Enrichir un modèle pour obtenir un autre sans perdre les fonctionnalités initiales. Il est intéressant de comparer deux extensions possibles d'un même modèle ce qui implique d'exprimer l'extension sous forme d'un modèle. On aura donc à additionner deux modèles (l'opération dyadique).
Réduction/Filtrage	C'est le fait d'éliminer certains aspects pour n'en retenir qu'une certaine vue sur le modèle en s'assurant que le modèle obtenu reste conforme à un métamodèle standard.
Conversion	Consiste à représenter un modèle dans un autre format sans ajout ni perte d'information, tout en respectant le métamodèle correspondant (ex. changer le format de stockage). Nous pouvons la considérer une opération de transformation.
Vérification	C'est le fait de vérifier que le modèle respecte certaines contraintes. Ces contraintes peuvent être exprimées par OCL (Object Constraint Language).
Mesure	On peut faire des métrologies sur les modèles pour, par exemple, prévoir le nombre de ressources nécessaires ou mesurer le taux de tolérance aux fautes d'un système.
Normalisation	Des travaux visent à élaborer des règles de normalisation pour les modèles de la même façon qu'ils existent pour les SGBDs.
Prototypage rapide	C'est le fait d'associer au modèle une certaine exécutabilité afin d'évaluer quelques propriétés avant de le transformer en code exécutable.
Génération/Traduction de code	Elle permet d'avoir la réalisation d'un PSM en un langage de programmation. Nous considérons qu'elle est un cas particulier de conversion et plus généralement de transformation. Il est possible d'envisager une génération de code générique et paramétrable. Le reverse-engineering reste trop compliquée à réaliser.

Table 3.3: *Opérations monadiques sur les modèles.*

Comparaison/Confrontation	Il s'agit de ressortir les points en commun et les points de divergence entre deux modèles, pour vérifier par exemple la cohérence ou l'équivalence.
Mise en correspondance/Alignement	Il s'agit de mettre en relation les éléments de deux métamodèles afin d'établir une équivalence sémantique entre eux.
Conformité	Un modèle est conforme à un métamodèle si chaque élément du modèle est une instance d'un élément du métamodèle.
Fusion	Elle consiste à unir deux modèles pour en avoir qu'un seul. Si les deux modèles modélisent le même système selon différents points de vue, la fusion doit s'accompagner d'un tissage des aspects.
Transformation	C'est une opération générique incluant toute opération qui permet d'obtenir un modèle à partir d'un autre modèle.

Table 3.4: *Opérations dyadiques sur les modèles.*

des langages qui peuvent, par exemple, s'inspirer de la proposition QVT (Query, View and Transformation) discutés à l'OMG. Elles peuvent être elles-mêmes inscrites dans des modèles[3] conformes à des métamodèles pour capitaliser les transformations. Généralement, les modèles source et cible appartiennent à des formalismes différents et par conséquent à des métamodèles différents.

Une transformation est *inversible* ou sans perte si la transformation inverse existe. Elle est dite d'*optimisation* si elle transforme un modèle en un modèle équivalent optimisé selon une métrique donnée. Elle peut être totalement ou partiellement *automatisée*. La relation entre le modèle source et le modèle cible précise si la transformation a comme résultat un autre modèle cible, ou un autre modèle écrasant le modèle source, ou encore le modèle source modifié.

La transformation consiste en un ensemble de règles à appliquer sur le modèle source. La stratégie d'application des règles détermine la suite des éléments du modèle source sur lesquels les règles vont être exécutées. Concernant l'*ordonnancement des règles*, cela détermine l'ordre d'exécution des règles. Pour l'*organisation des règles*, elle spécifie la structure des règles (orientées source, orientées cibles, indépendantes) ainsi que les relations de composition entre elles. La *traçabilité* précise la manière de garder la trace de la transformation. Les traces peuvent servir aux analyses, au débogage, à la synchronisation entre modèles, ...etc. Le *sens de la transformation* spécifie si les règles sont uni ou bi-directionnelles. Selon le degré d'automatisation, une transformation peut être *complètement manuelle*, *guidée par les profils*, *guidée par les patrons et les marques* ou *complètement automatique*.

Il est à signaler que la transformation de programmes (code source, bytecode, code machine) est un cas particulier de la transformation de modèles. Il s'agit seulement du niveau d'abstraction du modèle source qui peut varier du plus abstrait au plus concret.

La transformation est qualifiée aussi d'*horizontale* si les modèles source et cible sont dans le même niveau d'abstraction. Par contre, si les modèles source et cible se situent dans des niveaux d'abstraction différents elle est dite *verticale*.

Enfin, la transformation de modèles est également utilisée dans la définition des DSMLs pour établir les mappings et les traductions entre différents langages. Ces différentes classes de transformation sont résumées sur la figure 3.3. En reposant en grande partie sur le problème de la transformation de modèles et sa résolution, l'IDM a eu un grand succès et autour de cette problématique les travaux de recherche s'accentuent dans les domaines académiques, industriels et de normalisation.

3.2.3.1 Outils de transformation

On peut grossièrement distinguer cinq catégories d'outils :

- 1catégorie : Langage de programmation «standard», comme Java, qui n'est pas forcément adapté sauf utilisant des interfaces spécifiques (JMI : Java Metadata Interface) ;
- 2 ème catégorie : Les facilités de type langages de scripts intégrés à la plupart des ateliers de génie logiciel. Le langage J dans Objecteering est un exemple. Leur désavantage est qu'ils sont propriétaires et inutilisables en dehors de l'AGL ;
- 3 ème catégorie : Langages liés à un domaine. Le dialecte XSLT dans le domaine XML et AWK pour les fichiers texte sont des exemples.
- 4 ème catégorie : Les outils conçus spécifiquement pour la transformation de modèles et prévus pour être plus ou moins intégrables dans des environnements de développement normalisés. Le standard QVT de l'OMG et le langage ATL constituent des exemples ;
- 5 ème catégorie : Les outils de méta-modélisation dans lesquels la transformation de modèles revient à l'exécution d'un méta-programme comme Kermeta.

Les données qui représentent le modèle manipulable par l'outil de transformation se présentent sous différents formats. En résumé, ils peuvent être :

1. Données sous forme de séquences, comme les fichiers texte. Il s'agit ici de la *transformation de structures*

3. Appelés modèles de transformation.

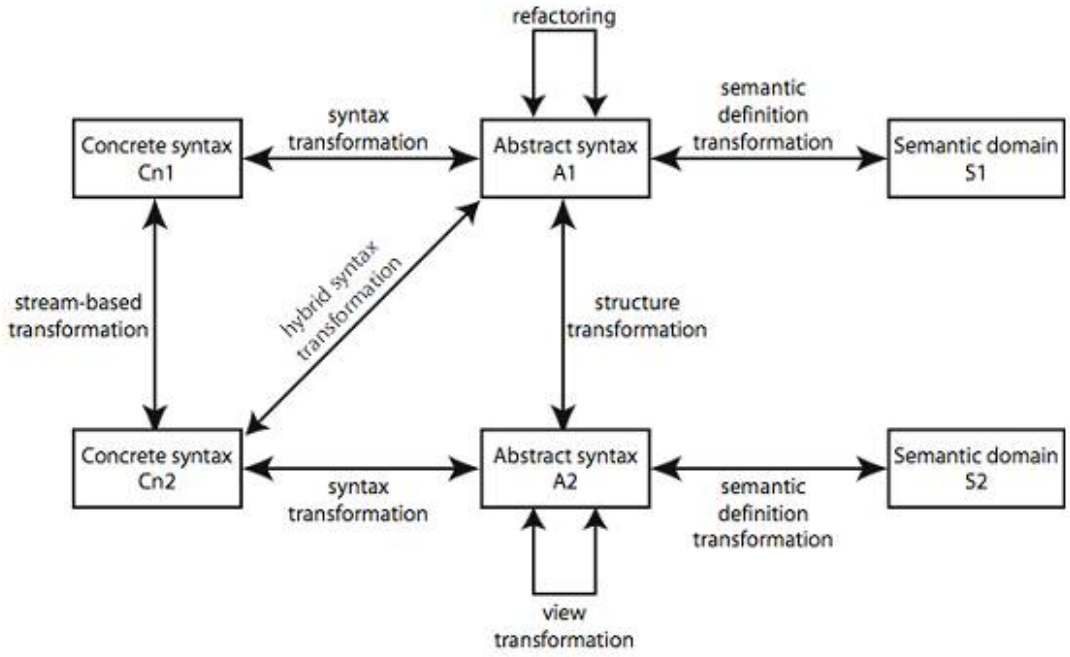


Figure 3.3: *Transformation de modèles dans la définition d'un DSML [Kleppe 06].*

séquentielles d'enregistrements.

2. Données sous forme d'arbres, comme les fichiers XML (et ses dialectes XPATH, XSLT, …). Il s'agit ici de la *transformation d'arbres*.
3. Données sous forme de graphes (et d'hypergraphes), comme les diagrammes UML. Il s'agit ici de la *transformation de graphes*.

La façon d'exprimer la transformation peut être :

- *Déclarative* : lorsqu'on recherche certains patrons (éléments et leurs relations) dans le modèle source, chaque patron trouvé est remplacé dans le modèle cible par une nouvelle structure d'éléments. Ici, l'écriture de la transformation est assez simple mais ne permet pas toujours d'exprimer toutes les transformations facilement.
- *Impérative* : proche des langages de programmation dans le sens ou on parcourt le modèle source dans un certain ordre et on génère le modèle cible lors de ce parcours. Ici, l'écriture des transformations est plus complexe mais permet de toutes les définir.
- *Hybride* : à la fois déclarative et impérative et c'est celle qui est utilisée en pratique dans la plupart des outils de transformation de modèles.

3.2.4 Transformation basée sur la méta-modélisation

Le processus de transformation est composé de trois étapes :

- La définition des règles de transformation.
- L'expression des règles de transformation.
- L'exécution des règles de transformation.

Les deux dernières étapes définissent un système de transformation.

3.2.4.1 Définition des règles de transformation

Étant donné que le modèle source est écrit dans le langage *ls* et le modèle cible dans un autre langage *lc*. Il s'agit dans cette étape d'élaborer une mise en correspondance des concepts de *ls* à ceux de *lc*, voir la figure 3.4.

Dès lors, on a recours à la technique de méta-modélisation pour mettre en place une base de règles exhaustive et générique.

Ainsi, un métamodèle de transformation permettant de définir un langage abstrait de transformation peut être mis en œuvre. Les modèles, instances de ce métamodèle, sont des spécifications de transformations entre deux métamodèles spécifiques. De cette manière, la spécification de la transformation devient lisible du fait que comprendre un modèle est plus facile que comprendre du code. Elle est réutilisable du fait que le métamodèle représente les règles de transformation de manière abstraite et indépendante du langage de sa mise en œuvre.

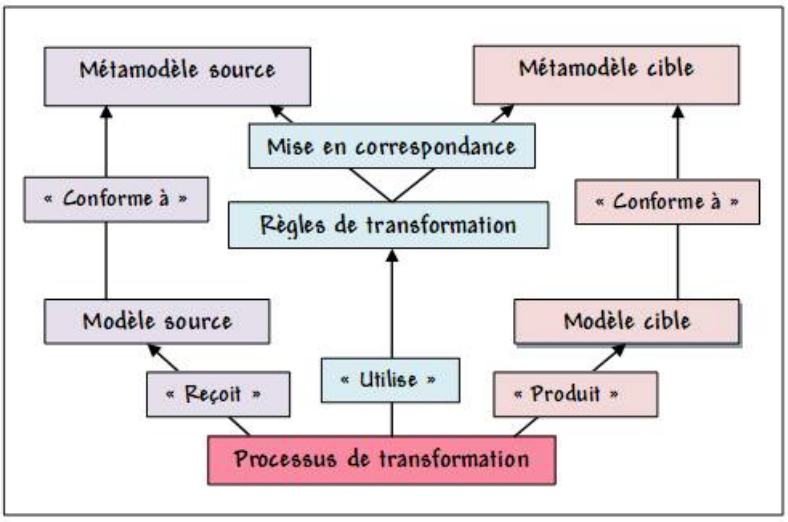


Figure 3.4: *Architecture du système de transformation basé sur la méta-modélisation.*

Le processus de transformation prend en entrée un modèle conforme au métamodèle source et produit en sortie un ou plusieurs autre(s) modèle(s) conforme(s) au métamodèle cible, en utilisant les règles préalablement établies.

3.2.4.2 Expression des règles de transformation

Un langage de transformation peut être déclaratif, impératif ou hybride. Il est à la charge de l'utilisateur de définir le langage de transformation qui répond le mieux à ses besoins et à ses compétences.

Dans la programmation déclarative, on décrit d'une part les données du problème à traiter et d'une autre part les contraintes sur ces données. Le programme s'exécute à partir de la situation courante décrite par les données tout en respectant les contraintes. Le langage déclaratif décrit ce qu'on devrait avoir à l'issu d'un certains nombre de données initiales.

Par opposition à un langage déclaratif, un langage impératif permet de décrire comment le résultat devrait être obtenu en imposant une suite d'actions que la machine doit effectuer.

Un langage hybride regroupe à la fois les paradigmes de programmation déclarative et impérative : l'ordre d'exécution des modules doit être spécifié tandis qu'au sein d'un même module, la détermination de l'ordre d'exécution des règles n'est pas à la charge de l'utilisateur.

3.2.4.3 Exécution des règles de transformation

Une fois spécifiées et exprimées dans un langage de transformation, les règles requièrent un moteur d'exécution pour être exécutées. Ce moteur prend comme entrée le modèle et le métamodèle source, le métamodèle cible, ainsi que le modèle de transformation (les règles de transformation écrites dans le langage de transformation, basées sur les correspondances entre les deux métamodèles source et cible) et son métamodèle (représentant la grammaire du langage de transformation) et produit en sortie le modèle cible. Ce processus est illustré par

la figure 3.4. Le moteur de transformation peut procéder par compilation ou par interprétation.

3.2.5 Transformation de graphes

La transformation de graphes basée sur les grammaires de graphes [Gerber et al 02] est un ensemble de techniques et de formalismes directement applicables à la transformation de modèles. Elle est apparue à cause du manque d'expressivité des approches classiques de ré-écriture comme les grammaires de Chomsky et la ré-écriture de termes pour gérer les structures non-linéaires. Dans ce cas les modèles source et cible sont représentés sous forme de graphes. Cette notation visuelle permet aussi d'exprimer les règles de transformation sous forme graphique[4]. Cette approche trouve son utilité dans le cas où les formalismes manipulés possèdent des syntaxes concrètes visuelles. Elle est formelle et bien fondée sur des bases mathématiques (comme la théorie des graphes et des grammaires formelles) ce qui permet de vérifier certaines propriétés de la transformation.

Cette approche vise à considérer l'opération de transformation comme un autre modèle conforme à son propre métamodèle (lui-même défini à l'aide d'un langage de méta-modélisation, par exemple le MOF).D'après [Vangheluwe et De Lara 02], une grammaire de graphe permet ainsi en tant que formalisme de modéliser une transformation. Une grande partie des outils récents adoptent cette approche avec certaines différences. Pour faire un choix judicieux, les études comparatives des différents outils, selon des critères variés [Taentzer et al 05], semblent nécessaires.

3.3 Spécification formelle des DSMLs visuels

Pour un langage informatique, on distingue la syntaxe concrète notée CS, manipulable par l'utilisateur du langage, la syntaxe abstraite notée AS qui est la représentation interne souvent d'un programme manipulable par la machine. Dans les langages de programmation, la représentation interne sous forme d'arbre abstrait est dérivée de la syntaxe concrète. Ainsi, la syntaxe d'un langage de programmation est définie par les syntaxes concrète et abstraite ainsi que par un lien de dérivation ou d'abstraction, entre la syntaxe concrète et la syntaxe abstraite noté Mca. Ce lien d'abstraction permet d'écarter tout le superflu syntaxique inutile à l'analyse du programme. D'autre part, on distingue également le domaine sémantique noté SD qui représente l'ensemble des états atteignables ou les états possibles du système. La sémantique d'un langage de programmation est alors donnée en liant les constructions de la syntaxe concrète avec les état correspondants dans le domaine sémantique noté Mas.

Un langage de programmation est défini selon le tuple AS, CS, Mca, SD, Mcs où AS est la syntaxe abstraite, CS est la syntaxe concrète, Mca est le mapping de la syntaxe concrète vers sa représentation abstraite, SD est le domaine sémantique et Mcs est le mapping de la syntaxe concrète vers le domaine sémantique.

Généralement, pour la spécification des langages visuels, il existe trois grandes orientations. La première dite *grammaticale* et étend les travaux sur les langages textuels. La seconde est dite *logique* basée sur la logique mathématique (que ce soit de premier ordre, spatiale ou bien d'autres formes) [Janneck et Esser 01]. Son avantage est sa puissance d'exprimer en même temps la syntaxe et la sémantique. La dernière est basée sur la *métamodélisation* et issue de l'IDM. Une autre approche mérite aussi d'être citée ici et s'agit de celle dite *algébrique* qui fait usage des fonctions de composition pour générer des constructions complexes à partir des formes simples.

3.3.1 Les profils UML

Les profils UML introduits par l'OMG avec les notions de stéréotypage permettant l'ajout de nouveaux éléments au métamodèle, les valeurs marquées permettant l'ajout de propriétés à une méta-classe et les contraintes permettant l'ajout ou la modification de règles, offrent aux utilisateurs un moyen d'adapter UML à leurs

4. Le terme *grammaires de graphes* est employé lorsque les règles sont utilisées pour générer un langage graphique et le terme *systèmes de transformation de graphes* est utilisé lorsqu'il s'agit d'un processus de réécriture.

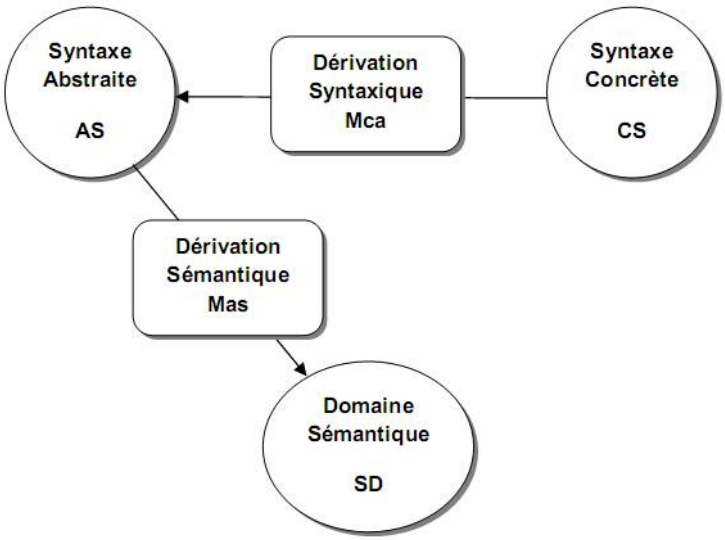


FIGURE 3.5: *Lien entre Syntaxe Abstraite, Syntaxe Concrète et Domaine sémantique.*

usages. Ils sont supposés avoir le même métamodèle qu'UML et peuvent servir à définir ainsi des dialectes ou des DSMLs. L'OMG a définit tout de même plusieurs profils tels que :

- Le profil EDOC (Enterprise Distributed Object Computing) vise à faciliter les développement de modèles d'entreprises, de systèmes ou d'organisations.
- Le profil CORBA pour exprimer la sémantique de CORBA IDL en utilisant une notation UML et pour supporter CORBA IDL dans des outils UML.
- Le profil modélisation temps réel (Schedulability Performance and Time) pour modéliser des applications en temps réels.
- Le profile SysML dédié au domaine de l'ingénierie des systèmes.

Par ailleurs, nous pouvons trouver par exemple le profile AUML spécifique à la modélisation orientée agents où les briques de base sont le diagramme du protocole d'interaction et le diagramme de classes agent. Ces deux diagrammes sont respectivement des extensions du diagramme de séquence et du diagramme de classes.

Bien que les profils sont supportés par la majorité des outils CASE, la spécification des contraintes pour la vérification de la consistance et l'intégrité des modèles est trop compliquée. Les DSMLs doivent offrir potentiellement une puissance d'expression qui dépasse de loin les profiles. La syntaxe concrète sera certainement loin du domaine et l'apprentissage d'UML en plus des concepts du domaine sera une charge supplémentaire.

3.3.2 La syntaxe

Deux courants très actifs peuvent être identifiés pour la spécification de la syntaxe d'un langage visuel. Le premier, fait usage des grammaires de différents types en s'inspire du succès réalisé avec les langages textuels. Le second issu de l'IDM et se base sur les métamodèles.

Les langages iconiques possèdent une base théorique solide et bien établie. Deux sortes d'icônes sont utilisées. Les icônes de traitement exprimant des calculs et les icônes d'objets qui peuvent être élémentaires ou composés. Les icônes élémentaires représentent des primitives du langage, tandis que les icônes composés sont construits en combinant visuellement (à l'écran) les icônes élémentaires. La combinaison est basée sur des opérateurs spatiaux tels que hor (pour horizontal), ver (pour vertical), ovl (pour overlay ou chevauchement) et con (pour connecter). Ce type de langages peut être spécifié en utilisant une grammaire G = (N, T, OP, S, P), où N est l'ensemble des non-terminaux, T est l'ensemble des terminaux qui dénotent les icônes, OP est l'ensemble d'opérateurs relationnels spatiaux, S est le symbole de départ et P est l'ensemble de productions ayant dans le coté droit une expression qui implique des opérateurs relationnels.

Pour les langages diagrammatiques, nous pouvons trouver dans la littérature une variété de formalismes pour

leur spécification. Cela est dû à l'absence d'un consensus sur les éléments primitifs de tels langages. Parmi les formalismes utilisés pour leur spécification, nous pouvons citer :

1. Grammaire attribuée[5] : Les attributs sont utilisés pour manipuler les informations liées à la disposition spatiale des symboles (qui reflètent la nature de l'interface graphique) et la vérification de types. Une production durant le parsing n'est applicable que si les contraintes sur les valeurs des attributs sont vérifiées. Les grammaires de position (Positional grammars) [Costagliola et al 97], les grammaires relationnelles (Relational grammars) [Wittenburg et Weitzman 98], Constraint Multiset-grammars [Marriott 94], Picture Layout Grammars [Golin et Reiss 89] qui est bien adaptée aux langages orientés géométrie, font partie de cette classe. Par exemple l'application des grammaires relationnelles aux langages multidimensionnels pour définir leur syntaxe visuelle a été proposée dans [Crimi et al 91]. Les attributs peuvent participer à la spécification de la sémantique statique et dynamique du langage.
2. Grammaires non attribuée : D'autres grammaires n'utilisent pas d'attributs. Elles spécifient les relations entre les symboles dans les multi-ensembles à un haut niveau d'abstraction, moins indépendant de l'implémentation de l'interface graphique sous-jacente.

   Les grammaires symbole-relation (Symbol-Relation grammars) [Ferrucci et al 96] et plusieurs formes de grammaires de graphes tel que edNLCgrammars [Janssens et Rozenberg 80], grammaires d'hypergraphes (Hypergraph grammars) [Minas 97] et grammaires de graphe par couches (Layered graph grammars) [Rekers et Schürr 97] appartiennent à cette classe.

   Les grammaires utilisées pour spécifier les langages diagrammatiques étendent, en générale, celles utilisées pour les langages textuels. Elles utilisent des ensembles ou des multi-ensembles de symboles et spécifient plusieurs relations entre objets plutôt que la seule relation de concaténation. Par conséquent, l'analyse des langages visuels est beaucoup plus difficile. D'autres méthodes de spécification formelle sont basées sur les systèmes de réécriture [Najork et Kaplan 93].

Tool	Grammar Formalism	Type	Modeling	Environment Specification	Supported Editing
VLDesk	Extended Positional Grammars	Picture	Connection Geometric	Visual Textual	Free-hand
PROGRESS	Layered Graph Grammars	Graph	Connection Geometric	Visual Textual	Syntax-directed
Penguin	Constraint Multiset Grammars	Multiset	Connection Geometric	Textual	Free-hand Syntax-directed
GENGED	Contextual Layered Graph Grammar	Graph	Connection Geometric	Visual	Free-hand Syntax-directed
DIAGEN	Hypergraph Grammars	Hypergraph	Connection Geometric	Textual	Free-hand Syntax-directed
VISPRO	Reversed Graph Grammars	Graph	Connection	Visual Textual	Free-hand
VLPEG	Symbol Relation Grammars	Relational	Connection Geometric	Textual	Free-hand

Figure 3.6: *Différents outils basés sur les grammaires [Risi 04].*

Une Grammaire de graphe permet de construire de façon itérative un graphe régulier infini à l'aide d'un nombre fini de motifs. Elle est donnée sous la forme d'un axiome, et de règles de réécriture. Chaque règle est composée de deux parties. La partie gauche est un ensemble de sommets étiqueté par une lettre, appelé hyperarc. La partie droite décrit un morceau de graphe qui doit être recollé à partir du motif en partie gauche, et peut lui aussi comporter des sommets reliés par un hyperarc.

5. Trois types d'attributs se présentent : graphiques, syntaxiques et sémantiques.

Par ailleurs, les langages de méta-modélisation tels que le standard MOF de l'OMG, offrent les concepts et les relations élémentaires en termes desquels il est possible de décrire un métamodèle représentant cette syntaxe abstraite. De nombreux environnement et langages de méta-modélisation sont actuellement disponibles pour définir cette syntaxe, citons ici : Eclipse-EMF/Ecore [Budinsky et al 03], GME/MetaGME [Ledeczi et al 01], AMMA/KM3 [Jouault et Bézévin 06] ou XMF-Mosaic/Xcore [Clark et al 04].

La syntaxe concrète d'un langage fournit un formalisme graphique ou textuel, pour manipuler les concepts de la syntaxe abstraite et ainsi pour en créer des «instances». Le modèle ainsi obtenu sera conforme au métamodèle de la syntaxe abstraite. La définition d'une syntaxe concrète ad'hoc est à ce jour bien maitrisée et outillée [Combemale et al 06]. En effet, le problème majeur, actuellement posé, réside dans l'automatisation de cette concrétisation de la syntaxe. La génération automatique d'une syntaxe concrète à partir d'une syntaxe abstraite donnée, par une approche générative, permettant de normaliser la construction des syntaxes concrètes demeure un champ de recherche très actif.

3.3.3 Sémantique formelle

On peut classer la sémantique selon les catégories définies pour les langages de programmation en sémantique axiomatique, dénotationnelle et opérationnelle. L'utilisation de l'une ou l'autre est gouvernée par les besoins de l'application développée (preuve de modèles, exécution de modèles, …) et les outils disponibles.

Pour les langages visuels, une distinction est faite entre *la sémantique superficielle* (réelle) et *la sémantique profonde*. La première permet d'associer chaque construction du langage à un concept telle qu'elle est perçue par les spécialistes du domaine d'application. Elle constitue la représentation textuelle (en XML par exemple) du graphe associé au modèle (ou programme). La seconde s'intéresse à la calculabilité du modèle (ou programme visuel) [Grigorenko et al 05].

3.3.3.1 Sémantique axiomatique

Elle est basée sur des logiques mathématiques et définit une technique de preuve pour certaines propriétés des constructions d'un langages de programmation. Cette technique peut être très générale (par exemple un triplet de Hoare) ou spécifique à la garantie de la cohérence de ces constructions (par exemple le typage). Dans le cadre d'un DSML, la seconde utilisation est exprimée par le biais de règles de bonne formation (WFR : Well-Formed Rules), au niveau du métamodèle. Ces règles devront être respectées par les modèles conformes à ce métamodèle et peuvent être vérifiées par simple analyse statique des modèles.

Dans le cadre de MDA, l'OMG préconise l'utilisation du langage OCL. Il permet d'ajouter des propriétés, principalement structurelles, qui n'ont pas pu être capturées par la définition du métamodèle. Plusieurs outils sont désormais disponible pour l'analyse statique comme OSLO[6]. Un métamodèle permet de définir et donc de contraindre la structure des modèles qu'il décrit. Par exemple, les multiplicités des propriétés spécifient les nombres minimum et maximum d'objets qui peuvent participer à chaque association. Cependant, les langages de méta-modélisation ne permettent pas d'exprimer toutes les contraintes relatives à un métamodèle. Par exemple, une contrainte relative à plusieurs associations du métamodèle et n'est pas possible à assurer par construction.

Sur le plan pratique, il est souvent nécessaire de rajouter des contraintes à un métamodèle dans le but d'assurer la cohérence des modèles qu'il permet de définir. Le problème est que les langages de méta-modélisation ne fournissent pas directement de solutions efficaces. Par conséquent, les contraintes sont tout simplement exprimées en langage naturel. Lorsqu'il est nécessaire de vérifier automatiquement des contraintes, le langage de contraintes OCL (Object Constraint Language) est utilisé. Le langage OCL a été initialement conçu pour ajouter des contraintes sur les modèles UML sous la forme d'invariants de classes et de pré-conditions et post-conditions pour les opérations. Les diagrammes de classes d'UML et les métamodèles MOF (ainsi qu'ECore) étant des formalismes très proches, OCL peut être directement réutilisé pour définir des invariants sur des classes MOF et des pré-conditions et post-conditions pour les opérations d'un métamodèle [Fleurey 06].

6. Open Source Library for OCL http ://oslo-project.berlios.de.

Un ensemble de constructions qui servent à naviguer parmi les objets d'un modèle UML afin de vérifier des contraintes est offert par le langage OCL. Ces constructions n'ont pas d'effets de bord et donc ne permettent pas la création, la destruction ou la modification des objets du modèle. La vérification des contraintes est réalisée sans altérer le modèle. En plus de la possibilité de définir de contraintes, OCL facilite la spécification du comportement des propriétés dérivées et d'opérations toujours dans le cadre de ne provoquer aucun effet de bord sur le modèle.

L'utilisation d'OCL est limitée pour la spécification des corps des opérations des métamodèles, mais il est impossible de spécifier des modifications sur les modèles. OCL demeure une bonne solution pour exprimer des contraintes sur les modèles (partie statique) sans avoir la possibilité décrire complètement le comportement dynamique des modèles.

Dans la figure 3.7, la sémantique statique est exprimée sous forme de règles de bonne formation sur le méta-modèle (*MM1*, *MM2* …), par exemple avec OCL. La vérification statique de ces règles est réalisable utilisant des outils dédiés à ce langage, ou par une transformation vers un modèle de diagnostic.

3.3.3.2 Sémantique dénotationnelle

Le principe de la sémantique dénotationnelle est de s'appuyer sur un formalisme rigoureusement défini pour exprimer la sémantique d'un langage donné [Mosses 90]. Les constructions manipulée par le langage sont simplement considérées comme représentation d'objets mathématiques abstraits. Le principe est d'associer à chaque construction du langage un objet mathématique approprié (nombre, fonction, …). La construction est dite ***dénote*** l'objet mathématique qui est dit une ***dénotation***. Une traduction des concepts du langage d'origine vers ce formalisme est alors réalisée. C'est cette traduction qui donne la sémantique du langage d'origine. La sémantique d'un modèle (ou programme) est obtenu par ***composition*** à partir de ses composants.

Dans le cadre de l'IDM, il s'agit d'exprimer des transformations vers un autre espace technique. Autrement dit, définir un pont entre l'espace technique source et cible [Blay-Fornarino 06]. On parle aussi de sémantique de traduction (translation) et parfois de métamodélisation opérationnelle[7]. L'intérêt est que les liens entre les deux technologies offrent la possibilité de bénéficier des différents outils pour exécuter, vérifier ou même simuler les modèles, offerts par l'espace technologique cible. Ainsi, il est possible d'exécuter le modèle dans un autre formalisme.

Dans la figure 3.7 le modèle d'origine *M* est traduit vers un modèle *Msem* conforme à un métamodèle exécutable *MMsem* (Uppaal, IF...). Cette traduction *Mt1* est bien sûr faite en s'appuyant sur les métamodèles correspondants. Une transformation *Mt2* doit permettre de traduire les résultats de l'exécution/simulation de *Msem* en des concepts définis dans *MMx*.

L'approche MIC a récemment proposé, en réponse à la spécification de la sémantique, un ancrage sémantique dans un modèle formel bien formé ASM (Abstract State Machine) [Gurevich 01], en utilisant le langage de transformation GReAT (Graph Rewriting And Transformation language) [Agrawal et al 06].

3.3.3.3 Sémantique opérationnelle

Elle permet de décrire le comportement dynamique des constructions d'un langage. Autrement dit, elle spécifie non seulement le calcul fait mais comment il se fait. Elle consiste à rendre exécutables des modèles et peut se faire soit par méta-programmation avec par exemple Kermeta ou xOCL [Clark et al 04] pour définir un métamodèle dynamique *MMx*, soit par la réalisation d'une succession de transformations endogènes, voir la figure 3.7.

Dans le cadre de l'IDM, elle vise à exprimer la sémantique comportementale d'un métamodèle afin de permettre l'exécution des modèles qui lui sont conformes.

7. The precise UML group : http ://www.cs.york.ac.uk/puml

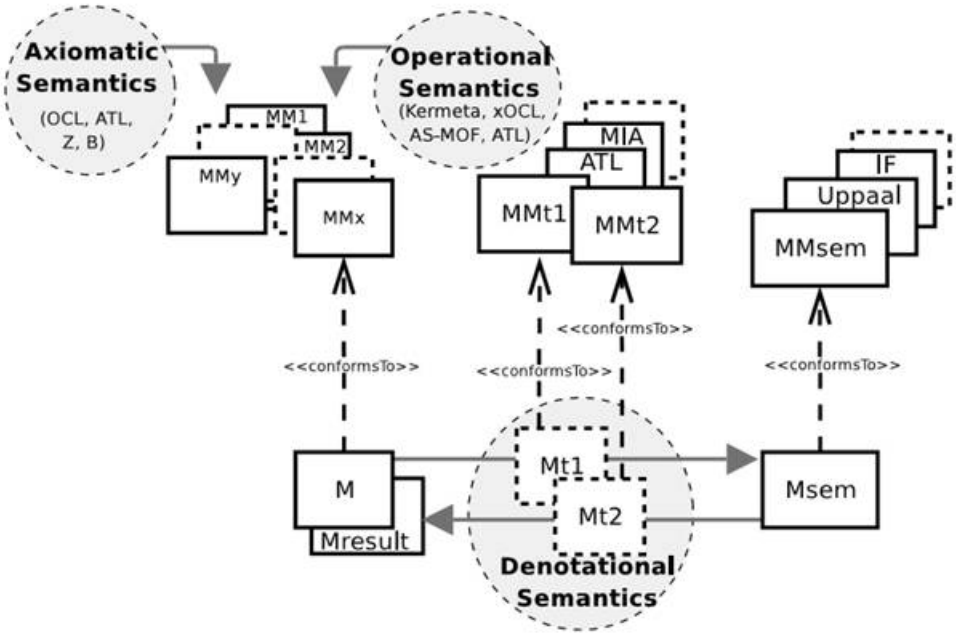


FIGURE 3.7: *Approches pour définir la sémantique des métamodèles [Combemale et al 06].*

3.4 Conclusion

En IDM, la sémantique s'exprime au niveau de la syntaxe abstraite (au niveau métamodèle). Plusieurs syntaxes concrètes peuvent ainsi correspondre à une même syntaxe abstraite (textuelles pour traitement automatique, graphiques pour la présentation, …). Le métamodèle statique donne la syntaxe abstraite ainsi que la sémantique statique et celui dynamique donne la sémantique dynamique. Au lieu de se baser sur des grammaires de différents types, l'IDM tire profit des métamodèles.

La transformation de modèles, clé du succès de l'IDM, est le moyen le plus approprié pour la génération automatique du code. C'est une technique d'opérationnalisation de modèles. Elle se fait actuellement entre syntaxes abstraites en se concentrant sur les concepts des domaines source et destination pour gagner en abstraction.

Cette opération sert aussi, dans le domaine de modélisation de parler de multiformalisme qui permet de cerner tous les aspects d'un système. Selon les principes de l'IDM, la définition d'un système complexe fait généralement appel à l'utilisation de plusieurs DSMLs ayant des relations entre eux et constituent une famille de langages, restreignant ainsi l'ensemble des combinaisons valides des modèles conformes à ces différents métamodèles.

Les outils de méta-modélisation et de transformation de modèles offrent actuellement non seulement la possibilité de spécifier formellement DSMLs, mais permettent également de générer automatiquement les environnements de développement intégrés associés à ceux-ci. Ces environnements, bien qu'intégrés à ces outils de métamodélisation, offrent les fonctionnalités nécessaires à la manipulation des DSMLs comme la création, la modification, la validation, la vérification de modèles (édition) et la génération de code, ce qui augmente la productivité.

Chapitre 4

Concept-maps en modélisation et simulation

Sommaire

4.1 Introduction

La simulation est sans doute l'outil le plus puissant adopté par les décideurs responsables de la conception et de l'analyse des systèmes complexes. Un domaine qui réuni l'art et la science [Shannon 98]. La science se révèle dans tout ce qui est probabilité, statistiques, programmation …etc. La modélisation, l'analyse des résultats et leur interprétation sont beaucoup plus de l'art acquis par l'expérience et le bon sens.

La simulation de systèmes complexes est un projet qui implique une équipe variée et passe par différentes étapes. L'hétérogénéité de l'équipe du projet sur le plan de compétences et de responsabilités implique des moyens de communication efficaces. La succession des étapes à partir de la formulation du problèmes jusqu'à la documentation de la solution passe certainement par différents niveaux d'abstraction du problème et le considèrent sous différentes angles, ce qui nécessitent différents formalismes de modélisation adaptés chacun à une étape bien particulière.

4.2 Projet de modélisation et de simulation

A l'exception de quelques petites études de simulation conduites par un seul expert, un projet de simulation bien réussi provient souvent de la coopération fructueuse de plusieurs intervenants. Il est par essence un

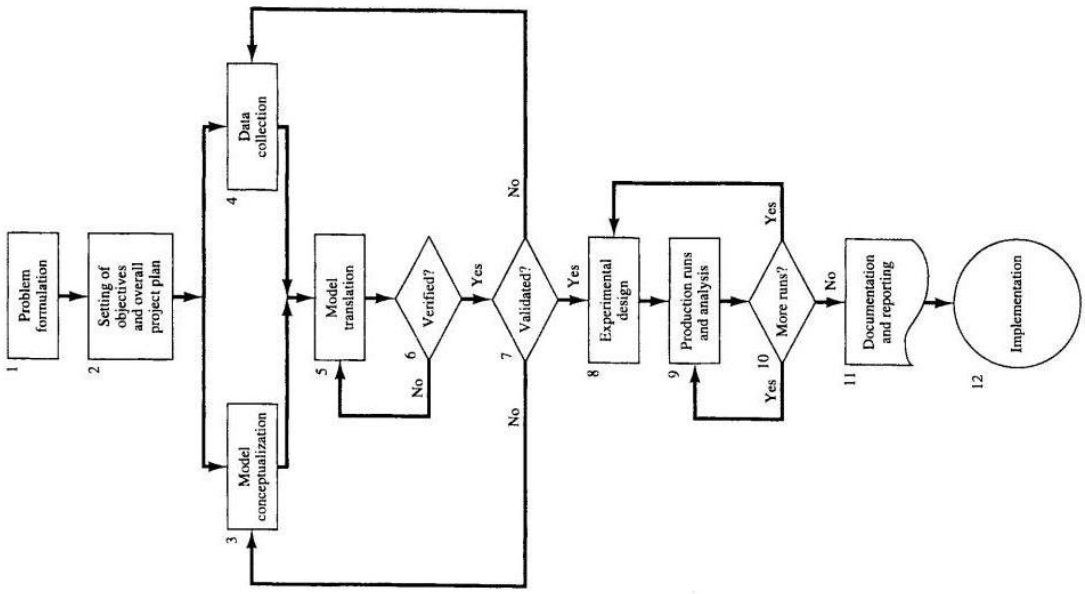


FIGURE 4.1: *Les différentes étapes d'un projet de simulation, [Banks et al 05].*

travail de groupe [Ormerod 01], [Paul et al 05] et [Carson 05]. Les compétences nécessaires pour l'étude des systèmes complexes ne peuvent pas être réunies par une seule personne. Nous préférons parler de rôles que de personnes physiques dans le cas où un seul individu peut jouer plusieurs rôles en présentant divers qualités, ou bien un groupe d'individus collabore à la réalisation d'un rôle.

Plusieurs recherches ont mis la lumière sur la démarche prise en modélisation et simulation [Musselman 92], [Shannon 98], [Law et McComas 01], [Robinson 04], [Paul et al 05] et [Banks et al 05]. Il s'agit, bien évidement d'une étude bien organisée qui à partir d'un problème donné mène à l'implémentation d'une solution adéquate. Les recherches s'accordent sur le principe que c'est une succession de processus à appliquer, mais diffèrent sur un certain nombre de points dont l'appellation, la composition et le nombre de processus impliqués. Les figures 4.1 et 4.2 illustrent les différentes étapes d'un projet de simulation.

La démarche n'est pas linéaire dans le sens où des itérations au sein d'une même étape sont envisageables. Un retour en arrière est possible presque de n'importe quelle étape. De plus le déplacement entre les étapes est souvent dans les deux sens. Une étude menée par [Willemain 95] a bien montré que les experts en modélisation et simulation présentent un *raisonnement fortement itératif localement*, au sein d'une même étape, ainsi que *globalement*, entre différentes étapes.

Certains chercheurs [Robinson 04] ont même démontré que le fait de considérer la vérification et la validation (V & V) comme étapes séparées est trompeur. Celles-ci peuvent être confondues avec les autres étapes, ce qui implique une V & V permanente tout au long du projet.

4.2.1 Formulation du problème

L'analyste essaye de bien comprendre le problème posé et de le formuler clairement en coopérant avec les décideurs et le personnel intéressé par l'étude. Parfois le problème tel qu'il est posé par les décideurs est mal exprimé, ambigu ou incomplet et on doit arriver à un accord mutuel.

Les activités liée à cette étape sont :
- Une définition formelle du problème.
- Justification de l'adéquation de la simulation à l'étude du système.
- Établissement des objectifs spécifiques.

Différents diagrammes ont été recommandés dans [Chung 04] pour la formulation du problème comme les diagrammes Fishbone et Pareto. Le diagramme de causes et effets, diagramme d'Ishikawa illustré dans la

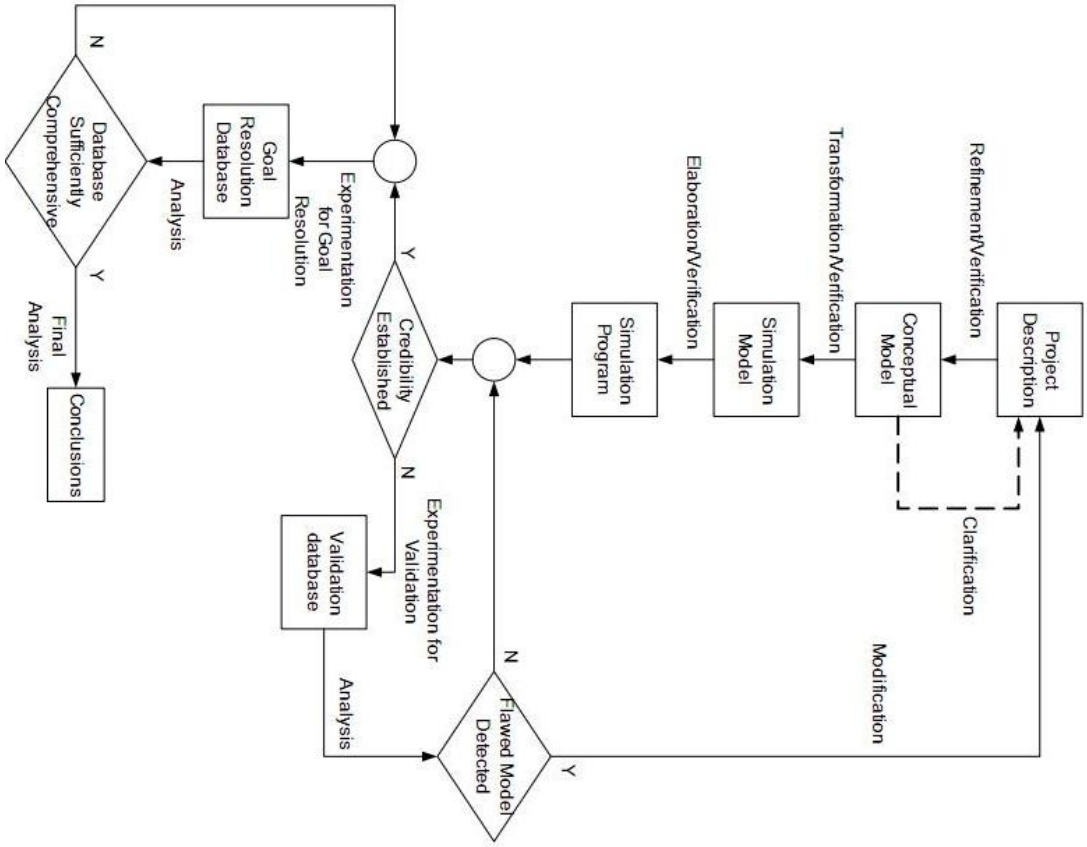


Figure 4.2: *Les différentes étapes d'un projet de simulation [Birta et Arbez 07].*

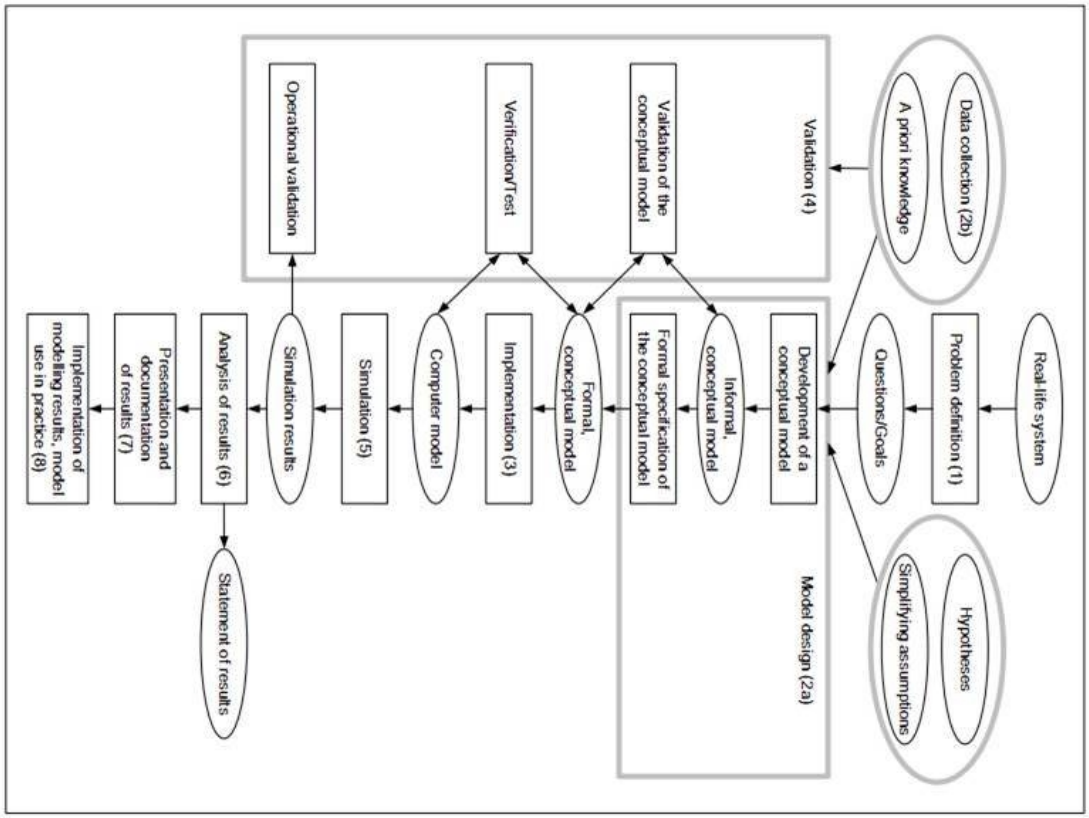


Figure 4.3: *Les différentes étapes d'un projet de simulation [Page et Kreutzer 05].*

figure 4.4 est le fruit des travaux de **Kaoru Ishikawa** pour la gestion de la qualité. Cet outil graphique issu d'un *brainstorming*, recense les causes aboutissant à un effet. Son analyse permet une aide à la décision pour soit corriger un fait existant, soit la mise en place d'un projet. Les causes sont réparties dans les cinq catégories appelées 5M :

1. Matière : Les matières premières, et plus généralement les inputs du processus.
2. Matériel : Concerne l'équipement, les machines, le matériel informatique, les logiciels, et les technologies.
3. Méthode : Le mode opératoire et la recherche et développement.
4. Main d'œuvre : tout ce qui concerne les ressources humaines.
5. Milieu : l'environnement, le positionnement, le contexte.

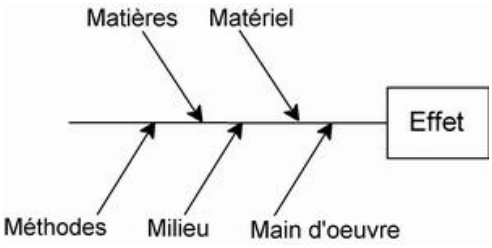


FIGURE 4.4: *Diagramme Fishbone (Ishikawa).*

Chaque branche reçoit d'autres causes ou catégories hiérarchisées selon leur niveau d'importance ou de détail. Le classement doit aussi mettre en évidence les causes les plus directes. Ce sont celles les plus proches de l'arête de poisson.

Le diagramme de Pareto est un graphique représentant l'importance de différentes causes sur un phénomène, voir la figure 4.5. Ce diagramme permet de mettre en évidence les causes les plus importantes sur le nombre total d'effets et ainsi de prendre des mesures ciblées pour améliorer une situation. Ce diagramme se présente sous la forme d'une série de colonnes triées par ordre décroissant. Elles sont généralement accompagnées d'une courbe des valeurs cumulées de toutes les colonnes.

Ce diagramme est construit en plusieurs étapes :
- ✍ Collecte des données.
- ✍ Classement des données au sein de catégories.
- ✍ Calcul du pourcentage de chaque catégorie par rapport au total.
- ✍ Tri des catégories par ordre d'importance.

Pour éliciter les objectifs du projet, d'autres outils sont plus appropriés. D'après [Chung 04] les outils candidats sont :

1. Le Brainstorming : ici les outils déjà disponibles pour la création de concept-maps, mind-maps et dérivés sont de grand intérêt. Les formes de concept-maps discuté au chapitre 1 quelque soit la façon de les exploiter (papier+stylo, tableau+craie, logiciel ou autres) peuvent servir durant cette phase. Le fait que les concept-maps favorise le travail de groupe, tous les membres du projet de simulation impliqués dans l'étude doivent coopérer.
2. Technique du Groupe Nominal : celle-ci est utilisée après l'identification des objectifs du projet. Elle consiste en une session silencieuse d'évaluation de l'intérêt de chaque objectif et se déroule en un processus séquentiel en votant à chaque fois un des objectifs. En d'autres termes il s'agit d'une pondération des objectifs avec possibilité d'élimination.
3. Processus (ou méthode) Delphi : ressemble à la technique précédente en terme de vote. En cas où des influences (politiques, économiques, ou autres) persistent. Après le Brainstorming, le vote se déroule dans l'anonymat pour éviter toute embarras.

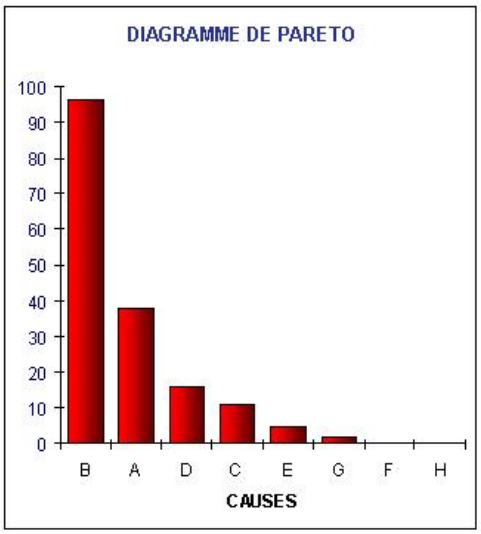


Figure 4.5: *Diagramme de Pareto indiquant la gravité de chaque cause.*

4.2.2 Objectifs et plan du projet

L'analyste, avec le chef de projet de simulation, essaye de :
- Préciser les questions à répondre par l'étude.
- Vérifier l'adéquation de la simulation au problème posé.
- Définir le personnel participant à l'étude.
- Estimer le coût et la durée de l'étude.
- Établir un plan de travail avec des étapes bien claires.

L'élaboration d'un plan de projet est un processus qui consiste au moins à :

1. Établir une structure détaillée appelée (work breakdown structure WBS) : Si on considère un deuxième niveau de détail, chacune des 09 étapes illustrées par la figure 4.1 devra être décomposées en sous tâches. Si un troisième niveau de détail est considéré alors les sous tâches précédemment définies seront à leur tour décomposées ….
2. Établir un graphique de responsabilités appelé (linear responsibility chart LRC) : Ceci revient à affecter chaque sous tâche à des responsables. Les intervenants possèdent des responsabilités distinctes pouvant être primaire, secondaire, assistant, ….
3. Établir un diagramme de Gantt (Gantt chart) : comme tout projet, il est nécessaire d'effectuer un ordonnancement et une gestion. Un diagramme de Gantt permet d'estimer les durées des sous tâches ainsi que les liens entre celles-ci, voir la figure 4.6. Certaines tâches nécessitent que d'autres soient terminées (*finish to start relationship*), d'autres sont synchronisées et doivent démarrer en même temps (*Start to start relationship*) ou bien finir en même temps (*Finish to finish relationship*). Plusieurs autres relations peuvent être exploitées, par exemple le décalage entre deux tâches.

Pour résumer, le déroulement du projet peut être représenté par un concept-map où les arcs sont annotés par les tâches, l'orientation permet de préciser la précédence. Les activités (tâches) factices sont représentées par des arcs pointillés. Les nœuds représentent des évènements (début ou fin d'activités).
- A = Problem statement
- B = Project planning
- C = System definition
- D = Input data
- E = Model translation
- F = Verification
- G = Validation
- H = Experimental design

WBS	Activity	1	2	3	4	5	6	7	8	9	10	11	12
1.0	Problem formulation												
1.1	Orientation												
1.2	Prob. stat.												
1.3	Objectives												
2.0	Project planning												
2.1	WBS												
2.2	LRC												
2.3	Gantt chart												
3.0	System definition												
3.1	Id. comp. to model												
3.2	Id. I/O variables												
4.0	Input data collection												
4.1	Collect data												
4.2	Analyze data												
5.0	Model translation												
5.1	Select language												
5.2	Flowchart of model												
5.3	Develop model												
6.0	Verification												
6.1	Debug												
6.2	Animate												
7.0	Validation												
7.1	Face validity												
7.2	Statistical validity												
8.0	Experimentation												
8.1	Configurations												
8.2	Simulation runs												
9.0	Analysis												
9.1	*t*-Test or ANOVA												
9.2	Duncan test												
10.0	Recs. and conclusions												
10.1	Recommendations												
10.2	Conclusions												

FIGURE 4.6: *Diagramme de Gantt d'un projet de simulation [Chung 04].*

✍ I = Analysis
✍ J = Conclusions

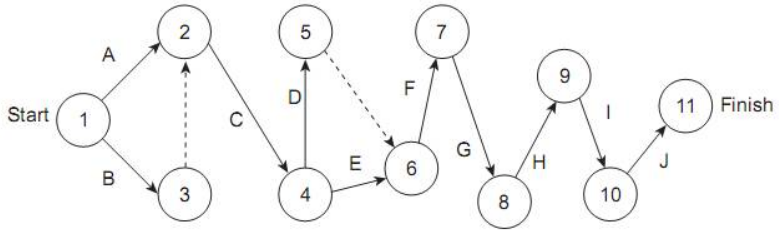


Figure 4.7: *Concept-map pour un projet de simulation [Chung 04].*

Il est très difficile de donner une valeur fixe pour la durée d'un projet typique. Le temps nécessaire pour achever une étude de simulation dépend fortement des délais exigés par les clients. Ce sont les contraintes temporelles et ce sont ces délais exigés par les clients qui dictent l'échelle temporelle du projet et non plus l'inverse. La proportion du temps consacrée à chacune des phases du projet dépend aussi de la nature du problème posé. Les trois grandes phases : construction du modèle conceptuel (collecte et analyse de données incluse), codage de celui-ci pour obtenir un modèle de simulation et l'expérimentation, nécessitent pratiquement les mêmes durées [Robinson 04].

4.2.3 Élaboration du modèle conceptuel

Suivre une approche ascendante du simple au complexe pour bâtir son modèle. L'analyste doit définir les composants (sous-systèmes), leurs interactions, les frontières du système, le niveau de détail à considérer …, pour bien définir le modèle conceptuel. Cette étape est considérée beaucoup plus comme un art et une expertise. Il est courant de trouver dans la littérature des ouvrages qui traitent la définition du système et l'élaboration du modèle conceptuel comme étant deux étapes distinctes. D'autres [Page et Kreutzer 05] considèrent même que l'identification du système se fait en parallèle avec la formulation du problème.

Le premier choix à faire est de décider de la nature du système à étudier et de faire une classification. Il peut s'agir d'un système à évènements discrets, un système continu ou bien hybride. Ce choix parait souvent trivial mais il est aussi possible de modéliser des systèmes discrets comme étant continu (évolution d'une population) ou l'inverse. En plus, il s'agit là aussi de s'assurer de la terminaison du système. Le fonctionnement de certains systèmes se termine après une certaine durée temporelle, tandis que d'autres fonctionnent sans cesse, ce qui peut influencer sur l'état initiale du système ainsi que sur l'existence d'un évènement naturel de terminaison.

Le modèle conceptuel peut être spécifié dans un formalisme suffisamment puissant pour pouvoir décrire le système fidèlement. Il s'agit souvent d'un langage visuel proche des concept-maps mais plus rigoureux sur le plan syntaxique et sémantique.

Après l'identification des composants et de leurs propriétés, des évènements, des processus et des services modélisables, une ébauche du modèle conceptuel peut être construite utilisant un formalisme adéquat. Il peut s'agir d'une simulation qui vise une évaluation quantitative ou bien qualitative.

Les principaux formalismes utilisés pour la modélisation des systèmes à évènements discrets sont les organigrammes, les langages graphiques (Réseaux de Petri de différents types, diagrammes de cycle d'activités (DCA), Diagrammes états-transitions, automates et Réseaux de files d'attente), formalismes mathématiques (algèbre (max, +), chaînes de Markov, Discrete Event System Specification (DEVS)) et des langages de simulation spécialisés (blocs GPSS figure 4.9, Arena/SIMAN figure 4.10, SlamII figure 4.11, …etc. La possibilité d'utiliser les diagrammes UML pour la modélisation des systèmes à évènements discrets a été traitée dans [Page et Kreutzer 05]. Parfois, le modèle conceptuel est décrit textuellement dans un langage naturel, mais pour les formalismes visuels, ils peuvent être considérés pour la plupart des dérivés des concept-maps.

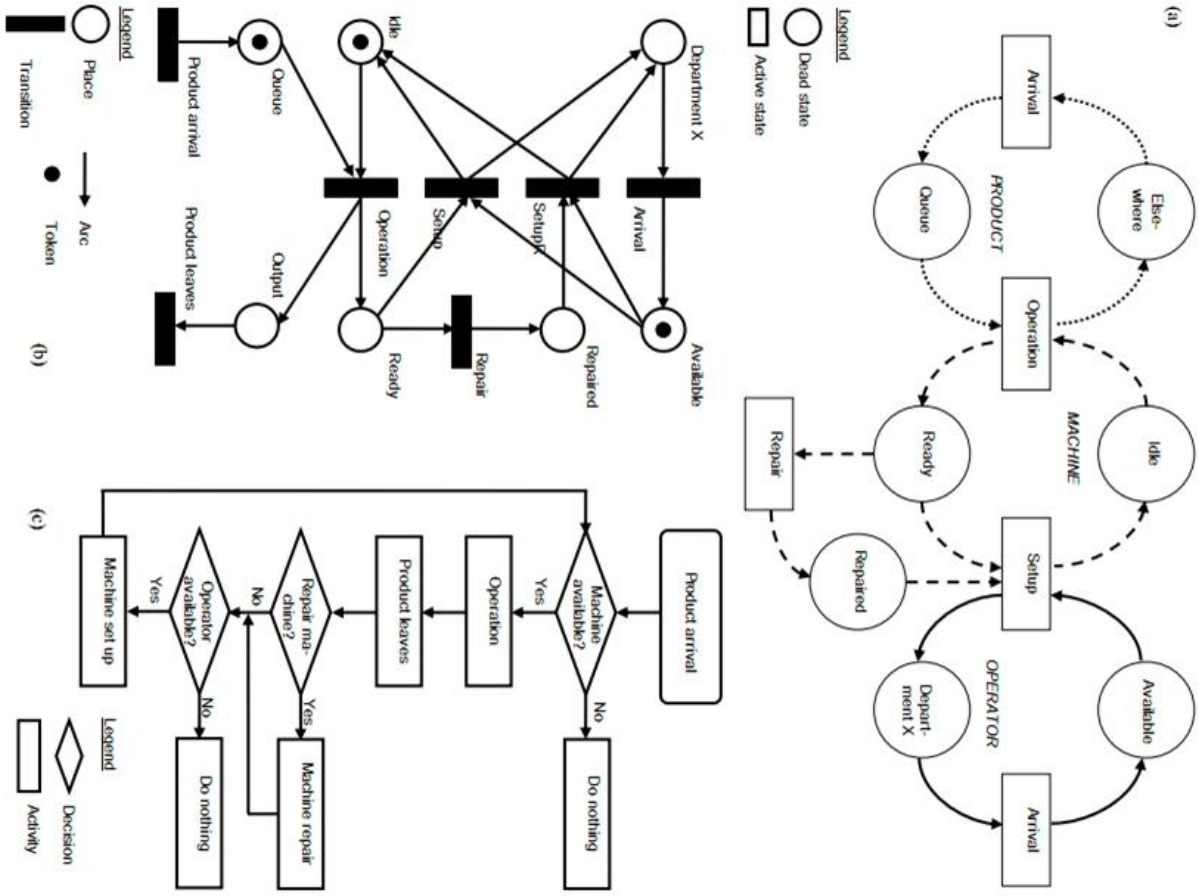


FIGURE 4.8: *Trois modèles conceptuels pour un même système utilisant les formalismes : (a) DCA, (b) RdP et (c) Organigramme [Van Der Zee et Van Der Vorst 07].*

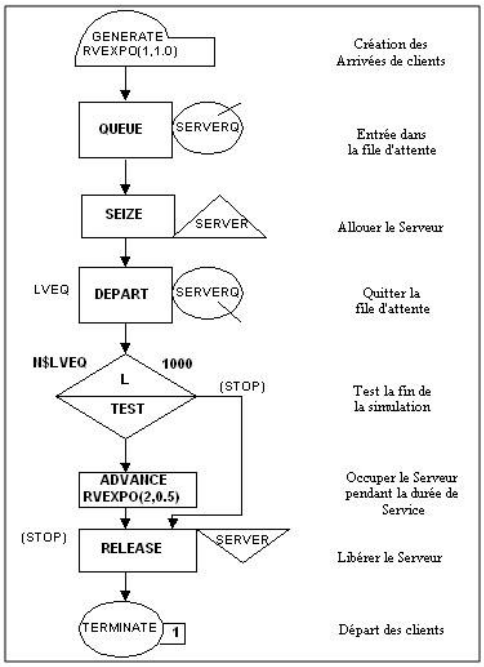


FIGURE 4.9: *Modèle en blocs GPSS pour un guichet de la banque.*

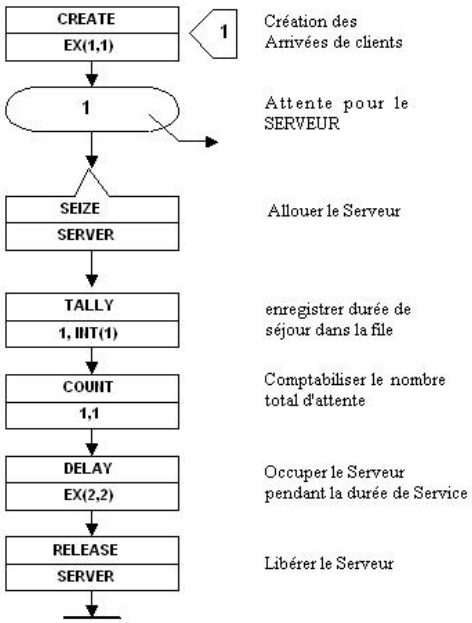


FIGURE 4.10: *Modèle en blocs SIMAN pour un guichet de la banque.*

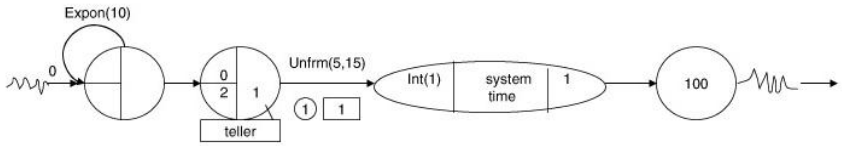


FIGURE 4.11: *Modèle en SlamII pour un guichet de la banque.*

4.2.4 Collecte de données

Deux types de données sont à considérer, les données d'entrée du système et les données de sortie. Les données d'entrée sont celle qui guide le fonctionnement du système, tandis que celle de sortie sont le résultat de fonctionnement de celui-ci.

Deux catégories principales de données d'entrée sont à considérer. La première concerne les *entités* du système et l'autre est liée aux *ressources*. Une liste non exhaustive de ces données inclus :

1. Temps inter-arrivées : les durées temporelles qui séparent l'entrée dans le système des différents groupes d'entités. Les entités peuvent être des clients, des requêtes, des tâches, des pannes, …etc. Ces durées sont souvent probabilistes et suivent certaines distributions de probabilité.
2. Tailles des lots : les entités peuvent se présenter aux entrées du système ou aux ressources pour acquérir des services en groupes (ou lots). Les membres d'un lot (batch) se présentent en même temps et sont traités comme un tout. Une taille d'un lot est souvent probabiliste et suit une certaine distribution de probabilité.
3. Balking, Reneging et Jockeying : certaines entités hésitent à s'introduire dans le système en observant le comportement d'une file d'attente (queue). La décision de ne pas rejoindre le système (balking) peut être probabiliste (ex : selon la longueur de la queue) ou déterministe (ex : queue de capacité limitée). Le désistement (reneging) qui est souvent probabiliste est la décision de quitter la queue après une certaine durée d'attente et avant d'acquérir le service. le Jockeying c'est la décision (probabiliste) que peut prendre une entité pour quitter une queue et rejoindre une autre similaire afin d'acquérir le service le plus tôt possible.
4. Classifications : les entités du système ne sont pas toutes de même nature dans le sens ou leurs cycles de vie sont différents. En plus les entités possèdent souvent des priorités distinctes qui les distinguent

pour l'acquisition de services.

5. Temps de service : les durées temporelle (probabilistes ou déterministes) qui caractérise l'acquisition d'un service (seulement le temps d'occupation de ressources est considéré et non le temps d'attente).
6. Taux de pannes : la fréquence des pannes ou d'indisponibilités des ressources. S'il s'agit de pannes, les temps de réparation sont à considérer.
7. Maintenance planifiée : généralement une maintenance est prévue de façon périodique ou selon un calendrier (déterministe) ce qui rend indisponible certaines ressources. Parfois, la maintenance est probabiliste et elle est liée beaucoup plus au nombre d'entités traitées, durée de fonctionnement, …etc.
8. Repos (Break) : la disponibilité de certaines ressource peut être affecté par des temps de pause ou de repos. Cela est dû essentiellement aux opérateurs humains qui les manipulent. Dans ces circonstances le service peut être interrompu, achevé ou refait et les ressources peuvent bénéficier d'un temps de repos complet ou non.
9. Temps de déplacements : le mouvement des entités dans le système peuvent être instantanés ou non. Parfois il faut modéliser ces durées temporelles qui peuvent être probabilistes ou déterministes.

Les données de sortie représentent généralement des mesures de performances. Une liste non exhaustive comporte :

1. Temps moyen de résidence dans le système (ou un sous-système).
2. Temps moyen d'attente dans une queue.
3. Longueur moyenne de la queue.
4. Taux d'utilisation des ressources.
5. Compteurs statistiques.

Les données d'entrée et de sortie sont collectées par observations, mesures, spécifications, suppositions, recommandations ou à partir de l'historique du système. Il faut que la quantité de données collectées soit suffisante pour pouvoir dégager des conclusions en particuliers pour les tests d'adéquation.

Ceci va permettre de :

- ✍ Analyser les données d'entrée, pour déduire les lois qui gouvernent le système et l'estimation de leurs paramètres.
- ✍ Valider l'adéquation du modèle par la suite.
- ✍ Concevoir les expérimentations.

4.2.5 Translation du modèle

Traduire le modèle conceptuel en un modèle programmé, après le choix de l'outil de mise en œuvre : langage de programmation général, langage de simulation général ou langage de simulation spécialisé dans le domaine de l'étude.

Le modèle conceptuel, étant normalement indépendant de tout choix d'implémentation, peut être exprimé sous forme d'un programme exécutable qui nécessite par la suite une vérification. Si l'analyste ou l'expert en modélisation possède une base solide en programmation il peut s'en servir d'un langage de programmation général ou d'un langage de simulation général. Sinon il doit opter pour un langage de simulation spécialisé. En tout cas une bonne connaissance de l'outil de mise en œuvre est nécessaire.

Les environnements de modélisation et de simulation offrent plus de fonctionnalités qui varient de l'assistance à la collecte et à l'analyse des données en entrée et en sortie, de la construction de modèles conceptuels de façon graphique, débogage animation graphique, …etc.

4.2.6 Vérification et validation du modèle

La vérification permet de décider si le modèle programmé est adéquat au modèle conceptuel ; un débogage est nécessaire. Cette étape est extrêmement importante dans tout projet de simulation. Elle concerne le modèle opérationnel (programme). Il s'agit de s'assurer que le modèle s'exécute sans erreurs. La vérification est primordial même pour des modèles de taille réduite car ces derniers peuvent aussi comporter des erreurs bien qu'ils soient très petits comparés à des modèles de systèmes réels par essence complexes. Il est vivement recommandé d'attendre jusqu'à ce que le modèle entier soit terminé pour commencer la vérification et que celle-ci se fasse de façon continuelle.

Les recommandations pour réussir cette étape sont :

Suivre les principes de la programmation structurée : le modèle doit être construit de façon modulaire selon une approche descendante. Il est recommandé d'établir d'abord un organigramme (dans le cas où le langage n'offre pas le symbolisme pour construire un modèle) traduisant en termes logiques les opérations qui doivent être réalisées par le modèle avant de passer au codage.

Documenter le modèle : il s'agit de commenter toutes les lignes du programme, expliquer le sens et l'utilité de toutes les données utilisées dans le programme. Normalement, une personne n'ayant pas participé au codage devrait être en mesure de comprendre ce que fait le programme.

Faire contrôler le modèle par différentes personnes : le fait de contrôler le modèle par plusieurs personnes vise à augmenter les chances de déceler toutes les erreurs. En général, on utilise des techniques inspirées de celles utilisées dans le test de logiciel (Inspection de code) fondées sur un travail de groupe dont les membres sont :

- ✍ Le modérateur : ou chef de groupe.
- ✍ Le concepteur : qui a développé le modèle conceptuel.
- ✍ Le programmeur : qui a traduit le M.C. en programme.
- ✍ Le testeur : responsable de la vérification.

Ces personnes organisent des réunions de travail au cours desquelles le M.C. est discuté, le programme aussi. Les erreurs détectées sont documentées et corrigées. Le processus reprend jusqu'à éradication de toutes les erreurs.

Vérifier que les données sont codées de manière appropriée : si par exemple le temps est exprimé en minutes, toutes les variables de type temps (temps d'arrivée, durée de service, temps d'attente, …) doivent être exprimées dans la même unité (et non en secondes, heures, …).

S'assurer que les sorties varient raisonnablement en fonction des entrées : par exemple si on remarque que le nombre d'entités dans une file d'attente est de 100 alors qu'on devait avoir au plus 10, il y a sûrement un problème. Ceci peut par exemple provenir du fait que la file d'attente est associée normalement à une ressource ou serveur ayant une capacité de deux unités (2 serveurs en parallèle) alors que dans le modèle, cette ressource a été modélisée comme ayant une capacité d'une seule unité.

Utiliser les outils de débogage avec le langage : certains langages de simulation offre des outils de mise au point (ex : SIMAN) qui facilitent beaucoup la vérification du modèle. Ils permettent de scruter la valeur de n'importe quelle variable, de poser des points d'interruption, …etc.

Utiliser l'animation graphique si l'outil de mise en œuvre en dispose : certains langages de simulation permettent de visualiser sous forme d'animation graphique tous les changements d'états qui se produisent au fur et à mesure que la simulation progresse. Il devient alors plus facile de détecter les anomalies ou erreurs dans le modèle. Par exemple, une ressource qui ne change jamais d'état est signe d'anomalie. Elle peut être la conséquence d'une erreur de programmation ou de modélisation. Le plus important est que l'erreur devient visible à l'écran et donc plus facile à l'écran.

La validation concerne l'adéquation du modèle conceptuel au système étudié. En utilisant les données précédemment collectées, le comportement du modèle par rapport à celle du système est vérifié. Dans le cas où le système existe, la façon idéale de valider le modèle conceptuel est de comparer ses sorties avec celles du

système. Malheureusement, on n'a pas toujours cette possibilité, en particulier dans les projets de conception de nouveaux systèmes.

Les techniques de validation utilisées en simulation peuvent être classées en deux catégories : les *techniques subjectives* et les *techniques objectives*.

4.2.6.1 Techniques de validation subjectives

Les techniques les plus utilisées sont :

Validation d'apparence (Face Validation) : c'est la première validation à faire et qui consiste à soumettre le modèle conceptuel à des personnes familières du système réel modélisé. Ces personnes seront chargées de porter des critiques sur le modèle conceptuel, le but étant de s'assurer que toutes les suppositions ou hypothèses faites dans le modèle sont correctes. La crédibilité du modèle augmente au fur et à mesure que les anomalies détectées dans le modèle sont corrigées.

Analyse de sensibilité (Sensitivity Analysis) : il s'agit de tester si un changement dans les données d'entrée du modèle, entraîne un changement prévisible dans les sorties. Par exemple, si le taux d'arrivée des clients croit, le temps d'attente dans la file devrait croître aussi. Il s'agira en fait de déterminer si le modèle réagit correctement aux changements de valeurs de certaines données d'entrées et que cette réaction est bien celle à laquelle on s'attendait.

Test de conditions extrêmes : il s'agit de tester si le modèle se comporte normalement lorsque les données d'entrée sont à leurs valeurs maximales. Par exemple, si le taux d'arrivée des clients est fixé à sa valeur maximale, cette valeur aura-t-elle comme effet en sortie un accroissement du temps d'attente dans les files, du temps de séjour des clients dans le système, du nombre de clients dans les files, …etc.

Validation des Hypothèses : il y a deux types d'hypothèses (suppositions) qu'on peut faire concernant le modèle conceptuel : des hypothèses structurelles liées au fonctionnement du système modélisé et des hypothèses sur les données du modèle.

Les hypothèses structurelles peuvent être validées en observant le système, en les discutant avec le personnel qualifié. Il faut préciser qu'une personne seule ne peut en aucun cas connaître à elle seule tout sur le système réel et on sera amené impérativement à consulter plusieurs personnes de différentes spécialités (opérateur sur machine, ingénieur, chef d'atelier, administrateur, …etc.)

Les hypothèses ou suppositions concernant les données du modèle doivent aussi être validées. Ceci dépend évidemment de la nature des données. Si on a supposé par exemple que le temps séparant les arrivées des clients est une variable indépendante qui suit une distribution exponentielle, la validation d'une telle hypothèse consistera à :

- ✍ Faire une collecte des données sur les temps séparant les arrivées de clients.
- ✍ Effectuer un test statistique pour s'assurer que l'hypothèse d'indépendance de cette variable est acceptable.
- ✍ Estimer le paramètre de la distribution supposée (exponentielle dont le paramètre est une moyenne).
- ✍ Effectuer un test d'adéquation (goodness-of-fit) pour s'assurer que l'hypothèse d'une distribution exponentielle est aussi acceptable dans un certain intervalle de confiance.

Tests de Turing : il s'agit de comparer les sorties du modèle à celles du systèmes réel. Cette technique consiste à préparer un nombre N de résultats en faisant des expérimentations avec le modèle. On prépare aussi N résultats à partir d'observations du système réel. Ces résultats sont présentés sous le même format et contiennent les mêmes types de mesures. On les soumet à une personne familière du système (donc habituée à ces types de mesures). Si cette personne arrive à distinguer les résultats issus de la simulation de ceux issus des observations du systèmes réel, alors le modèle est considéré comme inadéquat. Dans le cas contraire, on considère que le modèle de conceptuel est une représentation fidèle du système réel et peut le remplacer en vue de réaliser des expérimentations.

4.2.6.2 Techniques de validation objectives

Ces techniques sont basées sur l'utilisation des méthodes statistiques en vue de valider les résultats produits par simulation.

Validation des sorties (test d'hypothèses) : cette technique consiste à comparer les sorties du modèle avec celles du système. Ceci suppose donc que le système existe déjà. Une méthode de comparaison très utilisée est celle basée sur le t-test dont le principe est le suivant :

Si la mesure de performance à valider est le temps d'attente moyen d'un client devant un guichet de service. Après observation du système durant 5 jours, on s'aperçoit que le temps moyen mesuré est égal à 3 mn. On effectue 5 simulations avec le modèle (une simulation pour une journée) qui donnent comme résultats pour le temps d'attente moyen : 1.63, 2.22, 3.12, 1.09, 2.11 mns.

Il s'agit alors de tester si l'hypothèse : $H_0 : E[X_i] = 3mn$ est acceptable ou non ($H_1 : E[X_i] \neq 3mns$) où Xi est la variable aléatoire correspondante au temps d'attente moyen durant la ième simulation.

La méthode statistique à utiliser est celle basée sur la distribution de STUDENT qui s'applique au cas des petits échantillons (comme ici). Il s'agira de calculer le paramètre t de la distribution et de se fixer le niveau de confiance à accorder au résultat (95% ou 99%, …), de regarder dans la table donnant t si la moyenne espérée (3mn) est à retenir ou à rejeter. (Voir aussi le test du χ^2).

Validation par les données historiques : il s'agit de conduire des simulations en utilisant les données historiques du système au lieu d'utiliser des données artificielles. On devrait s'attendre alors à ce que les sorties du modèle et celles du système soient en parfaite concordance. Pour valider cette hypothèse puisque les sorties sont par nature aléatoire, on aura besoin de conduire des tests d'hypothèse (Student, snédecor, Chi-deux, …).

Par exemple, les données historiques contiennent les temps séparant les arrivées de clients (TBC) et les durées de service ($TSRV$) pour 5 journées. Si on désigne par W_i le temps d'attente dans la file observé au cours de la journée i, On peut faire une simulation en utilisant les valeurs historiques de TBC_i et $TSRV_i$ (journée i) afin d'obtenir le temps d'attente moyen dans la file Y_i. La validation de ce résultat va consister à déterminer si $H_0 : E[D_i] = 0$ (ou $D_i = W_i - Y_i$) est acceptable ou non. Ceci va demander à conduire des tests d'hypothèses (Student, snédecor, Chi-deux, …) comme indiqué plus haut.

4.2.7 Conception des expérimentations

L'objectif est de déterminer les alternatives et les scénarios à simuler (les changements qu'on veut appliquer au système). Il s'agit ensuite de définir pour chaque scénario devant être simulé ou expérimenté un certains nombre de paramètres tel que :

- Durée de la simulation (la longueur de d'exécution de l'expérience).
- Nombre de simulations à faire (Réplications ou nombre de fois à refaire l'expérience de simulation).
- Etat initial du modèle.
- Règles de gestion des files d'attente.
- …etc.

4.2.8 Analyse des résultats

Le modèle opérationnel ou programmé est le support principal pour réaliser une simulation sur ordinateur. Il sera analysé et interprété par le simulateur qui délivre en sortie des résultats purement statistiques (moyenne, variance, écart type, minimum, maximum,...). L'analyse de ces résultats aura pour objectifs d'estimer les mesures de performances des scénarios qu'on a expérimenté et prendre les décisions nécessaires.

4.2.9 Documentation et implémentation

La documentation est nécessaire pour différentes raisons et concerne aussi bien le modèle que les résultats de la simulation. Si le modèle aura un jour à être réutilisé par d'autres personnes, la documentation les aidera à comprendre le fonctionnement du modèle et leur facilite toute modification ou mise à jour.

Si le projet à été réalisé pour un client, ce dernier est plus confiant s'il dispose d'une documentation sur le projet. En effet, il aura la possibilité de revoir toutes les alternatives prises en considération, les critères de comparaison qui ont été utilisés, les recommandations faites par l'analyste. Ceci va l'aider énormément dans la prise de décision qui sera principalement basée sur les résultats fournis par la simulation et rapportés dans la documentation.

L'objectif de toute simulation est de proposer pour un problème plusieurs solutions. Le choix de la meilleure solution devra être fait par l'analyste qui la justifiera dans la documentation et la propose (et ne l'impose pas) au client. La décision de retenir cette solution pour une éventuelle implémentation reste donc une responsabilité du client.

C'est ainsi que la qualité et la richesse de la documentation peut avoir une grande influence sur cette étape. De plus, l'implication du client tout au long de la conduite du projet augmente les chances d'une implémentation de la solution retenue. Le succès de l'implémentation de la solution obtenu suite à l'analyse des résultats est fonction du bon déroulement des étapes précédentes.

4.3 Choix d'un outils de simulation

Le problème majeur réside dans le choix de l'outil ou du langage de mise en œuvre. Plusieurs études cherchaient à établir une liste de critères dont souhaitaient disposer les utilisateurs au sein d'un outil de simulation ont été mené comme par exemple [Banks 91], [Hlupic et Mann 95] et [Tewoldeberhan et Bardonnet 02]. Les résultats de ces études sont très significatifs vu la notoriété des sociétés considérées par l'étude (IBM, Pritsker & Associates, CACI, Xerox, Aerospace, …etc.) et dont certaines ont une grande expérience dans le développement du logiciel de simulation (L.S) et d'autres comme utilisatrices de ce type de logiciel. Nous pouvons dans un premier temps dix critères classés par ordre de priorité qui sont :

1. Interface H/M conviviale.
2. Possibilité de disposer d'une base de données pour organiser les données.
3. Outil de mise au point (Débogueur).
4. Interaction via une souris.
5. Section d'assistance dans la documentation du L.S.
6. Sauvegarde des modèles de simulation.
7. Sauvegarde des Résultats.
8. Entrée des données et des commandes par clavier.
9. Librairie de modules réutilisables.
10. Affichage graphique des données et résultats statistiques.

L'étude détaillée menée par [Tewoldeberhan et Bardonnet 02] pour évaluer les outils de simulation à évènements discrets a démontré que l'équipe d'un projet de modélisation et de simulation adopte une liste de critères pondérés selon les figures 4.12, 4.13, 4.14, 4.15, 4.16, 4.17, 4.18. Les critères ont été classés en catégories.

Le plus intéressant pour nous d'après ces études, demeure l'importance de la construction graphique des modèles. Les utilisateurs semblent plus à l'aise en utilisant un formalisme graphique ($poids = 5$), la génération automatique du programme de simulation ($poids = 3$), l'importation et l'exportation de modèles ($poids = 5$).

Il faut signaler également les difficultés associées à :

Criteria	Weight
Graphical model building	**5**
Merging models	4
Conditional routing	4
Statistical distribution	**5**
Queuing policies	4
Reuse of user defined modules	3
Built-in functions	3
Link to other languages	3
Coding tools and utilities	3
Input from text files	**5**
Input from database	4
Input from spreadsheets	**5**
Automatic data collection	3
Batch input mode	3
Interactive input mode	**5**
Random number generators	**5**
Program generator	3

FIGURE 4.12: *Critères pour l'élaboration du modèle et l'analyse des données en entrée [Tewoldeberhan et Bardonnet 02].*

Criteria	Weight
Documentation	4
Maintenance support	**5**
Pedigree	3
Pre-purchase facility	2

FIGURE 4.13: *Critères concernant le vendeur [Tewoldeberhan et Bardonnet 02].*

Criteria	Weight
Multiple runs	**5**
Automatic batch runs	3
Reset capability	4
Start in non-empty state	3
Interaction with user (in running mode)	2
Warm up period	**5**
Ability to calculate appropriate warm-up period and replications	3
Speed control	**5**
Self executable versions	3

FIGURE 4.14: *Critères pour l'exécution [Tewoldeberhan et Bardonnet 02].*

Criteria	Weight
Integration of animation	3
Library of icons	3
Screen layout	3
Concurrent animation mode	3
Animation on/off feature	**5**
3D animation	1
Animation development feature	3

FIGURE 4.15: *Critères pour l'animation graphique [Tewoldeberhan et Bardonnet 02].*

Criteria	Weight
Error checker	**5**
Interacting debugger	**5**
Multitasking	2
Display features	3
Tracing	3
Breakpoints	4
Running backwards	1
Limits	2

FIGURE 4.16: *Critères pour le débogage, le test et l'efficacité [Tewoldeberhan et Bardonnet 02].*

Criteria	Weight
Standard report generation	4
Report customization	**5**
Integration with statistical packages	3
Integration with other simulation packages	3
Feature for exporting data to database	3
Feature for exporting data to spreadsheets	**5**
Feature for exporting data to text files or word processors	**5**
Optimization	3
Output analysis feature	4
Business graphics	4

FIGURE 4.17: *Critères pour l'analyse des résultats [Tewoldeberhan et Bardonnet 02].*

Criteria	Weight
Cost	2
Connectivity with internet	2
Package interoperability	2
Package link to different animation packages	2
Package has open source code	1
Package application area	**5**
Flow oriented modeling approach	4
High level architecture	2
Capability for continuous simulation	2
Simulation strategy	3

FIGURE 4.18: *Critères spécifiques aux besoin de l'utilisateur [Tewoldeberhan et Bardonnet 02].*

1. La translation du modèle conceptuel qui est une phase très intéressante en modélisation et simulation. Elle reste encore une source considérable d'erreurs qu'il faut éliminer dans la phase de vérification et de validation. Les règles de translation ne sont pas toujours faciles à établir, du fait que cela nécessite en plus d'une bonne connaissance dans le domaine de la programmation, une bonne expertise et un raisonnement saint.
2. La construction de l'environnement de modélisation graphique qui est un composant essentiel dans un outil de simulation, nécessite beaucoup d'efforts de programmation et de maintenance.
3. L'importation et l'exportation des modèles conceptuels en présence de plusieurs formats propriétaires.

D'un point de vue MDE, la construction graphique de modèles conceptuels repose sur un formalisme visuel que l'environnement complet de modélisation est généré automatiquement à partir de la spécification du formalisme (DSML). De la même manière, la translation du modèle conceptuel en un code exécutable n'est qu'une transformation de modèles entre deux formalismes qui peut être automatisée une fois les règles sont identifiées et mises en œuvre.

D'autre part, le problème d'importation/exportation de modèles est grandement facilité en se basant sur un format d'échange standard qui favorise l'interopérabilité et qui joue le rôle de pivot pour la transformation de modèles.

4.4 L'explication en simulation

La notion de l'explication est née avec les systèmes à base de connaissances (SBCs). Ces systèmes peuvent imiter le raisonnement des experts humains pour aboutir à des solutions qui ne sont pas généralement évidentes aux yeux des non experts et parfois même aux yeux des experts. Le module d'explication permet dans les SBCs d'éclaircir le raisonnement et de justifier les résultats obtenus. Le besoin de disposer d'une telle faculté au sein d'un environnement de simulation est motivé par l'insuffisance des outils offerts par les simulateurs à apporter des explications sur la dynamique des modèles simulés. Malgré son intérêt, elle n'a pas fait l'objet de recherches intensives.

L'intégration d'un module d'explication vise à donner la possibilité à l'utilisateur de poser des questions chaque fois qu'il est en face d'une situation qu'il ne perçoit pas bien ou qu'il pense qu'elle aurait pu se dérouler autrement, cela permet de :

- ✍ Mieux comprendre les modèles de simulation.
- ✍ Augmenter la confiance dans les résultats de la simulation.
- ✍ Servir comme outil d'apprentissage de la simulation.
- ✍ Aider dans le débogage des modèles de simulation.

La majorité des logiciels de simulation actuels offrent plusieurs outils à l'utilisateur afin de l'aider à mettre au point des modèles de simulation et de mieux appréhender la dynamique du système simulé, comme l'animation graphique, l'interactivité, le fichier trace d'exécution …etc. Malheureusement, ces outils seuls sont insuffisants, pour apporter des explications à l'utilisateur. Les étude menées dans [Belattar et Laraba 97] et [Belattar et Djoudi 00] ont essayé de démontrer l'importance de doter les simulateurs d'une dimension explicative.

L'investigation de ce sujet menée dans [Bourouis 03] a démontré que la problématique de l'explication dans les modèles de simulation est plus qu'un simple enrichissement du fichier trace. Les messages explicatifs soigneusement choisis réduisent de façon considérable l'ambiguïté et permettent à l'utilisateur de mieux comprendre le fonctionnement du modèle simulé. Ces messages sont généralement longs (nécessitant plusieurs lignes) ce qui encombre le fichier trace. Un fichier trace très volumineux plein de textes explicatifs, à notre avis ne résout pas définitivement le problème de l'explication, l'utilisateur n'a qu'à balayer et analyser ce fichier, ce qui n'est pas une tâche mince, tout simplement parce que l'explication est :

1. Une tâche intelligente qui demande généralement un raisonnement expert sur la trace d'exécution.

2. Une tâche fastidieuse si elle n'est pas automatisée, l'utilisateur se lace souvent avant d'arriver à une réponse en balayant et en analysant un fichier de trace volumineux.
3. Une tâche consommatrice de temps, une fois effectuée manuellement.
4. Les demandes d'explication sont variées et nécessitent plusieurs informations et connaissances en plus de la trace d'exécution que l'utilisateur ne peut pas les avoir toutes.

Donc, il est nécessaire de concevoir un module expert réservé à l'explication, qui à travers un raisonnement sur les modèles de simulation (sous forme de connaissances) pourrait satisfaire les demandes d'explication et de justification.

4.4.1 Connaissances explicatives

Il est impossible de donner une liste exhaustive des questions pouvant être posé par l'utilisateur d'un système de simulation face à une situation inaperçue, durant l'exécution d'un modèle de simulation. Une typologie de ces questions suffira largement. Ce sont les connaissances sur lesquelles portent les questions et les demandes d'explication qui nous intéressent le plus.

Les connaissances explicatives sont celles manipulées par le module d'explication et qui sont nécessaires à son fonctionnement. Trois niveaux de connaissances sont identifiables : les connaissances factuelles, actives et celles de contrôle.

Connaissances factuelles : ce sont les connaissances du domaine qui n'ont pas un rôle actif dans la construction de l'explication, mais elles sont essentielles pour activer les autres types de connaissances :

1. Des connaissances sur le domaine à expliquer.
2. Le modèle de l'utilisateur (interlocuteur).
3. L'historique du dialogue.
4. Les définitions conceptuelles.

Connaissances actives : ce sont les stratégies d'explication et les principes explicatifs qui exploitent les connaissances factuelles. A partir de l'analyse d'une question de l'utilisateur, un processus (stratégie) est déclenché pour la construction de l'explication. Les principes explicatifs sont des connaissances plus heuristiques, ils ne sont pas obligatoires mais leur ressemblance aux conseils permet d'améliorer la qualité de l'explication. En voici deux exemples de principes :

✍ Principe1 : illustrer les concepts complexes par des exemples.
✍ Principe2 : justifier les concepts inconnus de l'utilisateur.

Connaissances de contrôle : Un besoin de l'utilisateur peut être satisfait par plusieurs stratégies où différents principes explicatifs conflictuels interviennent. Les connaissances de contrôle permettent d'effectuer des choix aux moments opportuns, que ce soit au début ou à la fin du processus de construction de l'explication, donc ces connaissances visent deux objectifs :

1. Conditionner (contraindre) l'activation des stratégies d'explication et des principes explicatifs.
2. Évaluer les explications produites en cas ou le choix n'a pas pu s'effectuer au début du processus.

Le plus intéressant pour nous ce sont bien les connaissances factuelles, plus précisément les connaissances sur le domaine à expliquer. Dans le domaine de la simulation, que ce soit des demandes d'explication ou d'argumentation, les questions peuvent porter sur n'importe quel objet du modèle simulé (clients, serveurs, files d'attente …etc.), sur les opérations et les choix réalisés par ces derniers, sur leurs attributs y compris leurs historiques ainsi que sur l'état global du système. Donc pour répondre à ces questions, le module d'explication doit pouvoir acquérir à tout moment ces connaissances essentielles à son fonctionnement.

4.4.2 Connaissances factuelles du domaine de simulation

Les connaissances factuelles du domaine de la simulation qui sont indispensables sont :

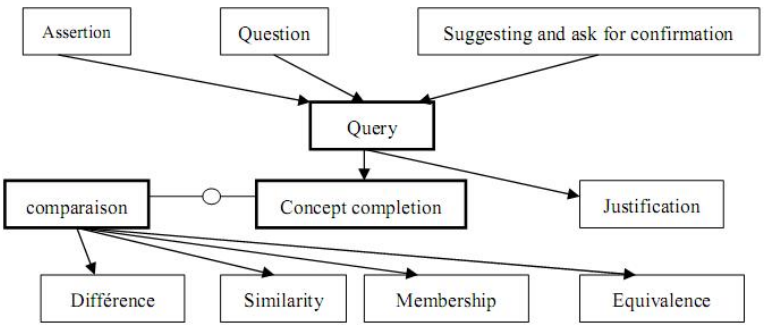


Figure 4.19: *Relations entre types de questions [Sarantinos et Johnoson 91].*

Connaissances sur la partie dynamique du système : Regroupent les modèles de toutes les entités actives, transitoires comme les clients ou permanentes comme les ressources intelligentes. C'est à dire un modèle décrivant en détail leurs séquences d'opérations dans le système et leurs attributs fixes (processus d'arrivée, durée de service, …etc.).

Connaissances sur la partie statique du système : Regroupent les modèles de toutes les entités statiques, les serveurs passifs et les files d'attentes, ainsi que leurs attributs fixes (capacité, durée de service, couleur …) et variables qui décrivent leurs états en fonction du temps de simulation (longueur d'une file d'attente, occupation d'un serveur …etc.).

Connaissances sur les critères de choix : Informations explicatives et justificatives sur tout type de choix pouvant être effectués par le système de simulation au gré des entités, c'est à dire la logique des choix et les variables mis en jeux.

Connaissances sur le cadre d'expérimentation : Nombre de réplications, réplication actuelle, durée de chaque réplication (ou en général le critère d'arrêt de la simulation), temps de simulation actuel …etc.

Connaissances sur la disposition graphique des objets du modèle : Le lien entre l'animation graphique et les objets du modèle simulé (disposition spatiale de chaque objet, nom et numéro identifiant …etc.). L'animation graphique demeure un outil indispensable pour la mise en œuvre de l'explication, c'est à travers celle-ci que l'utilisateur peut observer la dynamique du système et formuler ses questions.

4.4.3 Stratégies d'explication

Nous pouvons considérer le travail de [Sarantinos et Johnoson 91] pour une typologie des questions posées par l'utilisateur comme un modèle pour la définition de la stratégie de réponse appropriée à chaque type, tout en profitant des relations entre les types de questions afin d'améliorer la qualité des réponses. Deux genres de relations entre les types de questions sont à distinguer : multi-concept et hiérarchique, voir la figure 4.19.

Une relation hiérarchique rassemble les types de questions qui sont reliés sémantiquement, par exemple le type *comparaison* possède un nombre de sous-types comme *différences* et *similarités* (caractérisé dans la figure par une ligne annotée par un cercle). Une relation multi-concept relie des types qui sont différents sur le plan sémantique et conceptuel.

Dans le domaine de simulation, sur un plan très pratique, Rothenberg [Rothenberg et al 89] a donné une liste des types de questions que les utilisateurs et les experts en modélisation aimeraient poser à leurs modèles de simulation :

- ✍ **Pourquoi** : Pourquoi un évènement X s'est produit ?
- ✍ **Pourquoi pas** : Pourquoi un évènement X ne s'est pas produit ?
- ✍ **Quand** : Sous quelles conditions l'évènement X se produit ?
- ✍ **Comment** : Comment le résultat R peut être obtenu ?
- ✍ **Toujours et Jamais** : Peut X toujours produire Y ?

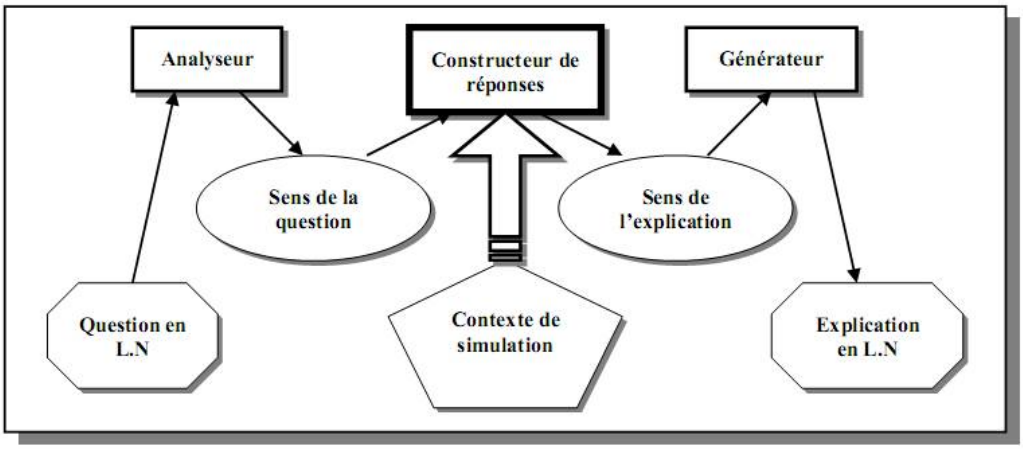


FIGURE 4.20: *Module d'explication.*

- **Optimisation ou dirigée par un but** : Quelles sont les conditions qui produiront la valeure la plus élevée de Z ?
- **État de la simulation** : A quelle date la condition C est vérifiée ?
- **Objets de la simulation** : A quoi ressemble l'entité E ?
- **Historique de la Simulation** : Combien de fois l'évènement X s'est produit ?
- **Causalité explicite** : Quels sont les évènements qui déclenchent l'évènement X ? Quels sont les évènements causés par Y ?
- **Autour du modèle** : Sous quelles conditions le plan X ne parvient pas son objectif ?

Nous pouvons remarquer que la majorité des questions s'intéressent à la dynamique du système, d'autres portent sur la partie statique, sur l'historique (le futur aussi) ou même sur l'analyse de résultats. En se servant de cette typologie, il suffit maintenant de définir, pour chaque type, une stratégie de réponse adéquate, donc il faut identifier les connaissances du domaine qui peuvent servir comme éléments de réponse, puis établir la manière de les acquérir et la façon de les présenter à l'utilisateur.

4.4.4 Le module d'explication

Le module d'explication que nous voulons concevoir correspond à la gestion d'un dialogue de question-réponse dans un langage proche du naturel. De façon classique, il se compose de trois parties principales :

- L'analyse de la question formulée en langue pseudo-naturelle, qui produit une représentation du sens de la question.
- La construction de la réponse ou génération profonde, qui produit une représentation du sens littéral de la réponse.
- La mise en langue de la réponse ou génération de surface, produit le texte explicatif final.

La figure 4.20 illustre les différentes étapes rencontrées dans la production d'une explication . Ce processus est inspiré des travaux dans les domaines de système de dialogue homme-machine (DHM) et ceux de l'explication dans les systèmes à base de connaissance. Pour l'adapter au domaine de simulation, il suffit d'introduire le contexte de simulation dans le processus de construction des réponses.

4.4.4.1 Architecture du module d'explication

Le module d'explication fait partie de l'IHM d'un système de simulation, il doit récupérer les connaissances explicatives nécessaires à son fonctionnement à partir du module de modélisation graphique, la figure 4 .21 montre l'emplacement du module d'explication au sein d'un environnement de simulation générale [Belattar et Laraba 97]. Le translateur traduit le modèle de simulation en un programme exécutable en se servant d'une bibliothèque de simulation. L'exécution du programme permet de faire une animation gra-

phique du modèle, c'est à ce moment que l'utilisateur peut poursuivre l'évolution du modèle et poser des questions. Le module d'explication a besoin du fichier trace de simulation, l'historique des ressources passives et des files d'attentes, il les récupère après exécution du programme de simulation. Les modèles de la partie dynamique (dans un formalisme directement exploitable) et les connaissances sur la partie statique sont fournis par l'interface de modélisation graphique.

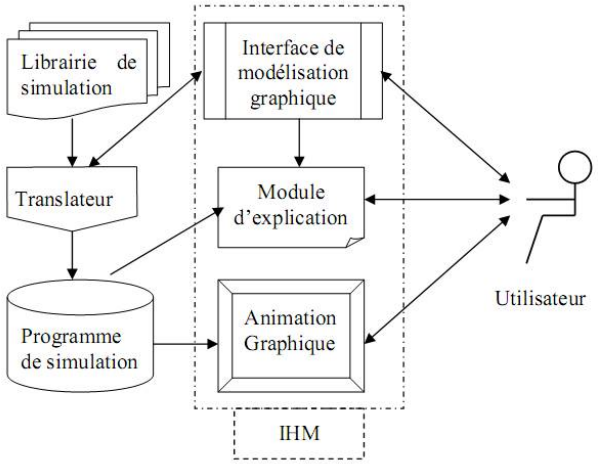


Figure 4.21: *Le module d'explication dans l'environnement de modélisation et de simulation.*

La figure 4.22 représente l'architecture du module d'explication réalisée dans [Bourouis 03]. Elle est basée sur la bibliothèque de simulation Silk [Kilgore 00] et le noyau de système expert Jess. Celle-ci comporte trois grandes parties :

Noyau : Une classe appelée *Explication* qui constitue l'interface entre l'utilisateur et le système d'explication contient un moteur d'inférence sous la forme d'une instance de la classe *Rete* de Jess. Cette classe communique directement avec la bibliothèque de simulation à évènements discrets Silk, pour collecter les informations requises, l'utilisateur peut même interrompre et reprendre l'exécution de la simulation pour poser des questions. Deux sources d'informations et de connaissances nécessaires au fonctionnement du noyau sont le contexte de simulation et les modèles des raisonnements explicatifs.

Le contexte de simulation : Comporte le fichier trace de simulation qui décrit les événements produits avec leurs dates d'occurrence, les fichiers trace des attributs des serveurs passives et des files d'attente, c'est à dire les changements d'état de ces derniers et les dates correspondantes, les modèles décrivant la dynamique du système simulé (modèle conceptuel) et les connaissances sur sa partie statique.

Modèles des raisonnements explicatifs : Sous forme de règles de production, ils permettent de : modéliser les stratégies d'explication (questions utilisateur et méthodes de réponse), collecter les informations nécessaires à la production des réponses et la mise en langue des réponses.

Nous avons choisi de séparer ce composant des autres parties pour garantir l'extensibilité (possibilité d'enrichir ce modèle). Les stratégies de réponse sont regroupées dans un fichier de script Jess, ce fichier comporte aussi des fonctions définies par l'utilisateur (l'utilisateur de Jess bien sure qui est le concepteur du module d'explication) pour la collecte des informations requises par le moteur d'inférence Rete.

L'analyse des questions en langue naturelle et la génération des réponses en langue naturelle sortent du cadre de notre travail, donc l'interface que nous essayons de concevoir ne prend pas en charge ces deux aspects, cela peut gêner l'utilisateur lorsqu'il pose ses questions.

Néanmoins, nous tentons de nous rapprocher de la langue naturelle dans les deux cas. Le fonctionnement du module d'explication est simple, il suffit de créer une instance de la classe *Explication* dans la méthode *Main* du programme de simulation. La classe *Explication* crée à son tour une instance du moteur d'inférence *Rete* et s'occupe de charger les fichiers contenant les modèles de la partie statique et dynamique ainsi que les modèles de question-réponse. Une fenêtre permettant d'interagir avec l'utilisateur sera affichée. L'utilisateur

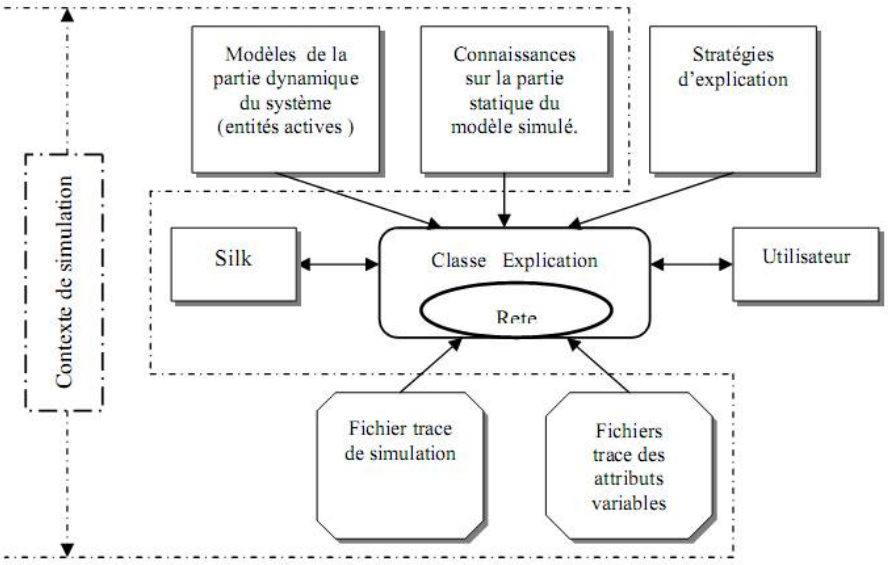


FIGURE 4.22: *Architecture de mise en œuvre.*

peut écrire des questions (une question à la fois) et avoir directement la réponse, la question est sous forme d'un fait Jess et n'est pas en langue naturelle. Après validation d'une question un nouveau fait est inséré dans la base de connaissance et les règles correspondantes sont déclenchées pour générer une réponse.

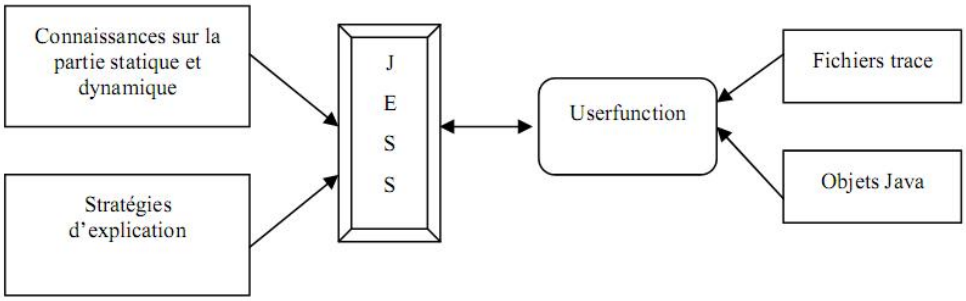


FIGURE 4.23: *Acquisition des connaissances par le système expert Jess.*

Les stratégies d'explication et les modèles de la partie dynamique et statique du système sont dans une forme directement exploitable pas Jess. Les autres connaissances sont sous forme brute, c'est à dire des fichiers textes ou des objets java, dans ce cas et pour pouvoir les exploiter Jess offre une interface appelée *Userfunction* pour l'acquisition des connaissances de l'extérieur, voir la figure 4.23. L'utilisateur (concepteur du module d'explication) peut définir ainsi des fonctions qui implémentent cette interface qui manipulent des objets Java externes, extraire des connaissances et les passer à Jess.

4.5 Conclusion

Le projet de simulation est par essence un travail de coopératif. La nature du groupe impliqué est variée sur le plan compétences et rôles. Cela nécessite des moyens de communication qui sont à la fois visuels et facile à manipuler. Les différentes catégories de concept-maps peuvent servir chacune à un stage particulier :

- Formulation du problème : les techniques et outils de concept-mapping et de mind mapping semblent être idéales, en particulier les outils supportant le travail coopératif.
- Le déroulement du projet peut être définie utilisant un concept-map qui détermine le séquencement des tâches et sous-tâches.

- ✍ L'élaboration du modèle conceptuel : une phase d'investigation pour la compréhension de la dynamique du système étudié, ses composants, leurs interactions, leurs propriétés, …implique la coopération de la majorité de l'équipe d'étude. La représentation des connaissances du domaine et les objectifs de l'étude formulé utilisant des concept-maps par les clients et les décideurs facilitera grandement la tâche des autres membres. Toues les variétés des concept-maps peuvent servir comme moyen de communication. Un formalisme visuel plus formel permet à la fin d'élaborer un modèle conceptuel exploitable par la suite.
- ✍

Les concept-maps constituent une bonne solution pour générer des explication. Le développement d'un module d'explication peut en bénéficier pour contourner la problématique de génération du dialogue en langage naturel. Les éléments de la réponse seraient les composants de base pour produire un concept-map explicatif. En plus, une explication visuelle aurait certainement une meilleure acceptation par l'utilisateur comparativement à un paragraphe explicatif.

Bien que les premières étapes d'un projet de simulation sont toutes propices à l'utilisation des formalismes visuels sous forme de concept-maps. Nous nous concentrons, dans le cadre de cette thèse, sur l'étape de l'élaboration du modèle conceptuel à travers le développement d'un formalisme visuel pour la modélisation des systèmes à évènements discrets qui s'inspire des concept-maps. Définit de façon formelle en se basant sur la théorie des files d'attente, il servira également à la conception ultérieure du module d'explication en fournissant les connaissances explicatives, en particulier celles factuelles concernant les parties statique et dynamique d'un modèle, nécessaire à la production des explications.

Deuxième partie

Conception et implémentation

Chapitre 5

La bibliothèque JAPROSIM

Sommaire

5.1 Introduction

Pour pouvoir exploiter pleinement les constats théoriques discutés dans les chapitres précédents, nous avons conçu un environnement de modélisation et de simulation bâti sur un noyau baptisé JAPROSIM. Ce cadriciel est conçu au départ avec l'idée qu'il s'agit d'une brique de base pour un simulateur à évènements discrets, à la fois extensible et open source, tout en essayant d'automatiser les tâches communes les plus utilisées en simulation pour éviter prématurément certaines erreurs commises par les programmeurs non expérimentés[1]. Le principe clé de la DSM, préconise l'automatisation du cycle de développement pour les experts des domaines, depuis le modèle conceptuel jusqu'au code exécutable. Cette bibliothèque constitue ainsi le niveau d'abstraction le plus bas dans l'architecture et permet de guider le processus de génération de code.

5.2 JAPROSIM : une vue générale

JAPROSIM est avant tout un projet open source qui vise à concevoir un environnement de modélisation et de simulation à évènements discrets. Avec une architecture modulaire, il est tout à fait possible de l'améliorer au cours du temps en lui intégrant de nouveaux modules, voir la figure 5.1. Sa brique de base est le noyau JAPROSIM, qui constitue un simulateur à évènements discrets.

Dans une première étape, l'utilisateur élabore le modèle conceptuel en EQNM^{2}L sous forme graphique, traitée dans le chapitre suivant. Ensuite, le modèle obtenu est sauvegardé dans un fichier XML conforme à un schéma particulier. Le résultat est désormais dans un format standard facilement exploitable par les différents outils de simulation ou d'analyse.

1. Il s'agit beaucoup plus des statistiques usuelles.

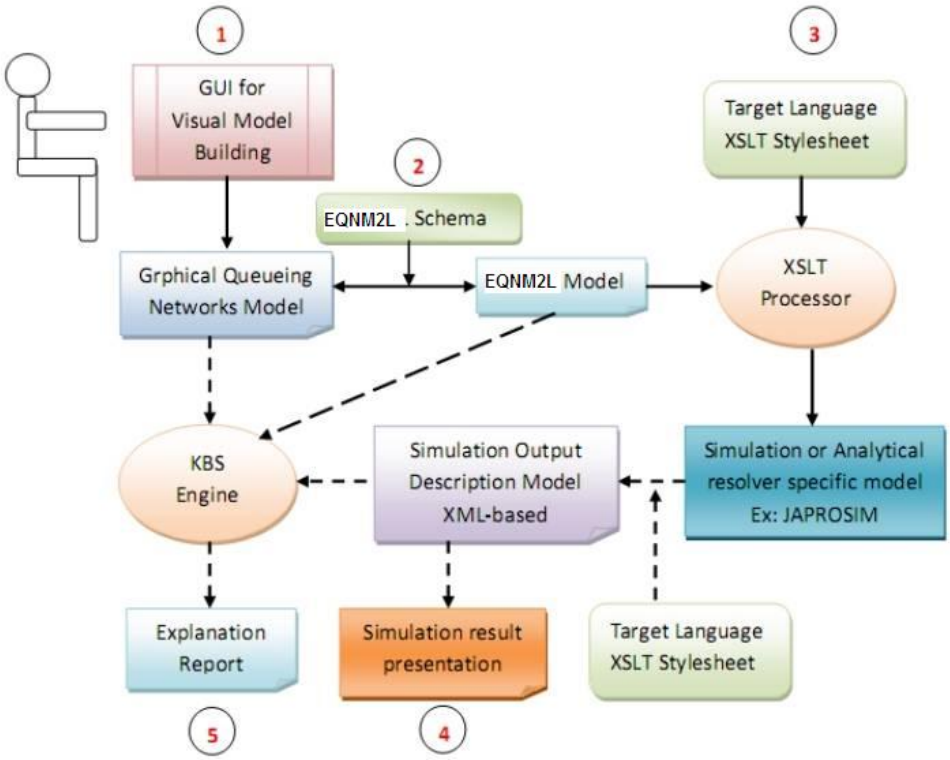


Figure 5.1: *Le projet* Japrosim *[Bourouis et Belattar 08b].*

Durant la 3 ème étape, le modèle conceptuel dans le format standard est transformé dans un format propriétaire spécifique à l'outil utilisé. La figure 5.1 propose des transformations basé sur le format standard en XML et par conséquent, l'utilisation d'un moteur de transformation supportant XSLT. L'utilisation de XSLT n'est pas obligatoire et tout autre langage de transformation de modèles suffisamment puissant et expressif peut le remplacer.

Les résultats de la simulation sont codés sous forme XML standard (conforme à un schéma particulier), dans la 4 ème étape, pour faciliter leurs traitements. Ainsi, la présentation graphique des résultats dans différentes formes (pie charts, histogrammes, …etc.) par différents outils serait ainsi facilitée grâce aux transformations de modèles.

Enfin, quelques scénarios et résultats de la simulation peuvent paraitre ambigus et nécessitent une compréhension profonde. Le modèle conceptuel et les résultats de simulation sont à la base de la génération automatique des explications suite aux requêtes formulées par l'utilisateur. L'explication est une tâche intelligente basée sur un système à base de connaissances (SBC) qui évalue les requêtes et décide de la stratégie à suivre pour générer les explications adéquates (en fonction d'une typologie de questions et le niveau d'expertise de l'utilisateur).

5.3 Simulation à évènements discrets par interaction de processus en Java

L'approche par interaction de processus, dans la simulation à évènements discrets, doit ses origines aux auteurs du langage SIMULA. La dynamique du système est représentée à travers ses entités actives. Tout comme en SIMULA, les elles sont transitoires et se déplacent dans le système (entités dynamiques). Un modèle selon cette approche est une description de la séquence des étapes de traitements effectués par ces entités. Cette approche plus naturelle et intuitive est actuellement dominante. Elle est supportée par la majorité des environnements de simulation. L'approche par flux de transactions n'est qu'un cas particulier de celle-ci.

Le système est modélisé sous forme d'un ensemble d'entités en interaction. L'interaction est la conséquence de la compétition et/ou la coopération pour l'acquisition des ressources critiques. Le cycle de vie de chaque

entité active est une séquence d'évènements, d'activités et de reports (delays). La routine qui implémente ce cycle de vie implique des mécanismes spéciaux pour interrompre, suspendre et reprendre son exécution ultérieurement sous le contrôle d'un ordonnanceur interne. Ce qui nécessite un langage de programmation offrant au minimum les coroutines de SIMULA pour l'implémentation. Donc, les langages qui offrent le multithreading comme Java sont de bons candidats.

Le cycle de vie d'une entité active est une séquence de phases passives et actives. D'une part, une phase active est caractérisée par l'exécution du processus (ou thread) correspondant. Normalement, cela correspond aux évènements durant lesquels l'état du système change sans progression du temps de simulation. D'autre part, une phase passive est caractérisée par des activités et des reports qui provoquent la suspension du processus associé et la progression du temps de simulation.

Les évènements sont à la base de l'ordonnancement, ce qui explique l'utilisation d'une liste des évènements futurs (Future Event List FEL). L'ordonnanceur n'est qu'un processus particulier chargé de la coordination de l'exécution du modèle de simulation.

Java est un langage de programmation permettant de créer des applications sûres, portables, robustes, orienté-objet, concurrentes (multithread) théoriquement pour n'importe quel domaine. Il offre un ensemble de bibliothèques riches pour le développement des interfaces graphiques, des applications réseaux et distribuées. Il offre aussi des bibliothèques utiles pour la manipulation des types de données tel que les listes, les vecteurs, les tables, ce qui justifie notre choix pour Java comme langage d'implémentation. Le concept-map ci-dessous résume les caractéristiques essentielles de Java.

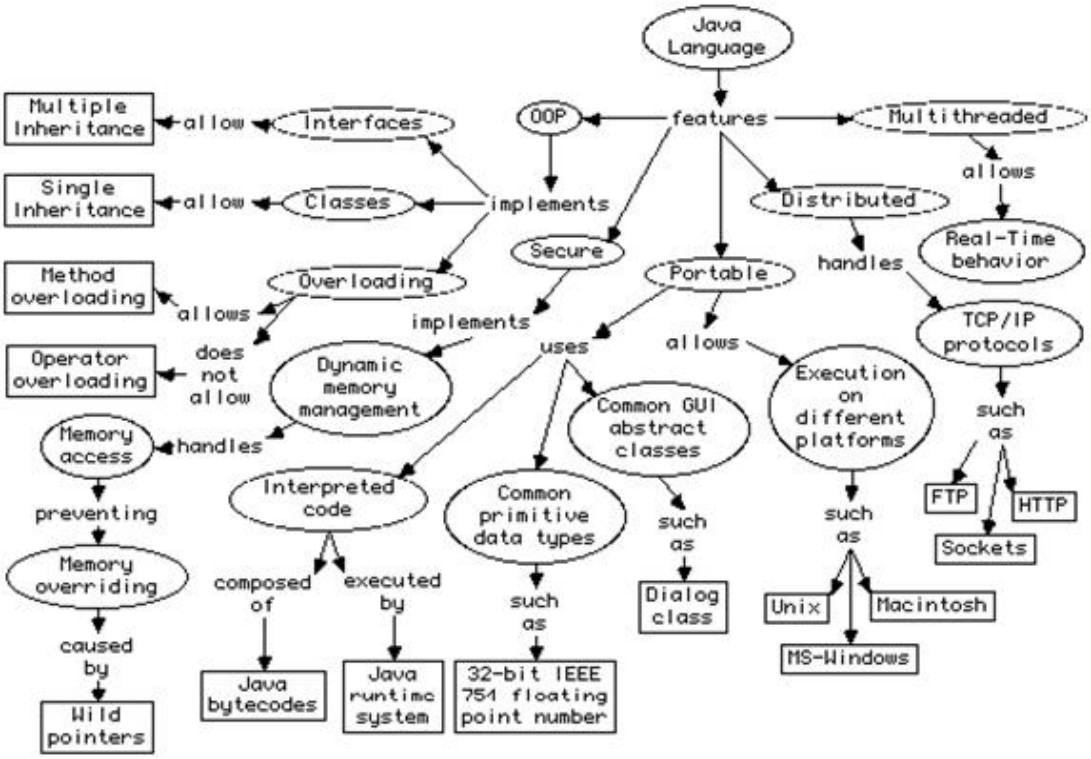


Figure 5.2: *Un concept map pour le langage Java*

5.4 Librairies similaires

L'idée de concevoir des simulations par interaction de processus utilisant un langage de programmation orienté-objet n'est pas nouvelle. Plusieurs outils ont été développés dans cette optique. Par exemple, CSIM++ [Schwetman 95] et YANSL [Joines et Roberts 96] sont basés sur C++, tandis que PsimJ [Garrido 01] et JSIM [Miller et al 98] sont basés sur Java. Cependant, Japrosim présente certaines caractéristiques uniques :

- Des bibliothèques comme PSIMJ [Garrido 01] et SSJ [L'Ecuyer et al 02] sont bien conçues et gratuites mais ne sont pas open source. SILK de Threadtec [Healy et Kilgore 97] et [Kilgore 00] est bien conçue mais ni open source ni gratuite.

Il existe aussi une large collection de librairies open source et nous pouvons considérer à titre d'exemple :

- JSIM [Miller et al 98] : un environnement de simulation et d'animation basé sur Java qui supporte des technologies web.
- JAVASIM [Little 99] : librairie développée à l'université de Newcastle upon Tyne pour le développement des simulations à évènements discrets orientées-processus similaire à celles de Simula et C++SIM.
- SIMJAVA [Howell et McNab 98] : une librairie de simulation à évènements discrets basée sur Java similaire à SIM++ de Jade avec un module d'animation graphique.
- JDISCO [Helsgaun 00] : une librairie de simulation à évènements discrets et continue.
- DESMO-J [Page et al 00] : une librairie qui supporte la simulation à évènements discrets par processus et par évènements.
- SIMKIT [Buss 02] : une librairie de simulation à évènements discrets par composants, influencé par MODSIM II et adopte l'approche dite graphe d'évènements.

JAPROSIM (**JA**va**PR**ocess **O**riented **SIM**ulation) est une librairie bien conçue, libre et open source qui adopte l'approche populaire orientée-processus. Sa conception est intuitive et facile à appréhender. Elle facilite la construction de simulations à évènements discrets pour des utilisateurs qui maitrisent Java ou même pour les experts en simulation ayant des connaissances élémentaires en programmation. Il faut noter aussi que JAPROSIM n'est pas une version Java d'une quelconque bibliothèque, comme c'est le cas de SIMJAVA et JAVASIM.

La bibliothèque JAPROSIM incruste en plus, un mécanisme caché pour la collecte automatique des statistiques. Cette technique permet une nette séparation entre l'implémentation de la dynamique du modèle et la collecte des informations sur celui-ci. Ainsi, les mesures de performances traditionnelles sont automatiquement calculées. Le modèle est développé sans se soucier des statistiques. Ainsi, aucun code pour la collecte des statistiques n'apparait dans le modèle de simulation, ce qui améliore certainement la lisibilité du code. Néanmoins, l'utilisateur possède toujours la possibilité, si nécessaire, d'implémenter des statistiques spécifiques utilisant les mécanismes offerts par cette bibliothèque dans le package statistics.

Cette caractéristique marque la différence entre JAPROSIM et les autres bibliothèques de simulation à évènements discrets écrites en Java. A l'exception de SIMKIT qui offre déjà un mécanisme similaire bien que celle-ci adopte une approche basée sur les graphes d'évènements.

5.5 Conception et paquetages

La bibliothèque (JAPROSIM) est écrite en Java et organisée actuellement en six paquetages :

- KERNEL : une collection de classes qui mettent en œuvre les entités actives, l'ordonnanceur, la queue et les ressources.
- RANDOM : un ensemble de classes pour la génération des flux de nombres aléatoires uniformes.
- DISTRIBUTIONS : un ensemble riche de classes pour les distributions de probabilités usuelles.
- STATISTICS : une collection de classes qui mettent en œuvre des variables statistiques intelligentes.
- GUI : un ensemble de classes pour la gestion de l'interface utilisateur, utile pour la paramétrisation du projet de simulation, ainsi que pour l'affichage de la trace et des résultants.
- UTILITIES : un ensemble de classes qui facilitent le développement rapide des modèles de simulation, en favorisant la réutilisation.

5.5.1 Le noyau

Un mécanisme très proche des coroutines a été mis en œuvre à travers les classes *SimProcess*, *Scheduler*, *StaticEnity* et *Entity*. Contrairement à JAVASIMULATION [Helsgaun 00] qui offre une classe Coroutine identique à celle qu'on trouve dans le langage SIMULA, notre objectif était de suivre étroitement la sémantique de SIMULA sans pour autant avoir la même conception. L'avantage est que la conception sera simple sans avoir

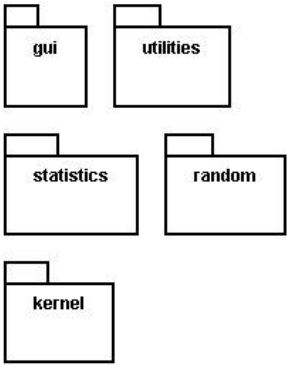


Figure 5.3: *Packages Japrosim.*

une implémentation explicite d'une classe pour coroutines. La sémantique de SIMULA est préservée ainsi que les mécanismes de bases bien connus et testés à fond depuis de longues années sont supportés.

Le support natif d'une exécution concurrente (multithread) est un aspect fondamental dans l'implémentation de l'approche par processus en Java. Le cycle de vie de chaque entité active est exécuté dans un thread séparé. Dans cette approche, les processus de simulation sont placés dans l'échéancier, dans un ordre chronologique croissant et gérés par l'ordonnanceur.

Les processus sont exécutés en pseudo-parallélisme et seulement un d'entre eux s'exécute en un instant donné du temps réel. Les processus de simulation peuvent s'exécuter en concurrence à n'importe quel instant dans le temps simulé. Par conséquent, l'ordonnanceur qui est un processus particulier de simulation s'exécute en alternance avec les autres processus. Ce comportement partagé est modélisé par la classe abstraite *SimProcess* qui hérite de la classe *Thread* de Java. D'une part, la méthode *processResume(Entity e)* est invoquée par l'ordonnanceur pour réactiver un processus de simulation et d'autre part, la méthode *mainResume()* est invoquée par un processus de simulation pour réactiver l'ordonnanceur. Chaque processus de simulation est doté d'un objet **verrou** nommé *lock* et l'ordonnanceur possède un objet verrou nommé *mainLock*. Ces verrous sont utilisés en combinaison avec les méthodes *wait()* et *notify()* offertes par la classe *Object* de Java pour synchroniser les processus de simulation (les threads qui les implémentent). Les verrous sont utilisés pour remédier au problème posé par les anciennes méthodes *suspend()* et *resume()* actuellement dépréciées. Un thread qui invoque l'une des méthodes *processResume(Entity e)* ou *mainResume()* se bloque sur son propre verrou après avoir notifié le thread approprié. La méthode synchronisée *schedule(Entity e)* de la classe *SimProcess* peut être invoquée par l'ordonnanceur ou bien un processus nouvellement crée pour l'insertion du processus en question dans l'échéancier.

A la fin de son cycle de vie, un processus de simulation invoque automatiquement la méthode *dispose()* pour réactiver l'ordonnanceur sans se bloquer. Ainsi, le thread correspondant peu se terminer normalement. Ceci permet de libérer la mémoire occupée par celui-ci et améliorer les performances de la simulation. Autrement, cela peut provoquer un usage excessif de l'espace mémoire comme nous l'avons décelé avec la librairie Silk [Bourouis et Belattar 06].

Le comportement spécifique de chaque processus de simulation doit être décrit par la méthode abstraite dédiée *body()* en la réécrivant. Le code sera une séquence ordonnée d'invocations de méthodes qui se termine implicitement par un appel automatique à la méthode *dispose()*. De la même manière, le cycle de vie de l'ordonnanceur est décrit par cette méthode.

Puisque la classe *SimProcess* est abstraite, elle est conçue pour être dérivé. Elle fournit les mécanismes de base nécessaires à la mise en œuvre de l'approche par processus. Les processus de simulation sont modélisés à travers la classe abstraite *Entity* qui hérite de *SimProcess*. Aucune instance ne peut être crée directement. L'utilisateur définit la classe qui correspond à un processus de simulation en héritant de la classe *Entity* et en décrivant le comportement dynamique, en particulier, par le biais des méthodes héritées de ces deux classes. Chaque classe dérivée de la classe *Entity* s'exécutera automatiquement dans un thread séparé. Une caractéristique

héritée indirectement de la classe *SimProcess*. La classe *Entity* fournit l'implémentation de la méthode *run()* qui à son tour invoque la méthode *body()*.

Cinq méthodes spéciales sont offertes, *insert()*, *remove()*, *seize()*, *hold()* et *release()*. Elles servent à modéliser les fameux scénarios des files d'attente. La méthode *passivate()* est utilisée pour modéliser les attentes conditionnées à un état particulier du système. L'invocation de cette méthode provoque la suspension du thread associé et son insertion dans la liste des processus passifs (PL :Passive List), cette invocation doit être placée dans une boucle *while()*. A chaque fois l'ordonnanceur prend le contrôle, il commence d'abord par la réactivation des threads suspendus dans la PL avant de passer au traitement de l'échéancier. Ainsi, le processus réactivé de la PL aura la possibilité de retourner à celle-ci s'il ne perçoit pas le changement d'état désiré.

La classe abstraite *StaticEntity* est utilisée pour modéliser les entités actives qui n'ont pas la capacité de déplacement. Une ressource intelligente est un exemple typique. La classe *StaticEntity* dérive directement de *SimProcess*. Puisque la classe *Entity* sert à modéliser les entités actives (capables de déplacer dans le système), elle dérive de *StaticEntity* et hérite de celle-ci les méthodes *seize()*, *hold()* et *release()*. En plus, la classe *Entity* offre deux autres méthodes qui sont *insert()* et *remove()*.

L'ordonnanceur procède en deux phases. Premièrement, il réactive tous les processus dans la PL. Les processus réactivés vérifient leurs conditions d'attente pour décider de continuer ou de retourner à l'attente. Deuxièmement, il réactive le processus dont la notice est dans l'en-tête de l'échéancier. Ces deux phases sont répétées tant la condition d'arrêt de l'expérience de simulation n'est pas vérifiée. Généralement, la condition d'arrêt est spécifiée sous forme d'une durée temporelle. La classe *Scheduler* qui implémente le comportement de l'ordonnanceur possède l'attribut *rng*, qui est une instance d'un générateur de nombres aléatoires uniformes. L'utilisateur peut spécifier un des générateurs offerts par la librairie JAPROSIM, sinon définir son propre générateur.

La classe *Resource* représente une entité passive caractérisée par une capacité. Généralement, un processus de simulation saisit quelques unités d'une ressource pour accomplir son service pour une certaine durée puis relâche ces unités à la fin du service. La méthode *hold()* offerte par la classe *StaticEntity* est utilisée pour spécifier la durée de service.

La classe *Queue* implémente un espace (limité ou non) réservé à l'attente. Normalement rattaché à une ressource, il offre une liste ordonnée où les entités peuvent résider. Typiquement, une entité est insérée dans la queue en invoquant sa méthode *insert (Queue q)*. Il n'existe pas une logique d'attente conditionnelle implicite associée à la classe *Queue*. Ainsi, le thread associé à une entité n'est pas suspendu automatiquement en attente d'un changement d'état du système en s'insérant dans une queue. Pour modéliser une attente conditionnelle, il faut se servir d'une boucle *while()* et la méthode *passivate()*. Par conséquent, une entité peut résider simultanément dans un nombre quelconque de queues. Cet avantage peut être particulièrement utile pour modéliser des cas réels complexes ou bien pour collecter des statistiques plus spécifiques. Une autre distinction importante est que le retrait d'une entité d'une queue, peut se faire de façon indépendante de l'ordre de celle-ci au moment de l'opération. L'utilisateur est obligé de spécifier explicitement l'entité à retirer. Typiquement, ceci est accomplit en invoquant la méthode *remove(Queue q)* de l'entité en question. Bien que les entités sont insérées et retirées des queues utilisant les méthodes *insert(Queue q)* et *remove(Queue q)* de la classe *Entity*, les mêmes opérations peuvent être accomplies utilisant les méthodes *insert(Entity e)* et *remove(Entity e)* de la classe *Queue*.

5.5.2 Le paquetage RANDOM

Les générateurs de nombres aléatoires sont à la base des outils de modélisation stochastique. Tout comme un artiste, un expert en modélisation doit connaitre bien ses outils. Les mauvais générateurs de nombres aléatoires peuvent ruiner la simulation et plusieurs pièges sont à éviter. Le paquetage *random* offre l'interface *RandomStream* qui est un cadre pour la création des générateurs de nombres aléatoires (RNG : Random Number Generators). Chaque *RNG* doit réécrire la méthode *randU01()* qui renvoie un nombre (Java double) uniformément distribué sur l'intervalle $[0, 1]$. Il est tout à fait possible d'utiliser la classe *java.util.Random* offerte par Java. Néanmoins, JAPROSIM offre déjà une collection riche de bons générateurs, les plus connus,

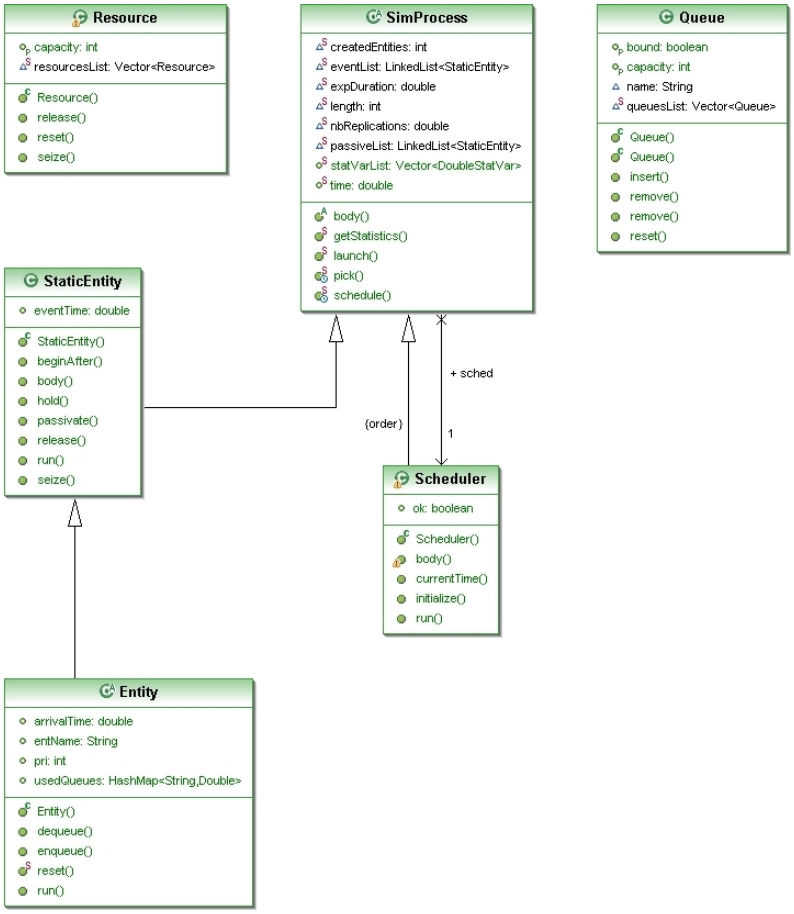


FIGURE 5.4: *Diagramme de classes du paquetage* KERNEL.

voir [L'Ecuyer 98] and [L'Ecuyer et Panneton 05], comme *Park-Miller*, *McLaren-Marsaglia* et *RandMrg* qui est un générateur combiné et récursif très puissant d'une période très longue proposé dans [L'Ecuyer 99].

La méthode *setSeed(long[] seed)* est utile pour spécifier les germes des *RNGs*, sinon des valeurs par défaut seront appliquées. L'utilisateur peut créer son propre *RNG* en implémentant l'interface *Randomstream*. Pour être utilisé dans JAPROSIM, une instance du générateur définit par l'utilisateur doit être affectée à l'attribut public et statique *rng* de la classe *Scheduler*.

Un autre ensemble riche de générateurs pour les distributions aléatoires discrètes et continues les plus utilisées est offert par le sous paquetage *distributions*. Cet ensemble couvre typiquement, la plupart des distributions de probabilité usuelles, mais l'utilisateur peut toujours l'enrichir encore en définissant de nouvelles distributions.

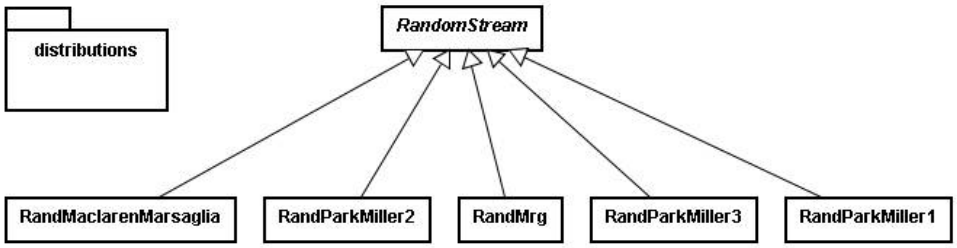


FIGURE 5.5: *Paquetage* RANDOM.

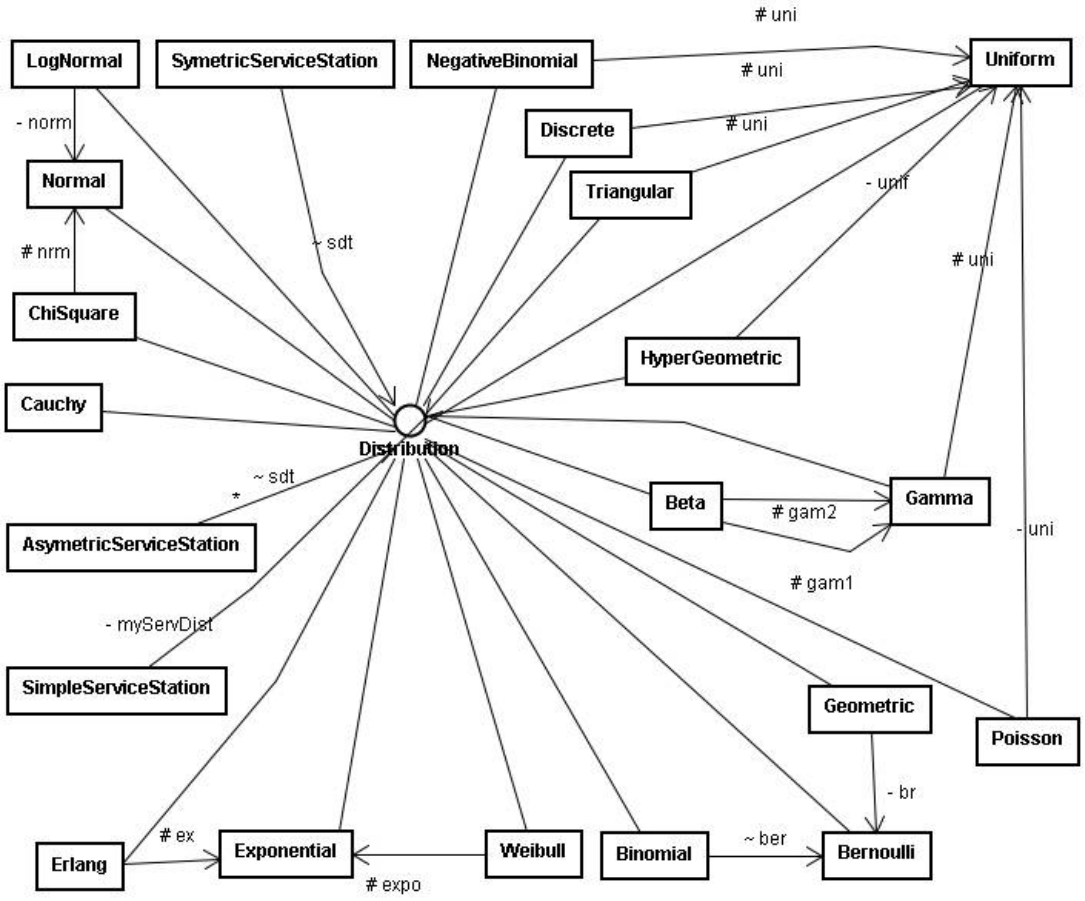


FIGURE 5.6: *Sous-paquetage* DISTRIBUTIONS.

5.5.3 Le paquetage STATISTICS

Le paquetage des statistiques fournit plusieurs classes dont deux sont très utiles. La classe *DoubleStatVar* qui traite les variables statistiques indépendantes du temps (valeurs doubles) comme le temps de réponse et le temps d'attente. Elle implémente le mécanisme qui permet de mémoriser les changements de valeurs en mettant à jour celle-ci utilisant la méthode *update()*. La classe *TimeIntStatVar* traite les variables dépendantes du temps (valeurs entières) tel que la longueur d'une queue ou le nombre de clients dans le système. Typiquement, l'utilisateur instancie la classe désirée, puis place les instances et les mettent à jours aux endroits adéquats dans le code.

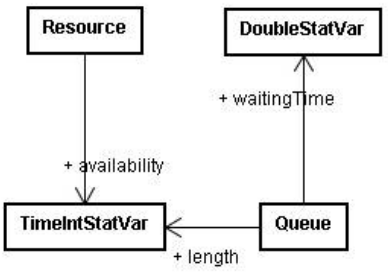


FIGURE 5.7: *Paquetage statistics.*

Le placement des variables statistiques et leurs mis à jours dans le code de simulation est une source de plusieurs pièges. Pour cette raison, JAPROSIM favorise la collecte automatique des statistiques usuelles que nous aborderons plus tard.

5.5.4 Le paquetage UTILITIES

Ce paquetage offre des entités spéciales avec des comportements spécifiques. Parmi les classes intéressantes, nous pouvons citer *SimpleServiceStation* permet de modéliser des serveurs intelligents (capables de prendre des décisions) comme les serveurs par lot (batch servers). La classe *SymetricServiceStation* modélise une station de services à plusieurs serveurs identiques, tandis que *AsymetrciServiceStation* permet de modéliser une station de services à plusieurs serveurs hétérogènes (peuvent avoir des distributions de services différentes).

5.6 Scénario simple de files d'attente

Pour démontrer les capacités de JAPROSIM, nous considérons le scénario simple de files d'attente suivant :

Le réseau contient deux stations de service, chacune possède une queue FIFO de capacité illimitée où les clients peuvent attendre pour être servis. Les clients arrivent de deux sources exogènes indépendantes et peuvent quitter le système depuis deux issus. Nous supposons que les arrivées externes suivent un processus de poisson et les services suivent la loi exponentielle. Les paramètres du modèle de simulation sont :

- $\gamma_1 = 3.57$ et $\gamma_2 = 4.82$ où γ_i est le taux d'arrivée exogène de la source numéro i (et le paramètre de la loi exponentielle du temps inter-arrivées).
- $\mu_1 = 4.15$ et $\mu_2 = 5.96$ où μ_i est le paramètre de la loi exponentielle pour un serveur de la station numéro i.
- $c_1 = 3$ et $c_2 = 2$ où c_i est le nombre de serveurs parallèles identiques de la station i.
- $r_{11} = 0.17$, $r_{12} = 0.33$, $r_{21} = 0.23$, $r_{22} = 0.18$ où r_{ij} est la probabilité qu'un client ayant fini son service à la station i se dirige vers la station j.

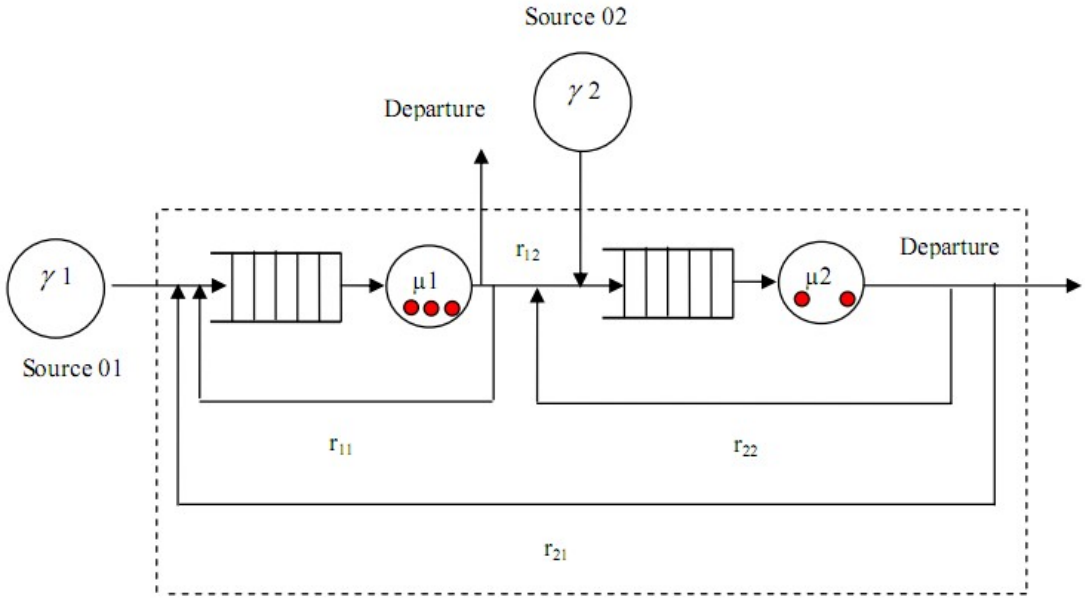


FIGURE 5.8: *Réseau ouvert de files d'attente.*

5.6.1 Solution analytique

C'est un réseau à classe unique (clients identiques) : $\begin{cases} \lambda_1 = \gamma_1 + r_{11}.\lambda_1 + r_{21}.\lambda_2 \\ \lambda_2 = \gamma_2 + r_{12}.\lambda_1 + r_{22}.\lambda_2 \end{cases}$

Où λ_i est le taux d'arrivée effectif à la station numéro i.

La solution du système d'équation précédent est λ_1 = 6.2041 and λ_2 = 6.8669.

Taux de charge des stations : ρ_1 =0.5360, ρ_2 =0.7184, qui signifie que les deux stations sont stables. Ainsi, le réseau entier est stable. Nous pouvons procéder au calcul des indices de performance.

Nous avons obtenu les résultats de la figure ci-dessous en utilisant l'outil RAQS développé à l'université d'Oklahoma (CCIM) qui utilise l'algorithme des deux moments.

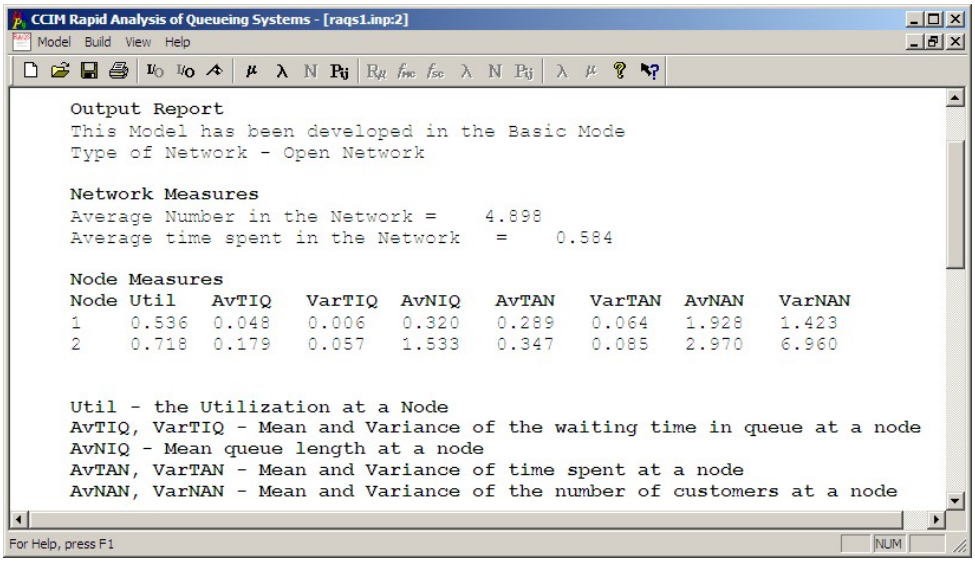


FIGURE 5.9: *Résultats obtenus utilisant RAQS [Bourouis et Belattar 08a].*

5.6.2 Solution par simulation utilisant Japrosim

Nous pouvons identifier dans le réseau de files d'attente deux ressources passives qui représentent les deux stations. La première ressource ayant une capacité de 3 et l'autre une capacité de 2. Puisque nous avons 2 entrées exogènes, nous devons faire la distinction entre deux entités actives ayant des cycles de vie un peu différents.

En Japrosim nous pouvons modéliser chaque entité active dans une classe séparée qui hérite de la classe *Entity*, comme nous pouvons considérer une seule classe dans laquelle nous faisons la distinction entre les deux entrées exogènes dans la méthode *body()* :

Listing 5.1: *La classe Transaction.*

```
01  import uoeb.japrosim.random.distributions.*;
02  import uoeb.japrosim.kernel.*;
03  public class Transaction extends Entity{
04    static Exponential arrival1 = new Exponential(3.57),
                         arrival2 = new Exponential(4.82),
                         serv1    = new Exponential(4.15),
                         serv2    = new Exponential(5.96);
05    static Queue queue1 = new Queue("Queue 01"),
                   queue2 = new Queue("Queue 02");
06    static Resource server1 = new Resource("Station 1",3),
                      server2 = new Resource("Station 2",2);
07    static Uniform selection = new Uniform(0.0, 1.0);
08    int trID;
09    double choice;
10    public Transaction(int id) {
11       trID = id;
      }
12    public void body(){
13         if(trID == 1){
14          new Transaction(1).beginAfter(arrival1.sample());
15              intoStation1();
           } else{
16          new Transaction(2).beginAfter(arrival2.sample());
17              intoStation2();
           }
      }
18    public void intoStation1(){
19         queue1.insert(this);
20         while(server1.getAvailability()<1){
21            passivate();
           }
22         seize(server1, 1);
23         queue1.remove(this);
24         hold(serv1.sample());
25         release(server1, 1);
26         choice = selection.sample();
27         if(choice<=0.17){ intoStation1(); }
28         else{if (choice<=0.5){ intoStation2(); }
           }
      }
29    public void intoStation2(){
30         queue2.insert(this);
31         while(server2.getAvailability()<1){
32             passivate();
           }
33          seize(server2, 1);
34          queue2.remove(this);
35          hold(serv2.sample());
36          release(server2, 1);
37          choice = selection.sample();
```

```
38              if(choice<=0.18){   intoStation2(); }
39              else{ if (choice<=0.41){ intoStation1(); }   }
        }
    }
```

Nous optons pour la deuxième alternative puisque les clients ont la même priorité. Nous n'avons besoin que d'une classe appelée *Transaction*. La structure de la classe consiste en une partie de déclaration (lignes 4 à 9) qui vont définir les caractéristiques des entités de la simulation. La méthode *body()* (lignes 12 à 17) permet de modifier les caractéristiques des entités au fur et à mesure que l'état du système change. Chaque instance de cette classe sera dotée d'un ensemble d'attributs (statiques) *arrival1* et *arrival2* pour les distributions d'arrivée, *serv1* et *serv2* pour les distributions de service, *queue1* et *queue2* pour les queues, ainsi que *server1* et *server2* représentant les stations de service (ressources). L'attribut *selection* représente une fonction de répartition que l'attribut *choice* est obtenu en appliquant la méthode de transformation inverse pour savoir la destination d'un client (selon les probabilités de routage). Dans la méthode *body()*, la distinction entre les sources des clients selon la valeur de l'attribut *trID*. Les méthodes *intoStation1()* et *intoStation2()* modélisent le comportement du client dans la station correspondante.

Dans cet exemple, chaque client qui rentre dans le système, crée l'arrivée de son successeur utilisant une instance du générateur de nombres aléatoires exponentiels (lignes 14 à 16). La méthode *hold()* sert à modéliser le service et son paramètre permet de spécifier sa durée (lignes 24 à 35). Pour modéliser une attente conditionnelle la méthode *passivate()* est utilisée dans une boucle *while()*, ce qui permet d'assurer l'attente jusqu'à ce que la condition devienne fausse (lignes 20, 21, 31 et 32).

Pour l'exécution d'un modèle de simulation en JAPROSIM, il est nécessaire de prévoir une classe particulière qui contiendra la méthode *main()* pour les applications autonomes et la méthode *init()* pour les applets. C'est l'endroit pour initialiser le modèle et démarrer l'ordonnanceur. Dans cet exemple, la classe est nommée *OpenNetwork* :

Listing 5.2: *La classe OpenNetwork.*

```
1 public class OpenNetwork {
2   public static void main(String[] args) {
3     new Transaction(1).beginAfter(0.0);
4     new Transaction(2).beginAfter(0.0);
5         SimProcess.sched.time = 0.0;
6         SimProcess.sched.start();
7   }
8 }
```

Au lancement de l'application, une fenêtre apparait. Il s'agit de la fenêtre principale de JAPROSIM qui permet de paramétrer le modèle de simulation. Les paramètres tels que le nombre de réplications, la durée de l'expérience de simulation et le *RNG* utilisé peuvent être fixés. Un bouton Run/Stop/Resume permet de lancer, interrompre et reprendre la simulation. Deux autres boutons servent à la présentation des résultats et de la trace de simulation. L'exemple précédent a été exécuté durant 35000 unités de temps. La précision des résultats peut être comparée à celle obtenues par des méthodes analytiques comme RAQS.

5.7 La collecte automatique des statistiques

Il est clair dans le code source de l'exemple précédent qu'aucune référence explicite aux classes du paquetage STATISTICS n'est utilisée. En plus, aucune structure de données Java n'est utilisée pour le calcul des statistiques. Une caractéristique clé de JAPROSIM réside dans le calcul implicite et automatique des statistiques usuelles. Ainsi, résulte la facilité d'utilisation de cette bibliothèque traduite par le confort de l'utilisateur dans le codage du modèle conceptuel. L'utilisateur ne s'inquiète plus du calcul des statistiques usuelles. Aucun effort de sa part n'est nécessaire pour la déclaration, le placement et la mise à jour des variables statistiques. Une utilisation des variables statistiques par des utilisateurs avec peu de connaissances en programmation ou en statistiques

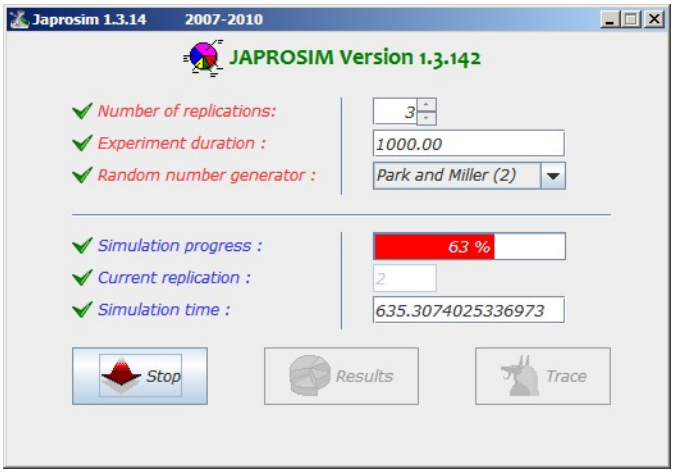


FIGURE 5.10: *Fenêtre principale de* JAPROSIM.

peut conduire à des erreurs indétectables et peut même ruiner la simulation puisque l'exactitude des résultats est cruciale. Donc, JAPROSIM facilite le codage des modèles conceptuels en modèles de simulation de façon plus sûre, particulièrement pour ceux n'ayant pas des compétences de haut niveau en programmation Java ou en statistiques.

Ce mécanisme est emboité dans la bibliothèque. La classe *SimProcess* possède un attribut *entitiesList* protégé et statique, qui est une *HashMap* Java pour collecter le temps de séjour dans le système de chaque entité de simulation. Le constructeur de la classe *Entity* insère automatiquement chaque nouvelle classe d'entités crée dans la liste précédente. Dans la méthode *run()* de la classe *Entity*, et après l'invocation de la méthode *body()*, le temps de résidence est mis à jour en utilisant le temps actuel de simulation est la date d'arrivée dans le système de l'entité en question. Dans l'exemple précédent, si 02 classes distinctes ont été utilisées pour les clients, nous obtenons deux entrées différentes dans *entitiesList*. En plus, chaque instance d'une classe dérivée d'*Entity* possède une liste (*Java HashMap*) pour enregistrer les queues qu'elle traverse. De cette façon, les invocations des méthodes *insert()* et *remove()* mettent à jour automatiquement le temps d'attente dans chacune des queues.

Chaque instance de la classe *Queue* possède une variable statistique pour garder trace du temps d'attente dans ceci. Cette variable est mis à jour également par les méthodes *insert()* et *remove()*. Le nombre d'entités dans une queue est une autre variable statistique dépendante du temps. La classe *Queue* possède un attribut statique (*Java Vector*) pour enregistrer tous les queues utilisés dans le système. De la même manière, la classe *Resource* possède une liste similaire pour garder trace de toutes les ressources du système. Ces listes ont une visibilité *package*, ce qui permet à tous les processus de simulation d'y accéder. La disponibilité d'une ressource est gérée par une variable dépendante du temps possédée par chaque instance d'une classe *Resource*.

Néanmoins, l'utilisateur est libre. Il peut créer, placer et mettre à jours des variables statistiques de out genre dans le code source du modèle de simulation. Il existe des systèmes complexes qui nécessitent des statistiques particulières (non usuelles) non prises en compte par JAPROSIM.

5.8 Conclusion

Nous avons présenté dans ce chapitre les différentes étapes du projet JAPROSIM basé sur une bibliothèque de simulation à évènements discrets par interaction de processus. Le projet vise à développer un environnement de modélisation et de simulation intégré. Un modèle conceptuel conçu graphiquement, dans le formalisme proposé (EQNM^{2}L), permet l'interopérabilité grâce à un format d'échange standard basé sur XML. Ce modèle peut être résolu analytiquement ou bien simulé grâce à la transformation de modèles permettant d'obtenir le modèle équivalent pour chaque outil cible. La génération du code de simulation, la présentation et l'explication des résultats de simulation est offerte aussi.

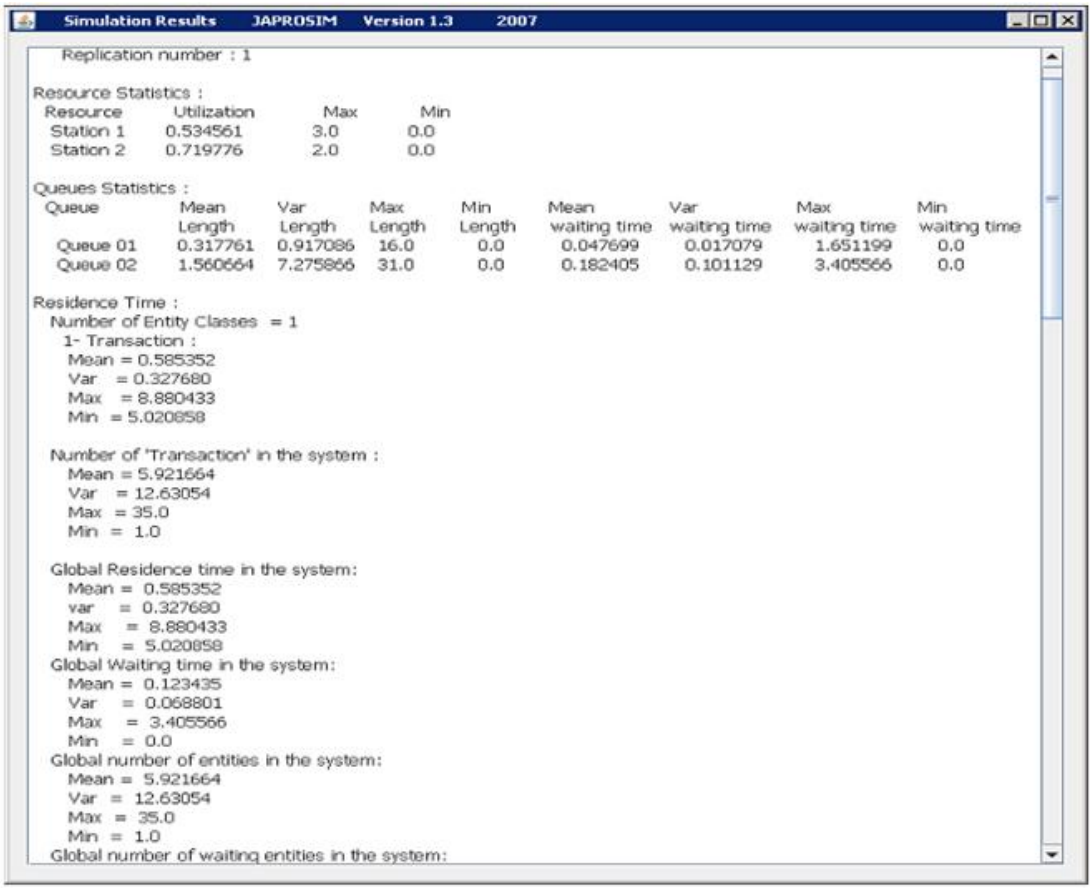


FIGURE 5.11: *Résultats de la simulation [Bourouis et Belattar 08a].*

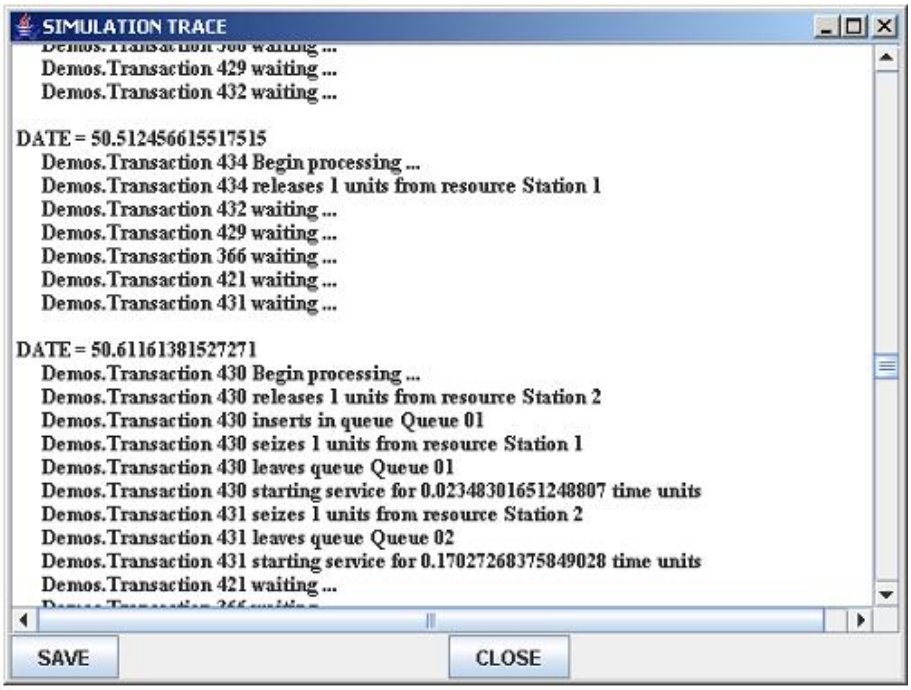


FIGURE 5.12: *trace de la simulation [Bourouis et Belattar 08a].*

La bibliothèque constitue le niveau d'abstraction le plus bas dans une architecture DSM. Elle représente le niveau des modèles de simulation programmés obtenus après translation du modèle conceptuel. Japrosim est assez mûre, mais évolue en continu, extensible et facile à utiliser même par les utilisateurs non expérimentés en programmation. Un mécanisme de collecte automatique et implicite de statistiques les plus utilisées est offerte, ce qui minimise les erreurs de programmation.

Chapitre 6

Extended Queueing Networks Modeling and Markup Language (EQNM^{2}L)

Sommaire

6.1 Introduction

Prédire le comportement ou les performances est fondamentale pour les décideurs dans le domaine de la conception et de l'analyses des systèmes. Les analyses qualitative et quantitative sont d'une grande

importance. Les réseaux de files d'attentes est un formalisme mathématique qui permet de modéliser un large éventail de systèmes et phénomènes discrets réels. Les modèles de files d'attente se prêtent bien à l'évaluation quantitative où les solutions peuvent être obtenues analytiquement ou par simulation. Nous allons dans la suite développer un DSML visuel dans un cadre formel qui permet de modéliser des réseaux de files d'attentes complexes. Durant ce processus, nous essayons de suivre les différentes phases de développement citées dans la section 2.5.1.

Ce formalisme diagrammatique, inspiré des concept-maps, trouvera toute son utilité à travers son format d'échange basée sur XML pour favoriser l'interopérabilité. En se basant sur les principes de l'IDM, les métamodèles et les modèles conceptuels qui servent toujours pour la communication et la documentation, deviendront productifs en donnant les codes complets pour l'environnement de modélisation et les simulateurs associés.

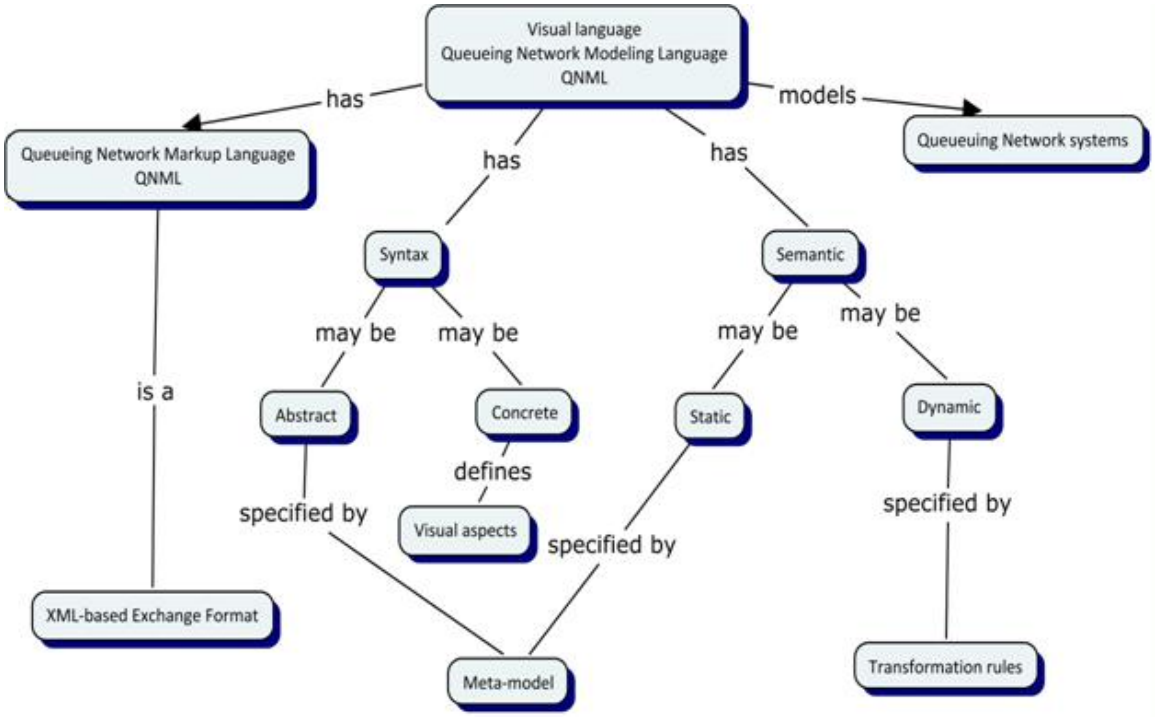


Figure 6.1: *Concept map sur EQNM^{2}L.*

6.2 Phase de décision

La description d'un réseau de files d'attente peut se faire sous forme graphique et/ou textuelle, mais il n'y a pas un langage standard pour ce faire. En plus, le langage de description doit permettre un traitement automatique et une manipulation et compréhension par l'utilisateur. Généralement, le développement d'un tel langage est un vrai défi. Le formalisme de réseaux de Petri utilisé notamment pour l'analyse qualitative des systèmes, possède déjà un formalisme visuel plus ou moins standard. Bien qu'il existe des variantes dans ce formalisme, un format d'échange standard (iso) basé sur XML nommé Petri Net Markup Language (PNML) a été développé [Billington et al 03]. Le formalisme de réseaux de files d'attentes mérite aussi une telle démarche qui renforce l'interopérabilité à travers un format d'échange standard entre les différents outils d'analyse et de simulation.

Il est important de noter ici les travaux de l'équipe du « Center for Computer Integrated Manufacturing Enterprises (CCiMe) » de l'université d'Oklahoma sur un langage nommé QNML aussi [Chalavadi 04]. Bien que nous partageons certains objectifs, mais ces travaux se sont concentré uniquement sur la résolution analytique des modèles et non sur leur simulation. En plus, les concepts sur lesquels est fondé ce formalisme sont influencés par l'outil RAQS développé par la même équipe, ce qui s'est répercuté directement sur sa capacité

d'expression. La figure 6.2 représente le métamodèle proposé. Notre objectif est de concevoir un formalisme suffisamment expressif sans se soucier des limitations des outils de résolution analytique ou même de certains logiciels de simulation.

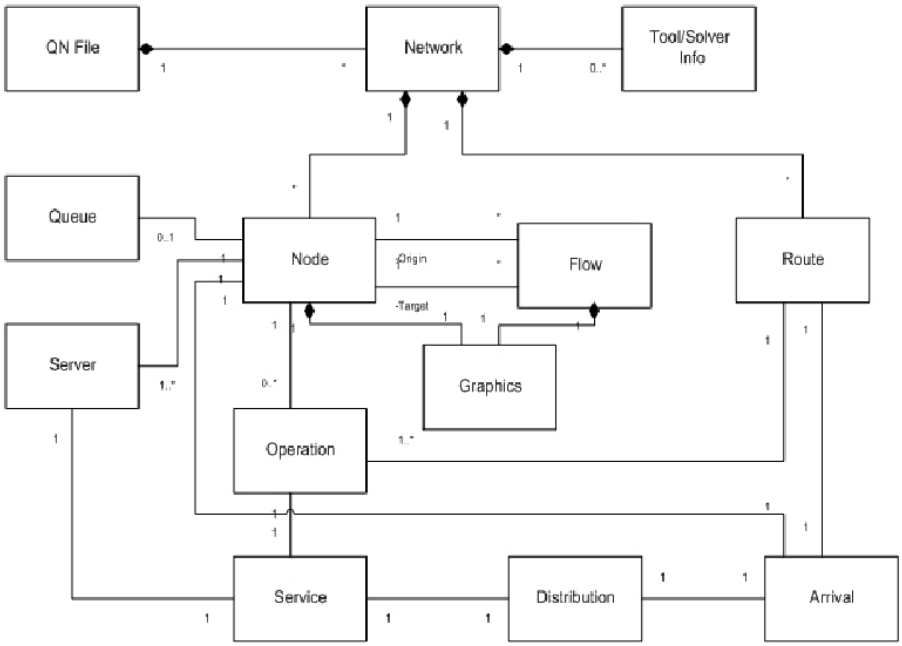


FIGURE 6.2: *Métamodèle de QNML [Chalavadi 04].*

6.3 Phase d'analyse : notions de base

Les modèles des réseaux de files d'attente ont été largement utilisés comme un formalisme pour obtenir des mesures de performances basées sur des méthodes analytiques ou sur la simulation. Ils permettent de modéliser une grande variété de systèmes à évènements discrets. La notation de *Kendall* reste un moyen très répandu pour spécifier et caractériser de telles modèles, en particulier, pour les systèmes simples. Pour des systèmes complexes, cette notation n'est pas assez expressive pour les décrire. Une notation graphique avec des annotations textuelles est nécessaire. Pour manipuler ces modèles, les résoudre ou les simuler par différents outils, ils sont codés et sauvegardés dans un format, généralement propriétaire. Ainsi, les modèles ne peuvent être échangés ni réutilisé.

Les notions de bases discutées ici sont basées sur [Bolsh et al 06] et [Lazowska et al 84] où les réseaux de files d'attentes sont considérés comme un ensemble de stations de services visitées par des tâches (jobs). Des termes comme *transaction*, *client* et *tâche* réfèrent tous au même concept qui caractérise une entité dynamique capable de se déplacer dans le système d'une station à l'autre pour acquérir des services.

6.3.1 Classes et priorités

6.3.1.1 Classes

Dans un réseau de files d'attente, les tâches qui se présentent peuvent appartenir à des classes distinctes. Ces classes peuvent se différencier par leurs distributions d'arrivées et/ou de service, le nombre d'unités de stations passives demandé et dans leurs probabilités de routage. Ainsi, différentes classes se traduit par différents cycles de vies. Il est à noter qu'il est tout à fait possible qu'une tâche change de classe d'appartenance durant son cycle de vie.

6.3.1.2 Priorités

Un mécanisme de priorités indépendant de la classe d'appartenance est considéré pour organiser la totalité de la population des tâches. Ainsi, les tâches peuvent être servies selon leurs priorités. Il est nécessaire de prévoir une autre discipline de service au sein d'une même catégorie de priorité pour faire la distinction entre les tâches ayant la même priorité. La priorité d'une tâche peut être statique (fixe) ou dynamique qui change avec le temps (en fonction du temps) sous la forme Pr(t) ou en fonction de l'état du système (de certains paramètres du système). Il est aussi possible de spécifier que les priorités sont considérées dans un ordre croissant (High Value First HVF) ou bien dans un ordre décroissant (Low Value First LVF).

Il est possible d'affecter une priorité pour chaque classe (class-dependent priorities), mais généralement, au sein d'une même classe il peut y avoir plusieurs catégories de priorités.

6.3.2 Stations de service

Les *stations*, *centres de service* ou *nœuds* réfèrent au même concept. Chaque station est distinguée par un identificateur unique et caractérisée par un certain nombre de serveurs parallèles, une queue d'une capacité limitée ou non pour l'attente, une ou plusieurs disciplines de service indépendantes des classes et une distribution de service pour chaque classe. Les taux de service sont souvent statiques, mais ils peuvent être dynamiques. Si la discipline de service utilisée est Round Robin, alors la tranche temporelle (slice) doit être spécifiée.

6.3.2.1 Station asymétrique

Ce genre de stations est caractérisé par des serveurs parallèles non identiques, dans le sens d'avoir différentes distributions de service. Dans la réalité, ceci modélise par exemple des machines parallèles de différents âges ou provenant de constructeurs différents. Ce genre de stations nécessite plus d'informations sur l'affectation d'une tâche à un serveur particulier. L'affectation se fait selon un critère comme :

- Aléatoire : la tâche est affectée à un serveur libre au hasard.
- Le plus rapide : la tâche est affectée au plus rapide parmi les serveurs libres (le plus petit temps moyen de service calculé depuis l'historique).
- Le plus inactif : la tâche est affectée au serveur qui a resté plus en chômage parmi les serveurs libres.
- Le moins chargé : la tâche est affectée au serveur qui a un taux d'occupation le plus faible parmi les serveurs libres.

Il est tout à fait possible de définir d'autres stratégies d'affectation. En plus, parfois le processus d'affectation ne permet pas de distinguer un serveur unique et le recours à un autre critère est nécessaire[1].

6.3.2.2 Station Fork/Join

Un type particulier de stations capable de décomposer une tâche en un ensemble de sous tâches (station *Fork*) ou de recombiner un certain nombre de sous tâches en une seule tâche (station *Join*). La synchronisation peut être nécessaire pour accomplir cette opération dans le sens où les même sous-tâches issues de la décomposition d'une tâche particulière doivent être recombinées. Les sous-tâches sont **généralement** considérées comme appartenir à une nouvelle classe, du fait que les durées de service sont différentes de celles des tâches d'origine.

Généralement la décomposition et la recombinaison sont des opérations instantanées. Néanmoins, il est tout à fait possible de considérer que celles-ci peuvent consommer certaines durées temporelles spécifiées utilisant une distribution de probabilité.

1. Dans ce cas, généralement l'affectation sera aléatoire.

6.3.2.3 Possession simultanée de ressources

Si on considère un système informatique, une tâche peut utiliser plusieurs ressources en même temps. Ici on s'intéresse aux ressources passives qui sont des stations simples avec un certain nombre de jetons (unités identiques) à louer. Aucun autre service ni offert par ces stations. Alors, la durée de l'allocation/libération est supposée nulle.

Une tâche qui se présente à une station passive d'allocation, demande un certain nombre de ressources (jetons). Si elle obtient les jetons demandés, elle peut progresser vers les autres nœuds du réseau, autrement elle doit attendre à les posséder. Une fois la tâche arrive à une station passive de libération (relâchement), elle libère tout ou une partie des jetons déjà acquis et qui deviendront alors disponibles pour les autres tâches. Il est à noter que chaque classe de tâches peut nécessiter un nombre différent d'unités. Ainsi, une station de libération doit faire référence à sa station de réservation [Bolsh et al 06].

6.3.2.4 Choix

Une tâche, durant son cycle de vie, peut avoir à prendre des décisions en fonction de l'état du système. Un système à évènements discrets est décrit par un ensemble de variables d'état. Ainsi, une structure de choix peut être assimilée à une station de service dans laquelle la tâche prend une décision. La décision est prise en fonction d'une expression booléenne contenant une ou plusieurs variables d'état du systèmes parmi celles prédéfinies[2] ou bien définies par l'utilisateur[3]. L'évaluation de l'expression booléenne peut avoir seulement deux valeurs possibles ? Ainsi, une structure de décision aura seulement deux issus. La tâche est dirigée ensuite selon le résultat de l'évaluation.

6.3.3 Types de service

6.3.3.1 Traitement par lot

Le traitement par batch est parfois nécessaire pour modéliser certains systèmes complexes. Un batch est caractérisé par une taille minimale et une autre maximale. Dans ce cas, le batch entier est traité en une seule fois comme étant une tâche unique. Il est clair que le batch est homogène dans le sens où les tâches qui le composent appartiennent à la même classe.

6.3.3.2 Préemption

Certaines disciplines de service peuvent provoquer une préemption de la tâche en cours de traitement si une nouvelle tâche plus prioritaire se présente. La tâche préemptée retourne à la queue et peut reprendre son service ultérieurement soit dès le début (pre-emption without resume) ou bien depuis le point d'interruption (pre-emptive resume). Par exemple, dans le cas des disciplines LIFO ou Priority, la préemption peut être activée ou désactivée.

6.3.3.3 Blocage

Le blocage dans une station peut se produire lorsqu'elle a une queue de capacité limitée et que celle-ci est atteinte. Plusieurs modèles de blocage sont distingués :

- ✍ Rejet : la tâche bloquée est forcée à quitter le système.
- ✍ Blocage après service : la tâche bloquée est forcée à attendre dans la station d'origine jusqu'à ce que la station de destination soit capable de l'accepter. Durant le blocage, la station d'origine cesse de fonctionner (occupée) et n'offre aucun service.

2. Les mesures de statistiques classiques pour chacune des stations du système ainsi que les variables d'état usuelles, comme la longueur d'une file d'attente …etc.

3. Des variables de différents types définies et mises à jour par l'utilisateur.

- ✍ Service répétitif : la tâche bloquée est forcée à répéter son service jusqu'à ce que la station de destination soit prête à la recevoir. Elle peut aussi choisir une autre destination.
- ✍ Queue d'attente : dans ce mode la tâche bloquée est insérée dans une queue d'attente temporaire jusqu'à ce que la station de destination soit en mesure de l'accepter.

6.3.3.4 Service dépendant de la charge

Dans des situations particulières, le service peut dépendre de la charge de la station exprimée par le nombre de tâches dans la station. Ainsi, pour chaque rang (fourchette) de charge de la station, une distribution de service est indiquée.

6.3.3.5 Pannes des serveurs

Dans la pratique, les systèmes réels ne sont pas toujours fiables. Les serveurs peuvent devenir indisponibles durant une certaine période pour maintes raisons (panne, maintenance, repos, …etc). La durée d'absence est spécifiée par une distribution de probabilité. Si les serveurs sont identiques, leur panne est décrite par une seule distribution, sinon autant de distributions que de serveurs sont nécessaires. L'indisponibilité peut être synchrone dans le sens où tous les serveurs sont concernés au même moment ou bien asynchrone s'ils cessent le service indépendamment l'un de l'autre.

6.3.4 Arrivées (Entrées)

Une station peut ou non recevoir des tâches en provenance de l'extérieur du système (entrée de tâches exogènes), mais doit impérativement avoir une sortie (départ) vers une autre station ou vers l'extérieur du système.

Une entrée exogène est caractérisée par un identifiant unique, associée à un flux de tâches avec une distribution de probabilité, une classe et une priorité pour bien décrire le processus d'arrivée. Il est aussi possible de spécifier des arrivées par batch en indiquant la distribution de la taille du batch.

6.3.5 Départs (Sorties)

Chaque station possède une ou plusieurs sorties. Une stratégie de routage doit être définie pour chaque classe de tâches. Une sortie est identifiée par une référence à la station de destination, associée à une classe de tâches et des informations de routage selon la stratégie adoptée.

6.3.6 Stratégie de routage

Après l'acquisition d'un service, une tâche d'une classe donnée se dirige vers une destination particulière selon une stratégie de routage. Une liste non exhaustive de stratégies est donnée ici :

- ✍ Probabiliste : la tâche se déplace vers une station selon une probabilité donnée.
- ✍ Round Robin : toutes les destinations possibles pour une classe de tâches sont choisies de façon cyclique.
- ✍ La plus courte queue : la tâche se dirige vers la queue la plus courte parmi un ensemble de destinations possibles.
- ✍ Le plus petit temps de réponse : la tâche se dirige vers la station ayant le plus petit temps de réponse parmi un ensemble de destinations possibles.
- ✍ Le plus plus petit taux d'utilisation : la tâche se dirige vers la station ayant le plus petit taux de charge parmi un ensemble de destinations possibles.
- ✍ Le service le plus rapide : la tâche se dirige vers la station ayant le plus petit temps de service moyen parmi un ensemble de destinations possibles.

6.3.7 Région à capacité limitée

Une région à capacité limitée (finie) généralise la notion de station à capacité finie. Elle consiste en un ensemble de stations qui possède une capacité limite à ne pas dépasser au niveau global. Ainsi, la limitation ne concerne pas une station particulière, mais une ensemble dont le nombre de tâches qui coexiste est contrôlé. Il est possible de définir deux types de ces régions :

- Partagée : le nombre limite de tâches est pris sans considérer la classe d'appartenance de ces dernières.
- Dédiée : le nombre limite de tâches ne concerne qu'une classe particulière.

6.4 Phase de conception

6.4.1 Le métamodèle

La méta-modélisation est le processus qui permet de spécifier explicitement la syntaxe abstraite d'un formalisme, ainsi que sa sémantique statique [Vangheluwe et De Lara 03]. Le formalisme dans lequel est exprimé ce métamodèle doit être suffisamment expressif pour pouvoir spécifier le langage considéré. Généralement, le formalisme d'entité-association ou le diagramme de classes d'UML en conjonction avec le langage de contraintes OCL sont deux moyens très puissants pour ce faire. Les outils actuels offrent parfois des formalismes dérivés qui sont adaptés à leurs capacités.

La figure 6.3 représente le métamodèle du langage EQNM²L exprimé en diagramme de classes UML. Les contraintes sémantiques peuvent être spécifiées utilisant le langage Object Constraint Language (OCL).

Un fichier EQNM²L ne peut contenir qu'un modèle qui représente un seul réseau de files d'attente *Qnet*. Le modèle contient une description *Desc* composée de plusieurs métadonnées *MetaData* (concernant l'outil utilisé, sa nature, sa version, l'auteur, …etc). Chaque modèle comporte plusieurs nœuds *Node* qui se distinguent par un identificateur unique (au niveau du modèle).

L'interface *Node* modélise le concept d'un nœud caractérisé par des disciplines de service, une capacité de la queue rattachée à celui-ci et un modèle de blocage. Par défaut, la discipline=« FIFO », capacité = $+\infty$ (représenté par un entier négatif) et blocking = « Reject ».

Pour représenter un flux de tâches qui se dirige d'une station de service vers une autre, une classe *Output* sous forme d'association est utilisée. Chaque flux est caractérisé par un numéro de classe et éventuellement une probabilité (si le routage est probabiliste). Il est clair d'après les cardinalités qu'un nœud peut recevoir plusieurs flux de tâches et dirige plusieurs autres vers les autres nœuds. Un nœud comporte plusieurs stratégies de routage ce que reflète la classe *Routing*. Pour chaque classe de tâches susceptible de passer par le nœud, la stratégie de routage doit être indiquée. Le numéro de classe « 0 » représente toutes les classes non explicitement indiquées.

Une réalisation plus concrète d'un nœud est la classe *ActiveStation*. Elle est caractérisée par un certain nombre de serveurs parallèles identiques *nbServers*. La *préemption* qui est désactivée par défaut et une valeur *slice* utile uniquement si la discipline est Round Robin. Du fait que les serveurs sont identiques, si leur panne est pris en compte, la classe *MTBF-MTTR* sert à la modéliser et deux distributions de probabilité permettent de spécifier le temps moyen avant une panne (temps moyen de fonctionnement) et le temps moyen de réparation. Chaque classe de tâches peut nécessiter un service particulier dans le sens où celui-ci peut nécessiter un certain nombre de serveurs *units* ce qui est réalisé par la classe *Service*. Par défaut, classe = « 0 », units = « 1 ». Le service peut être par batch selon une distribution particulière, reflété par le fait que la classe *Service* comporte une classe *Batch* qui comporte elle même une classe *Distribution*.

Chaque flux de tâches qui rentre dans le réseau se dirige vers un nœud. La classe-association *InputToNode* modélise ce concept. Chaque flux dirigé vers un nœud est caractérisé par un identifiant unique et concerne une classe particulière de tâches avec une priorité donnée. La classe *Batch* modélise les arrivées groupées où les tailles *min* et *max* sont spécifiées ainsi que la distribution de probabilité qui gouverne les arrivées.

La classe *AsymmetricStation* représente une station de service à plusieurs serveurs parallèles mais pas forcément

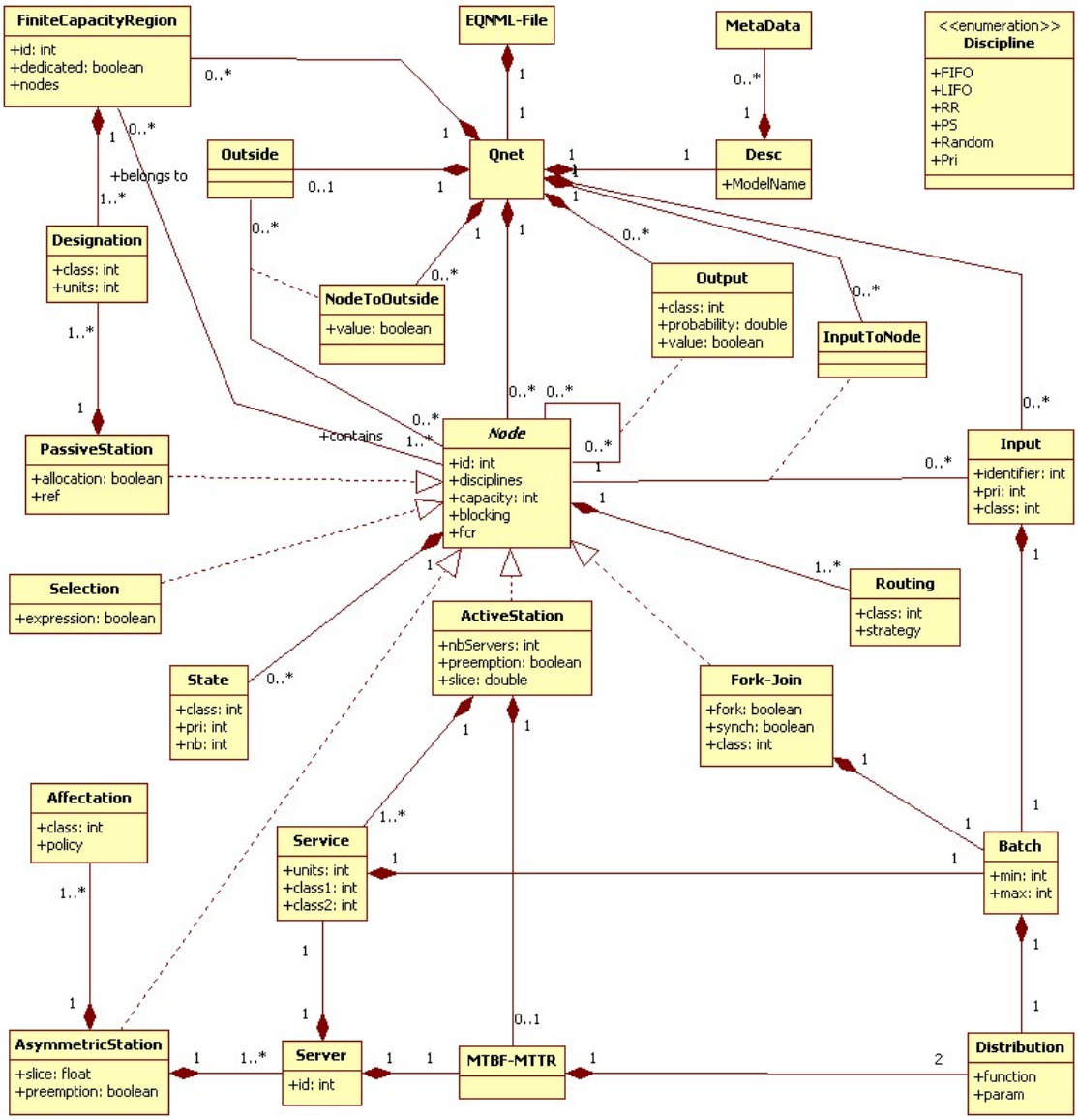


FIGURE 6.3: *métamodèle EQNM²L en diagramme de classes UML.*

identiques. Chacun des serveurs est une instance de la classe *Server* qui possède un identificateur unique (au sein du noeud) et peut se caractériser par une panne particulière. Chaque classe de tâches peut avoir son propre processus d'affectation à l'un des serveurs ce qui réalisé par la classe *Affectation*. Le numéro de classe « 0 » est utilisé pour désigner toute classe non explicitement spécifiée.

Pour les stations Fork/Join, la classe *Batch* représente le nombre de sous-taches à produire ou à combiner. Batch.min = Batch.max est une contrainte à respecter dans ce cas. L'opération fragmentation/recombinaison ne consomme, généralement, pas de temps. Néanmoins, il est possible de spécifier une durée temporelle à consommer utilisant la classe *Distribution*.

L'attribut *class* représente la classe prise en compte par la station. Seulement les tâches portant ce numéro subiront une fragmentation ou une recombinaison. En cas de station *Join* synchronisée (reliée normalement à une station *Fork*), seules les mêmes sous-tâches issues de la fragmentation préalable d'une tâche particulière seront combinées. Le numéro de la classe de la tâche d'origine et son identifiant doivent être préservés par les sous-taches pour permettre la recombinaison.

Le problème qui se pose au niveau des stations *Join* réside dans la synchronisation. L'attente de toutes les sous-tâches issues de la décomposition d'une tâche particulière peut engorger la station en question par des sous-tâches issues de différentes décompositions. Même si cette opération de recombinaison ne consomme pas du temps. La capacité de la station, si elle est limitée peut poser des problèmes et faire intervenir des mécanismes de blocage. En plus, cela peut introduire des délais de retard (Delay) non négligeables. Il faut bien veiller au placement de ce type de stations, pour remédier à ce type de problèmes.

D'après le métamodèle, la classe UML *Selection* implémente l'interface *Node* mais seule l'attribut *id* est signifiatif. Un choix se fait instantanément et les attributs hérités *disciplines* et *capacity* n'ont pas d'importance. De même, l'attribut *blocking* ne sert à rien parce que le blocage ne concerne que la station source. L'attribut *value* des classes *Output* et *NodeToOutput* prend son intérêt pour ce type de structure. Il est insignifiant pour les autres.

Du fait que le réseau de file d'attente possède un état initial et que celui-ci n'est pas toujours celui où tous les nœuds sont vides. La classe *State*, permet de spécifier le numéro de classe, la priorité et le nombre de tâches détenues par un nœud à l'état initial.

La classe *Discipline* comporte les principales disciplines de service retenues, mais la liste est non exhaustive et extensible. Chaque classe *Node* possède un attribut *disciplines* qui est une liste de disciplines de service à appliquer dans l'ordre indiqué. Du fait que certaines disciplines ne suffisent pas seules dans certains cas, la discipline suivante dans la liste est appliquée en cas d'ambiguïté si elle est indiquée, sinon ça sera la discipline *Random* par défaut.

La classe *FiniteCapacityRegion* permet de mettre en œuvre un mécanisme de contrôle sur le nombre de tâches qui résident dans une région. La région est un ensemble fini de nœuds et un nœud peut appartenir à plusieurs régions. Deux types de régions existent. La première est dite **partagée** dans le sens où le nombre de tâches est pris quelque soit la classe d'appartenance. La deuxième est dite **dédiée** en focalisant sur des classes de tâches particulières . La classe *Designation* permet de spécifier le numéro de classe concerné et le nombre de tâches maximum où le numéro « 0 » est interdit. Pour une région partagée le numéro de classe n'est pas significatif.

La classe *PassiveStation* en conjonction avec la classe *Designation* permet de mettre en œuvre le mécanisme d'allocation/libération de ressources. Chaque station de libération fait référence à une station d'allocation pour permettre de réquisitionner les jetons libérés. Chacune de ces stations possède les informations nécessaire à l'allocation/libération de jetons pour chacune des classes pris en compte. Le numéro de classe « 0 » désigne toute autre classe non explicitement spécifiée.

6.4.2 Sémantique statique

La sémantique statique fait partie du métamodèle et consiste en un ensemble de règles assurant qu'il est bien formé (well-formedness rules). L'ensemble des règles permet d'exprimer des contraintes et ainsi de réduire l'ensemble des modèles valides. Nous allons présenter ces contraintes en langage naturel tout en essayant d'être

concis et clairs. Par la suite, ces mêmes règles vont être exprimées en OCL durant la phase d'implémentation en respect à l'environnement de mise en œuvre choisi. En plus des contraintes de cardinalité déjà présentes dans le diagramme de classe, les autres règles sont :

1. Tous les identifiants *id* doivent être des entiers strictement positifs ($id > 0$).
2. L'identifiant d'un nœud est unique dans le modèle.
3. Un flux de clients de même numéro de classe et priorité ne peut être dirigé plus d'une fois d'un nœud vers un autre ou vers l'extérieur (Outside).
4. Pour une classe de tâches, le réseau de files d'attente peut être soit ouvert soit fermé.
5. Les entrées vers une même station doivent avoir des identifiants distincts.
6. Une probabilité *pr* est toujours un nombre réel compris entre 0 et 1 ($0 \leq pr \leq 1$).
7. Si le routage dans une station est probabiliste pour une classe de tâches, alors la somme des probabilités doit être égale à 1 ($\sum pr_i = 1$).

6.4.3 Syntaxe concrète et langage visuel

Il est possible de définir plusieurs syntaxes concrètes pour la même syntaxe abstraite. Nous avons opté pour la dérivation $M_c a$ illustrée dans le tableau 6.1 en faisant correspondre à chaque élément de la syntaxe abstraite un élément visuel (graphique). Il est important de prévoir des éléments visuels aussi distincts (ou semblables) que leurs équivalents abstraits.

6.4.4 Dialecte XML

Actuellement, il s'avère nécessaire de développer des architectures logicielles qui supportent des applications hétérogènes sur les plans modèles d'informations traitées, modes d'échange et de coopération. Cela est dû essentiellement à la complexité et la grande diffusion des systèmes informatiques et l'apparition de lubiquitous computing.

Pour nous, le problème posé pour le formalisme EQNM^{2}L réside dans l'échange de modèles développés entre les différents outils de simulation et d'analyse. Il doit être ouverts et favorise au maximum l'interopérabilité.

6.4.4.1 Interopérabilité

L'interopérabilité est définie comme étant la capacité qu'ont deux ou plusieurs systèmes ou composants de pouvoir échanger de l'information et de pouvoir s'en servir[4].

Le point commun entre toutes les définitions de l'interopérabilité, qu'on trouve dans la littérature, est l'échange et l'exploitation de l'information. On distingue deux niveaux d'interopérabilité : l'interopérabilité syntaxique et l'interopérabilité sémantique. La première concerne la capacité d'échange de l'information, alors que l'autre traite de l'interprétation commune du sens de l'information échangée et de la façon dont celle-ci devra être exploitée. À ces deux niveaux d'interopérabilité, il faut rajouter un troisième niveau : l'interopérabilité technologique qui concerne la coopération entre plusieurs entités logicielles issus de différentes technologies d'implémentation.

On désigne par interopérabilité ici, la possibilité d'échanger des fichiers, avec d'autres utilisateurs équipés de matériels ou de logiciels différents. Pour garantir l'interopérabilité il faut veiller à l'utilisation des *formats de fichiers ouverts*, c'est à dire dont les spécifications sont connues et accessibles à tous. On entend par *standard ouvert* tout protocole de communication, d'interconnexion ou d'échange et tout format de données interopérable et dont les spécifications techniques sont publiques et sans restriction d'accès ni de mise en œuvre. Ainsi, la *compatibilité* est la capacité de deux systèmes à communiquer sans ambiguïté et l'interopérabilité est la capacité à rendre compatibles deux systèmes quelconques. L'interopérabilité nécessite que les informations

4. La IEEE Standard Glossary of Sotware Engineering Terminology

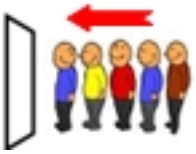
Syntaxe abstraite	Syntaxe concrète
ActiveStation	
AsymmetricStation	
PassiveStation	
Fork	
Join	
Input	
Routing	
Finite Capacity Region	
Outside	
InputToStation - Output - NodeToOutside	

TABLE 6.1: *Dérivation AS-CS pour EQNM²L.*

nécessaires à sa mise en œuvre soient disponibles sous la forme de standards ouverts. L'interopérabilité peut être définie comme étant la capacité d'échanger des informations et d'utiliser mutuellement les informations échangées.

Pour pouvoir échanger les modèles des utilisateurs du langage EQNM²L, il est nécessaire de se référer à un format standard d'échange. L'interopérabilité entre les différents outils d'analyse et de simulation sera encouragé par ce standard. Les standards actuels sont variés, mais la plupart se sert du XML pour son indépendance des plateformes. Nous avons opté pou ce même langage pour proposer une version préliminaire d'un format d'échange des modèles EQNM²L.

Les réseaux de files d'attente de base ou étendues constituent un formalisme mathématique très puissant pour spécifier et modéliser les systèmes discrets d'attente. Ce formalisme permet de décrire les systèmes parallèles et distribués en supportant la concurrence. Les systèmes sont décrit à un niveau d'abstraction permettant d'avoir une indépendance des choix de l'implémentation.

La standardisation de ce formalisme facilite grandement la tâche des spécialistes et des chercheur du domaine de l'évaluation des performances des systèmes. Une telle démarche permet de :

- ✍ Utiliser une terminologie communément acceptée.
- ✍ Avoir un format d'échange qui encourage l'interopérabilité.
- ✍ Développer des extensions sur une base commune et stable.

6.4.4.2 Format d'échange

Il s'agit d'un métamodèle décrit en utilisant un langage différent. Au lieu d'utiliser UML (ou MOF), nous utilisons ici le langage de schéma définit par le W3C. Les deux langages UML et XML-Schema ne manipule pas les concepts de la même manière, ainsi la puissance d'expression est certainement différente. Pour pouvoir décrire la syntaxe abstraite utilisant deux formalismes différents, il est intéressant dans ce cas, de penser à une transformation de modèles.

La syntaxe du format d'échange EQNM²L est décrite par un schéma XML qui reflète le métamodèle. Chaque élément intéressant du métamodèle est projeté en un élément XML équivalent. Pour le concept d'héritage, XML offre les *restrictions* et les *extensions* pouvant être utiles.

Un modèle EQNM²L est sauvegardé dans un fichier XML conforme à un schéma particulier qui décrit la syntaxe abstraite. L'élément racine du document XML est nommé *Qnet* dans la figure 6.4, qui contient, en plus de l'élément *desc*, figure 6.8, les différents composant du réseau de files d'attente. La possibilité d'inclure à chaque fois les différents éléments avec un nombre quelconque d'occurrences est mis en œuvre utilisant la construction *sequence*.

L'élément *Node* constitue l'élément de base depuis lequel les autres stations de service peuvent être définies, figure 6.4.

L'élément *ActiveStation* est une extension de l'élément *Node* et représente toute station active, figure 6.5.

De la même manière, l'élément *PassiveStation* est une extension de l'élément *Node* et représente toute station passive, figure 6.5.

Les éléments *Fork-Join* et *AsymmetricStation* sont aussi des extensions de l'élément *Node*, figures 6.6.

L'élément *Server* modélise un serveur dans une station asymétrique. Il possède un identifiant unique ainsi que les distributions de probabilité qui reflètent le temps moyen avant une panne et celui nécessaire à la réparation, figure 6.7.

L'élément *Service* sur la figure 6.8 représente la façon dont une station offre un service en indiquant pour chaque classe de tâches le nombre de ressources nécessaires. L'élément *Batch* représente un groupe de tâches et l'élément *Distribution* représente une distribution de probabilité avec ses paramètres.

Les arrivées des clients ainsi que leur départ est modélisé utilisant les éléments *Input* et *Output*, figure 6.7.

L'élément *Selection* dans la figure 6.9, représente une station de service particulière. Elle permet à une tâche

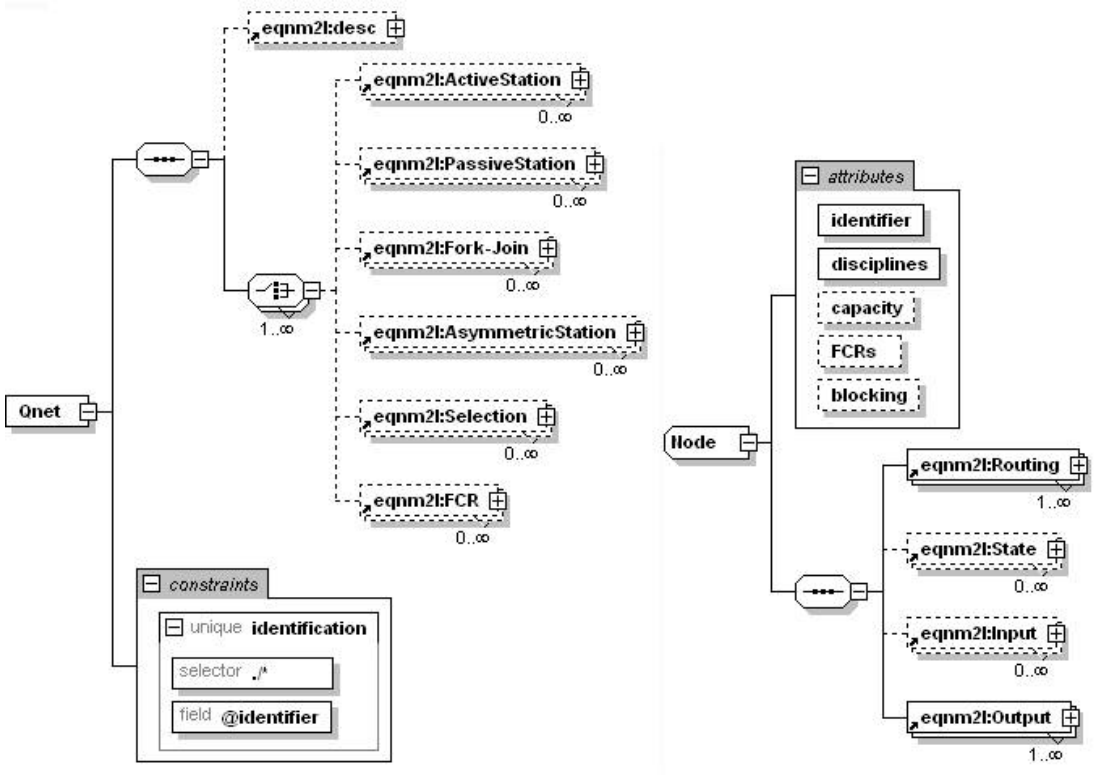


FIGURE 6.4: *Les éléments Qnet (racine) et Node Qnet.*

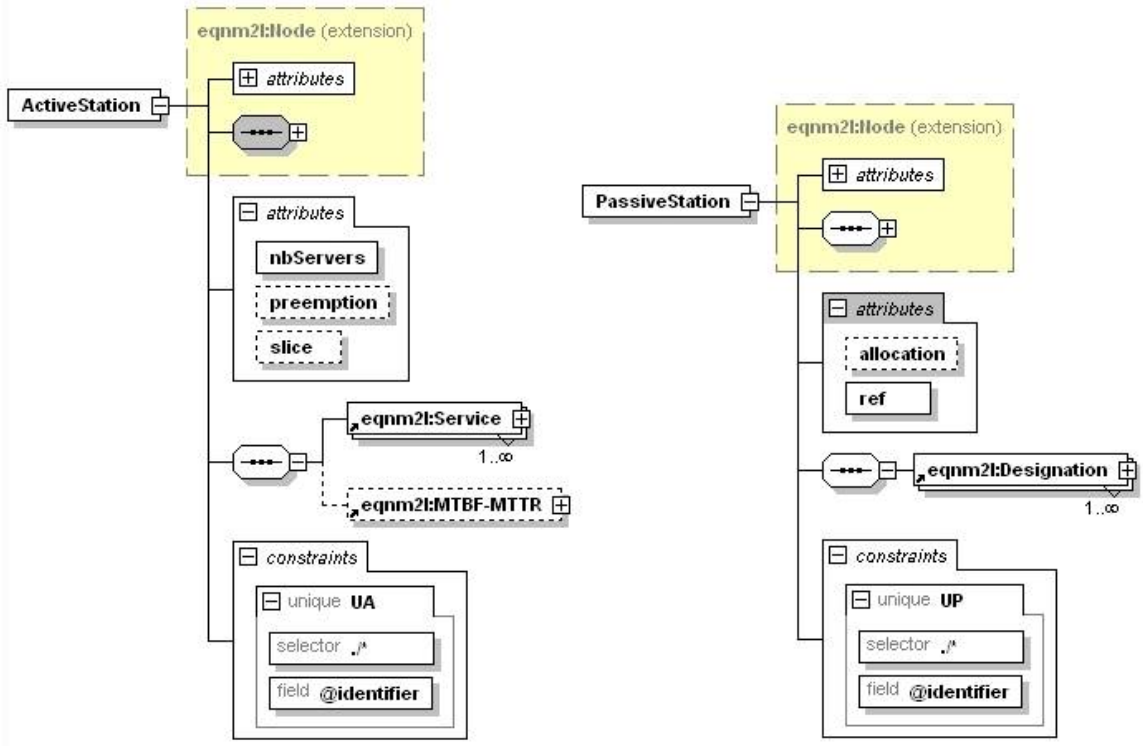


FIGURE 6.5: *Les éléments ActiveStation et PassiveStation.*

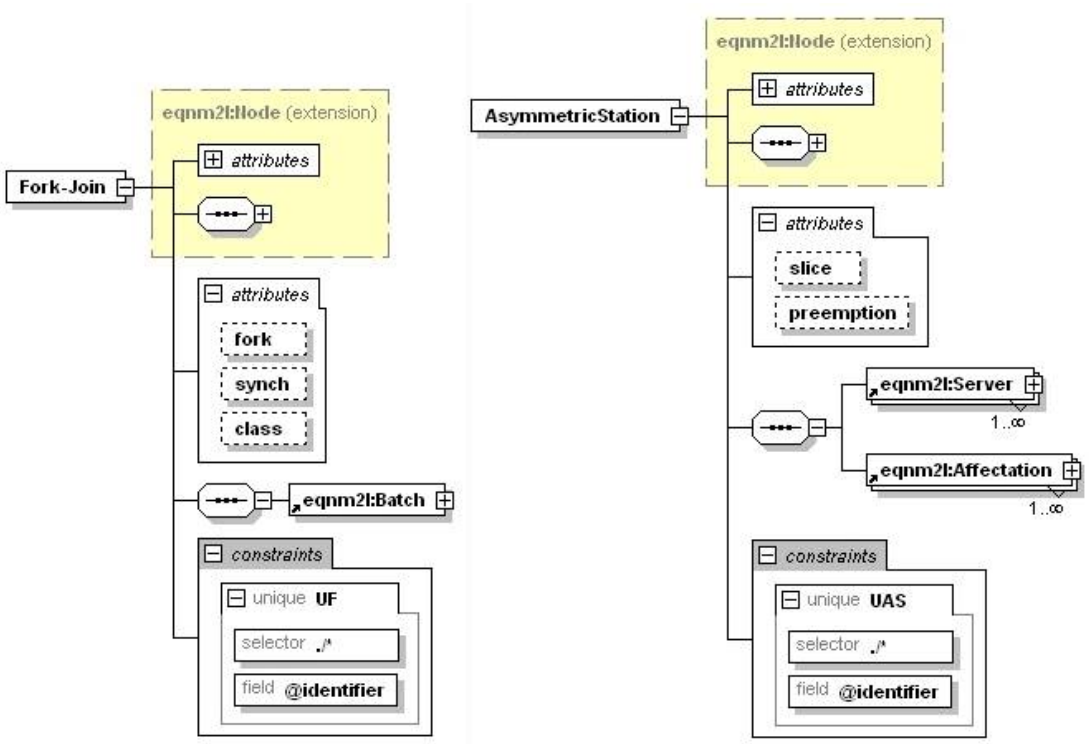


FIGURE 6.6: *Les éléments Fork-Join et AsymmetricStation.*

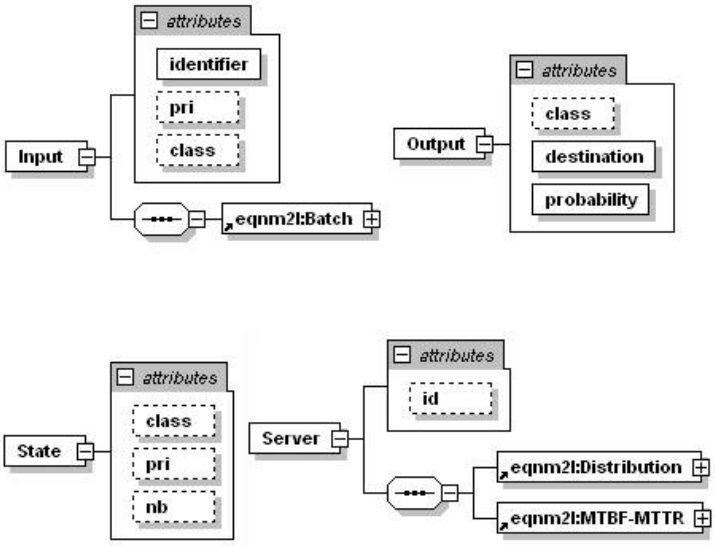


FIGURE 6.7: *Les éléments Input, Output, State et Server.*

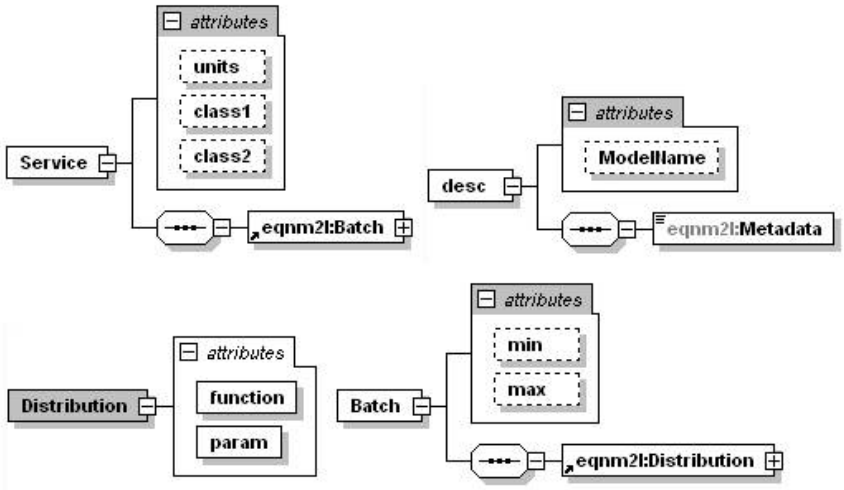


FIGURE 6.8: *Les éléments Service, desc, Distribution et Batch.*

de faire un choix (pour sélectionner une destination) selon l'état du système défini par une expression mathématique de valeur booléenne.

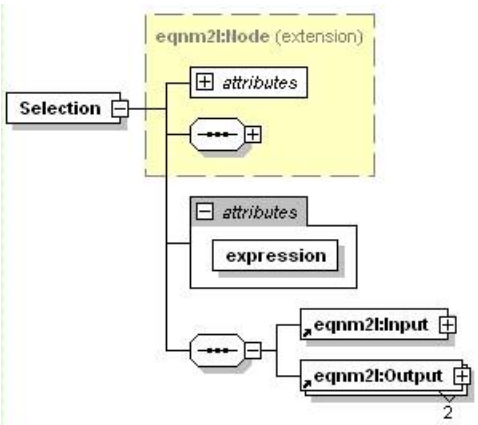


FIGURE 6.9: *L'élément Selection.*

L'élément *Designation* dans la figure 6.10, permet de spécifier un nombre pour une classe particulière de tâches. L'élément *Routing* permet de spécifier une stratégie de routage applicable à une classe de tâches. L'élément *Affectation* permet de spécifier la manière d'affecter les tâches d'une classe particulière à un serveur particulier d'une station asymétrique.

L'élément *FCR* représente une région à capacité limitée et l'élément *MTBF-MTTR* modélise les serveurs non fiables en permettant de spécifier leurs caractéristique concernant les pannes et les réparations.

6.5 Phase d'implémentation

6.5.1 GME ToolKit

L'environnement de modélisation générique GME (Generic Modeling Environment) [Ledeczi et al 01] est une boite à outils assez mûre et configurable pour la création d'environnements de modélisation dédiés et de synthèse de programmes. La configuration est la spécification des paradigmes de modélisation (DSML) à travers les métamodèles. Un paradigme de modélisation contient toutes les informations syntaxiques, sémantiques et de présentation concernant un domaine donné. Les concepts du domaine, les relations entre ceux-ci, la façon dont ceux-ci sont organisés et vus par l'utilisateur ainsi que les règles qui régissent la construction de

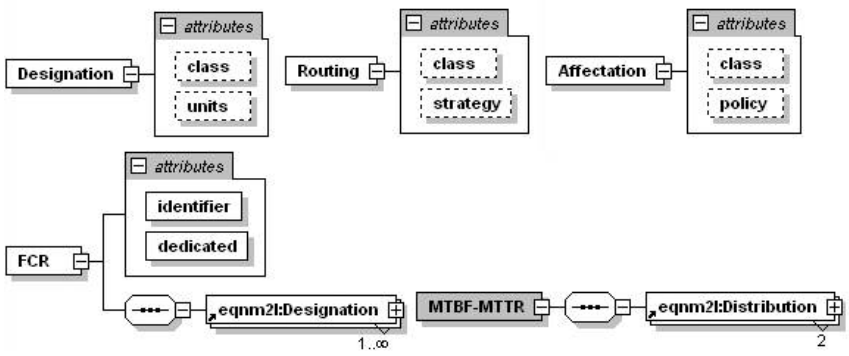


FIGURE 6.10: *Les éléments Designation, Routing, Affectation, FCR et MTBF-MTTR.*

modèles sont les briques de base de GME. Le paradigme définit une famille de modèles pouvant être créés par l'environnement de modélisation qui résulte.

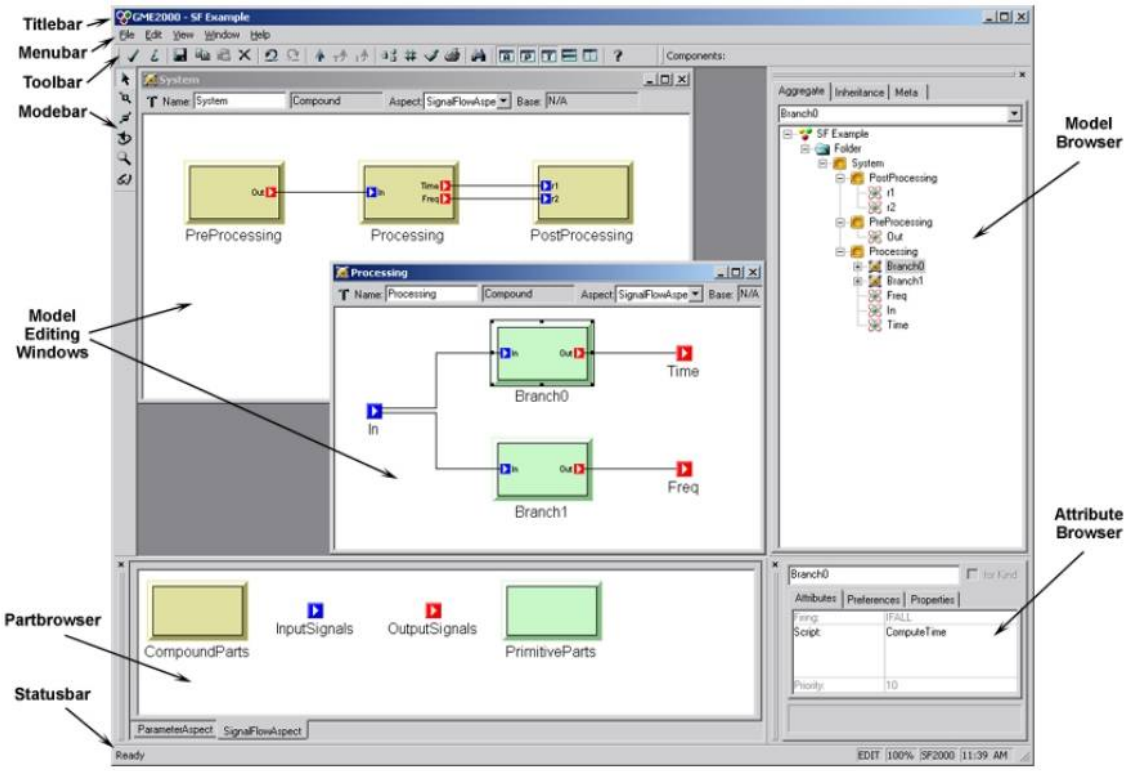


FIGURE 6.11: *Fenêtre principale de GME.*

Le langage de métamodélisation est basé sur les diagrammes de classes UML et les contraintes OCL. Les métamodèles qui spécifient les paradigmes sont utilisés pour générer automatiquement des environnements de modélisation graphiques pour langages dédiés. L'environnement généré est utilisé pour construire des modèles du domaine considéré, qui sont à leur tour sauvegardés dans la base de données de modèles ou dans un format XML.

L'architecture de GME, figure 6.12, est modulaire, extensible et se base sur Microsoft COM pour l'intégration. Les composants externes sont facilement intégrés et peuvent être écrits dans n'importe quel langage qui supporte MS COM. En plus, GME est doté de plusieurs caractéristiques avancées. Un gestionnaire de contraintes permet de forcer le respect des contraintes du métamodèle durant la construction des modèles. GME supporte aussi les aspects de modélisation. Il permet la composition de métamodèles pour renforcer la réutilisation et la combinaison des langages et de concepts de modélisation. Tous les langages de modélisation spécifiés en GME

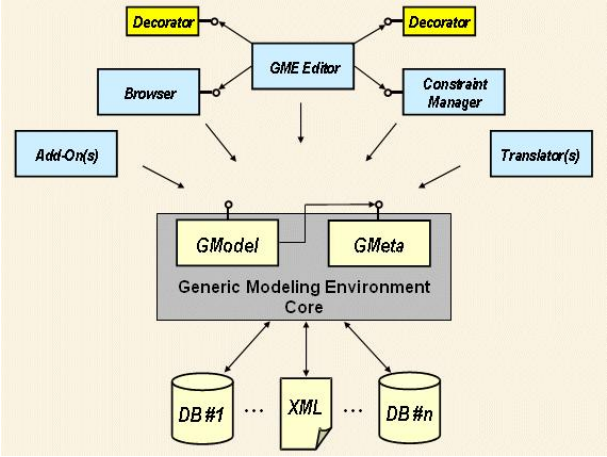


FIGURE 6.12: *L'architecture GME.*

permettent l'héritage des types. La visualisation de modèles est configurable à travers les interfaces décorateurs (Decorator Interfaces).

6.5.2 L'environnement de modélisation graphique d'EQNM²L

L'environnement de modélisation graphique développé sous GME repose sur le métamodèle EQNM²L spécifié à son tour dans le paradigme MetaGME, voir la figure 6.13.

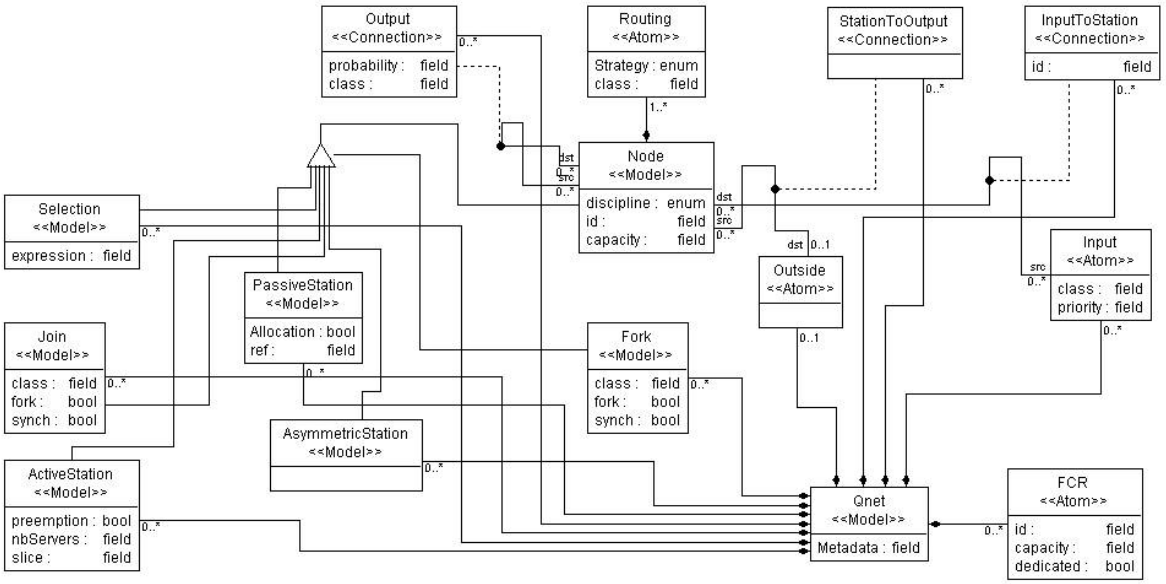


FIGURE 6.13: *Diagramme de classe avec GME et le paradigme MetaGME.*

Le métamodèle exprimé sous forme de diagramme de classes UML de la figure 6.3 est adapté pour MetaGME. Un volet spécial permet de définir les attributs des différents composants du métamodèle et de les relier à leurs classes comme le montre la figure 6.14.

Les contraintes sont exprimé en mode graphique. GME permet de définir deux types de contraintes et les associer aux classes concernées. L'expression des contraintes est spécifique à GME avec un support implicite du langage OCL. Le *Constraint Manager* permet de s'assurer du respect des contraintes durant le développement des modèles dans le formalisme construit. En plus des contraintes discutées dans la section 6.4.2 du chapitre précédent et illustrées dans la figure 6.15, il existe des contraintes prédéfinies par GME, en particulier celle qui permettent de vérifier le respect des cardinalités.

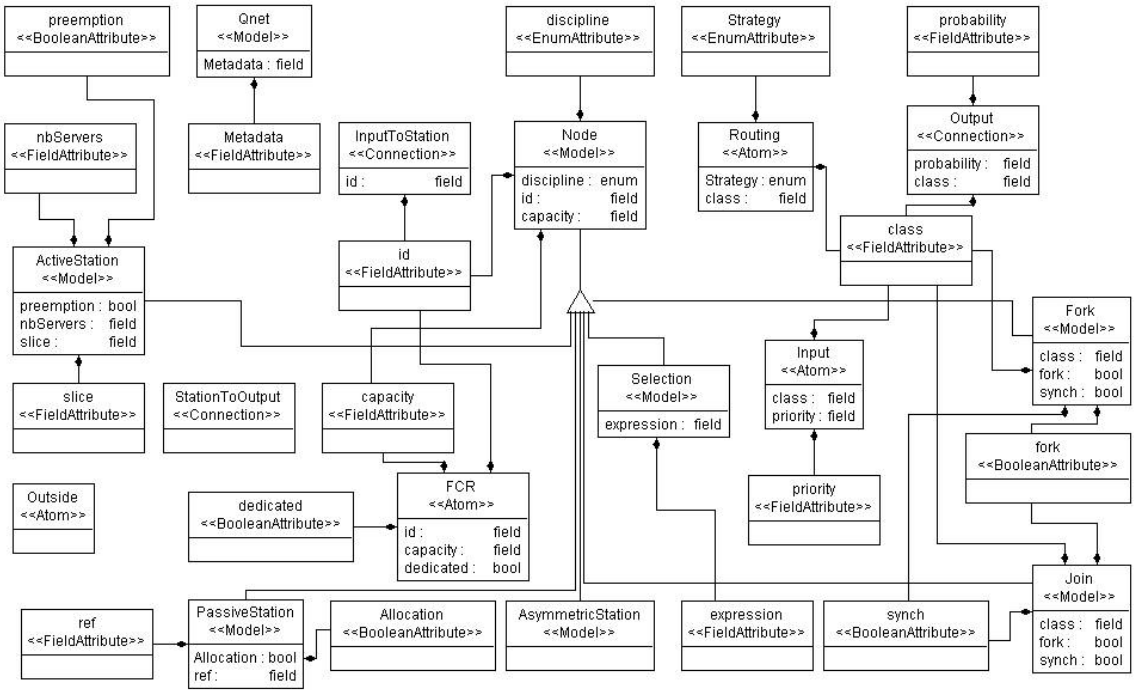


FIGURE 6.14: *Attributs EQNM²L en MataGME.*

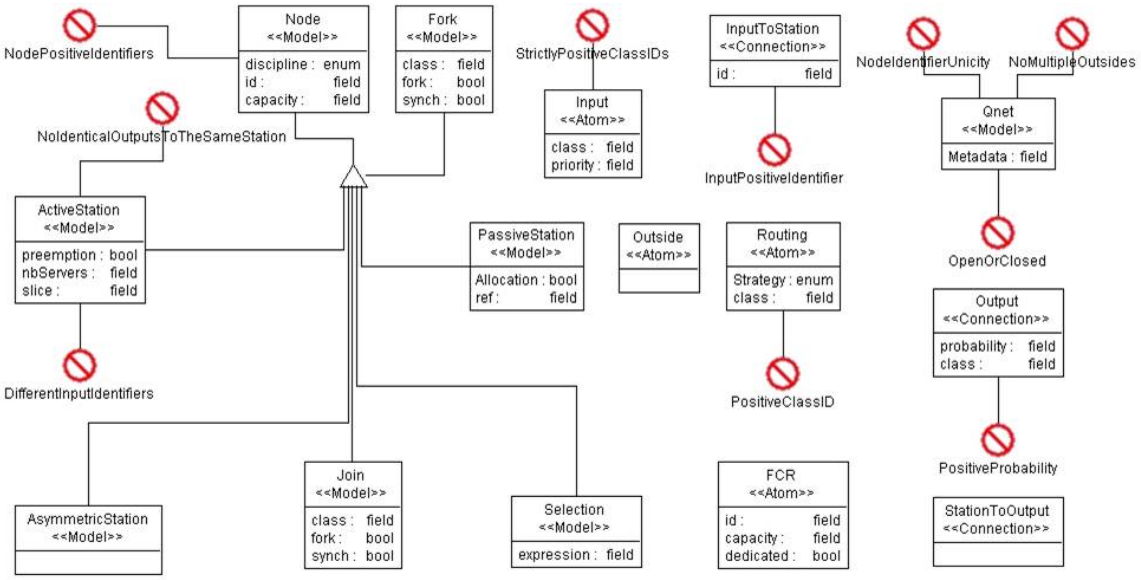


FIGURE 6.15: *Les Contraintes EQNM²L en MetaGME.*

GME permet aussi d'associer plusieurs aspects aux formalismes développés. Dans le cas de la version actuelle d'EQNM²L, un seul aspect est défini et nommé *Connectivity*. Il permet de visualiser tous les concepts nécessaires pour l'élaboration de modèles, voir la figure 6.16.

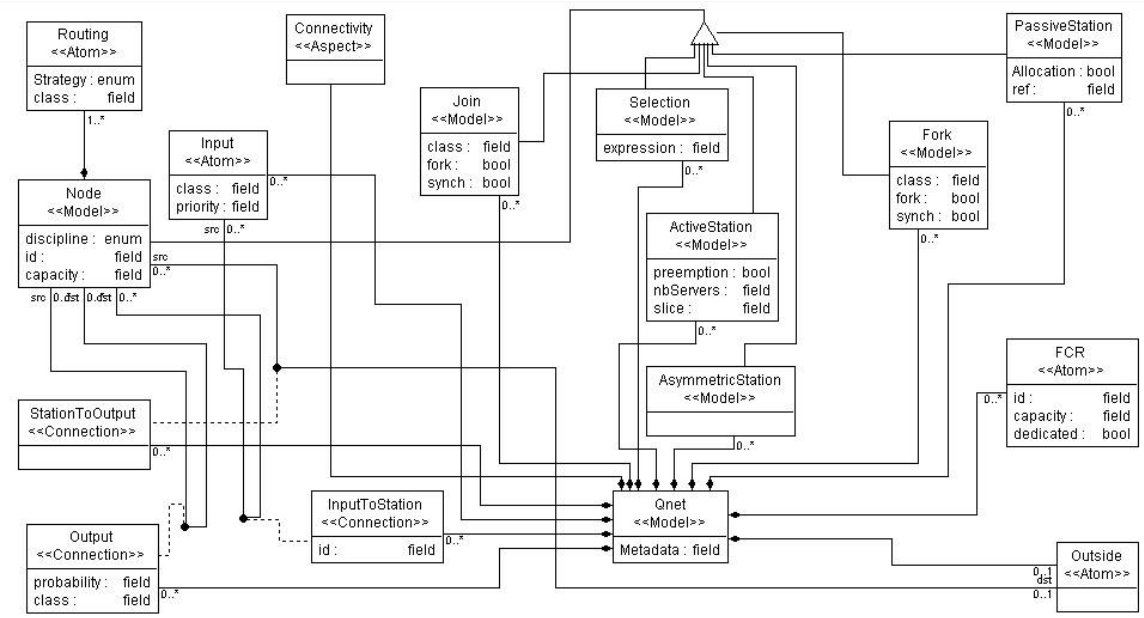


FIGURE 6.16: *Aspect «Connectivity» d'EQNM²L en MetaGME.*

Après interprétation du métamodèle, il est intégré dans l'environnement GME comme nouveau paradigme. Désormais, il est possible de créer des modèles EQNM²L de façon graphique. La figure 6.17 montre le développement d'un modèle de réseau de files d'attente dans un environnement d'édition complet basé sur GME.

6.5.3 Génération automatique de code

La sémantique dynamique d'un DSML peut être spécifiée à travers les différentes transformations que les modèles peuvent subir. Une sémantique opérationnelle est bien adaptée aux modèles directement exécutables ou simulables. Pour EQNM²L, un modèle est soit traduit dans un langage de simulation où il sera *exécuté*, soit traduit dans un format spécifique à un outil d'analyse où il sera *résolu*. Dans le premier cas, il s'agit d'un changement du domaine sémantique et les règles de transformation spécifient une sémantique par translation. Dans le second cas, le modèle obtenu par transformation sera résolu et la transformation peut être aussi considérée comme spécifiant une sémantique par translation[5].

Afin de pouvoir faire le lien entre le modèle source (modèle EQNM²L) et le modèle cible (code en JAPROSIM du simulateur), les deux métamodèles source (métamodèle EQNM²L) et le métamodèle cible (métamodèle du langage JAVA et JAPROSIM) sont nécessaires.

6.5.3.1 Métamodèle JAVA

Le métamodèle Java est décrit par un ensemble de diagrammes UML. La figure 6.18 illustre la notion de classe et son contenu. Elle décrit les notions de paquetage, classe, interface et exception ainsi que les liens entre celles-ci.

La figure 6.19 représente la relation entre une classe qui implémente une interface et étend une autre classe.

La figure 6.20 représente les types de données élémentaires utilisés.

5. La sémantique dynamique des langages de modélisation est une problématique d'actualité qui n'est pas complètement résolue.

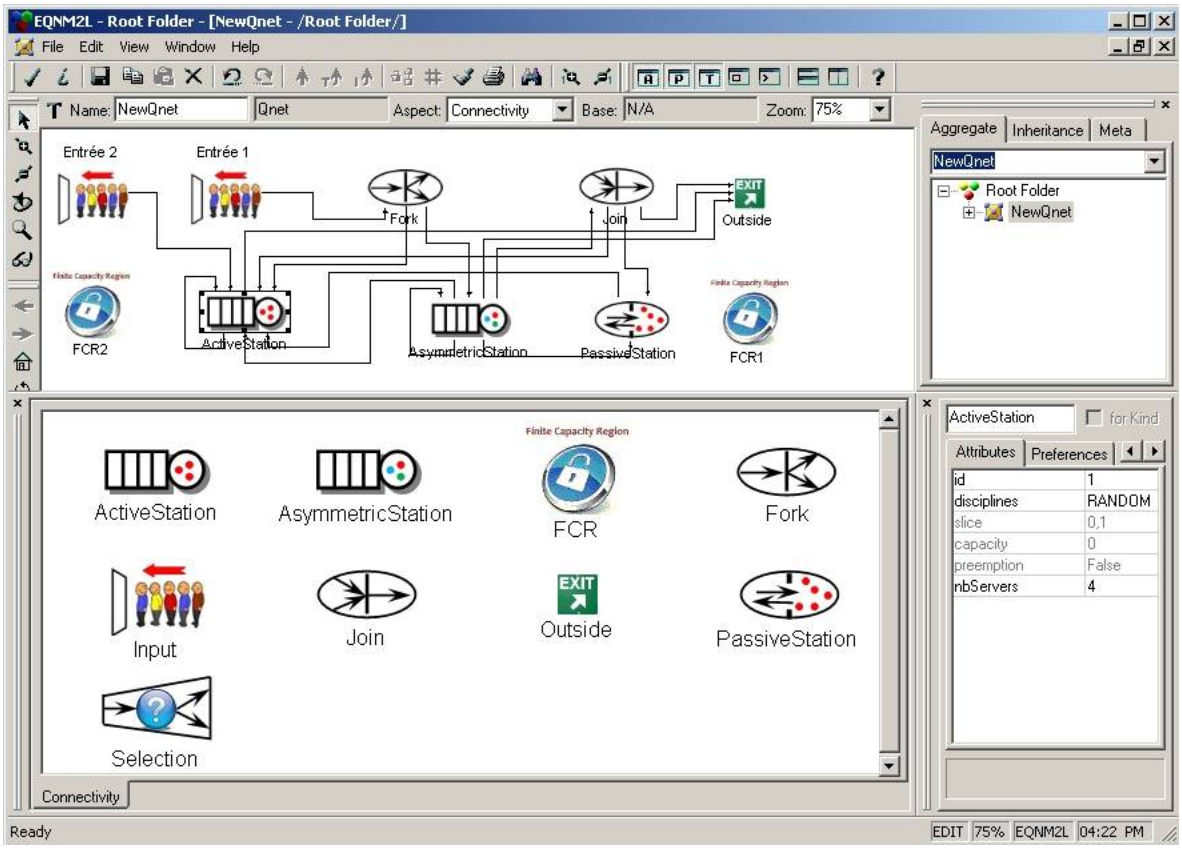


FIGURE 6.17: *Édition d'un modèle EQNM²L en GME.*

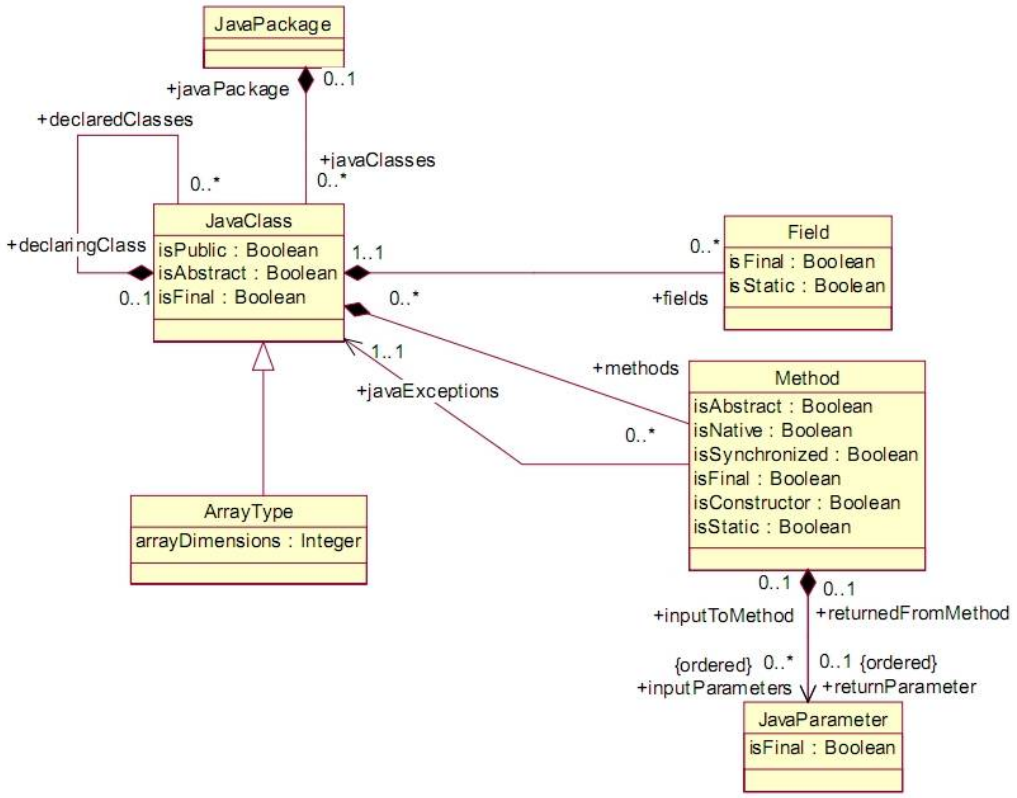


FIGURE 6.18: *Métamodèle Java (Contenu de classe).*

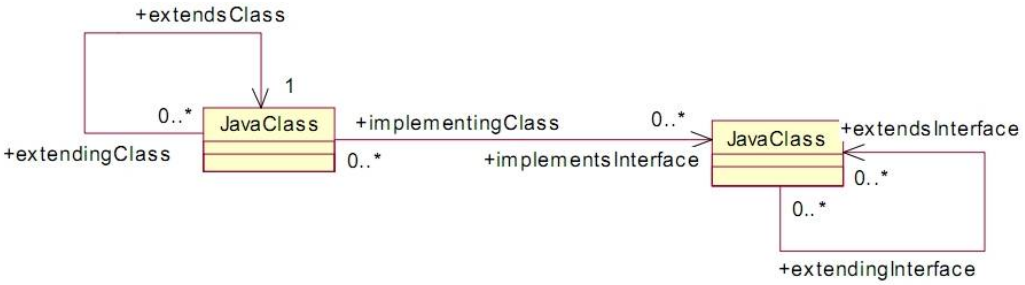


FIGURE 6.19: *Métamodèle Java (Polymorphisme).*

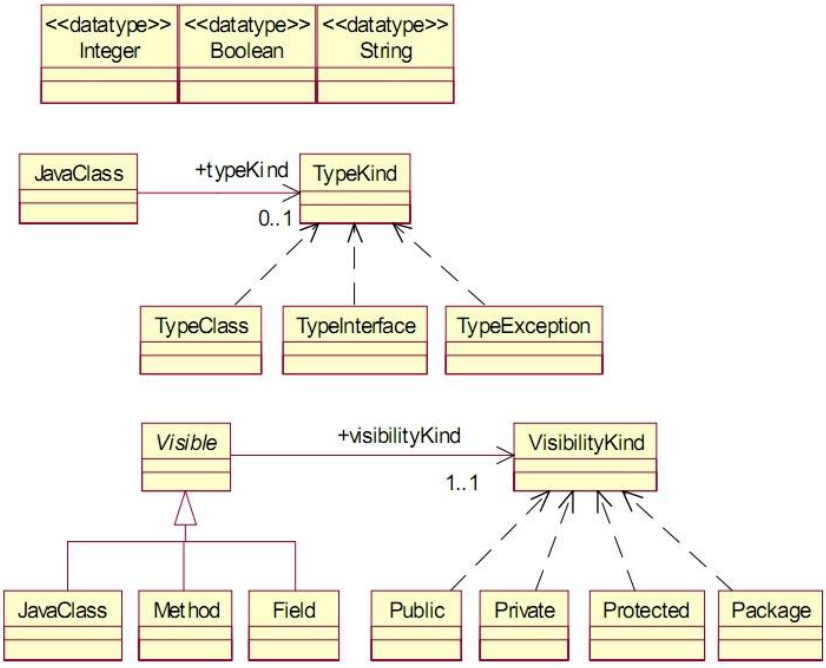


FIGURE 6.20: *Métamodèle Java (Types de données).*

En plus du métamodèle Java retenu, simplifié mais suffisant, nous devons prendre en compte la bibliothèque Japrosim. Le diagramme de classe du package Kernel est indispensable, mais il est nécessaire de prendre en compte tous les détails de la bibliothèque pour formuler les règles de transformation[6].

6.5.3.2 Spécification de règles

Pour transformer un modèle EQNM²L en une application Java autonome, en se basant sur la bibliothèque Japrosim [Bourouis et Belattar 08b], un ensemble de règles doit être spécifié. Nous décrivons ici, tout en essayant d'éviter les ambigüités, les règles nécessaires de façon informelle, utilisant le langage naturel :

1. Chaque modèle EQNM²L nécessite une application Java à part entière :
   (a) Création d'une classe Java, ayant le même nom (ou un nom par défaut) que le modèle EQNM²L, comportant la méthode *main()* et en insérant les instructions d'importation nécessaires.
   (b) Création des flux de tâches, qui correspondent aux éléments *Input* en créant pour chacun une instance de la classe *Transaction_i_j* et en invoquant la méthode *beginAfter()* de celle-ci. Cela permet d'initialiser le modèle.
   (c) Mise à jour des valeurs initiales du modèle concernant la date de début de simulation, la durée de simulation et le nombre de réplications.
   (d) Création d'une classe nommée *Transaction* qui hérite de la classe prédéfinie *Entity* de Japrosim. Cette classe comporte toutes les données partagées du modèle comme les queues, les ressources, les lois de services des différentes stations, …etc.
2. Chaque élément *Input* nécessite une classe Java nommée *Transaction_i_j* où *i* est l'identifiant de la station cible et *j* est l'identifiant de l'élément *Input*.
   (a) Une classe *Transaction_i_j* doit hériter de la classe *Transaction* de Japrosim.
   (b) Créer un générateur de nombres pseudo-aléatoires sous forme d'un membre statique nommé *arrival* selon la distribution d'arrivée indiquée dans l'élément *Input* (plus précisément ses éléments fils *Batch* et *Distribution*).
   (c) Création de la méthode *body()* de la classe *Transaction_i_j*.
   (d) Création d'une instance dans la méthode *body()* et invocation de sa méthode *beginAfter()* avec une valeur du générateur préalablement créé.
   (e) Invocation de la méthode publique sans valeur de retour *intoStation_j()* avec *j* l'identifiant de la station cible comme paramètre.
   (f) Prendre en compte la stratégie de routage de la station traitée. S'il est probabiliste par exemple, créer des tableaux qui permettent de calculer les valeurs des probabilités associées à chaque sortie et des les probabilités cumulatives.
   (g) Prévoir les instructions nécessaires à l'acquisition de service dans la station considérée.
   (h) Diriger la tâche vers sa destination en invoquant la méthode *intoStation()* appropriée.

Les règles ont été par la suite en XSLT, du fait que les modèles sources pris pour la génération du code Java sont ceux dans le format standardsous forme XML. Généralement la génération du code s'appuie sur la syntaxe abstraite et non sur les syntaxes concrètes.

Le fragment de code XSLT suivant :

Listing 6.1: *Règles de transformation en XSLT.*

```
4 <xsl:output encoding="-UTF8" -mediatype="text/plain␣"method="text␣"
      --omitxmldeclaration="yes"/>
5 <xsl:template match="input">
6 <xsl:text>
```

6. Pour les détails de la bibliothèque Japrosim, se référer au chapitre précédent 5.

```
7  class Transaction</xsl:text>
8  <-xsl:valueof select="../@identifier"/>
9  <xsl:text>_</xsl:text>
10 <-xsl:valueof select="./@identifier"/>
11 <xsl:text> extends Transaction{
12 </xsl:text>
13 <xsl:choose>
14 <xsl:when test="dist/@function[contains(.,'Exponential')]">
15 <xsl:text> static Exponential arrival = new Exponential (</xsl:text>
16 <-xsl:valueof select="dist/@param1"/>
17 <xsl:text>);</xsl:text>
18 </xsl:when>
```

Met en œuvre la 2ème règle citée avant où pour chaque élément *Input* (ligne 5) une classe Java nommée *Transaction_i_j* (ligne 7) où *i* est l'identifiant de la station cible (ligne 8) et *j* est l'identifiant de l'élément Input (ligne 10). La partie (a) est mise en œuvre par la ligne 11. Si la loi d'arrivée est exponentielle (ligne 12) alors un générateur correspondant est crée sous forme d'un membre statique (ligne 15) avec le paramètre correspondant (ligne 16) ce qui correspond à la partie (b). De la même manière, il faut tester sur la loi d'arriver si elle est uniforme, Normale, …etc.

Listing 6.2: *Règles de transformation en XSLT.*

```
66 <xsl:text> public void body ( ) {
67 </xsl:text>
68 <xsl:text> new Transaction</xsl:text>
69 <-xsl:valueof select ="../@identifier"/>
70 <xsl:text>_</xsl:text>
71 <-xsl:valueof select="./@identifier"/>
72 <xsl:text>().beginAfter(arrival.sample()) ;
73 </xsl:text>
74 <xsl:text> intoStation</xsl:text>
75 <-xsl:valueof select="../@identifier"/>
76 <xsl:text>();
77 </xsl:text>
78 <xsl:text> }
79 </xsl:text>
80 <-xsl:foreach select ="//station">
81 <xsl:text> public void intoStation</xsl:text>
82 <-xsl:valueof select="./@identifier"/>
83 <xsl:text>() {
84 </xsl:text>
85 <xsl:text> double[] vals={</xsl:text>
```

La ligne 66 met en œuvre la partie (c) de la 2ème règle. Les lignes de 68 à 73 pour la partie (d). La partie (e) est mise en œuvre par les lignes de 74 à 79. Les lignes de 80à 85 permettent d'écrire le corps de la méthode intoStation_*j*().

6.6 Conclusion

Dans ce chapitre, nous avons conçu et développé un DSML pour la modélisation des systèmes à évènements discrets. Le formalisme EQNM²L étend les réseaux de files d'attente simples avec de nouvelles constructions, ce qui lui permet de modéliser un grand éventail des systèmes à évènements discrets. La conception est bâtie sur une base formelle en s'adhérant à l'IDM et à la méta-modélisation. Son métamodèle qui incarne les concepts essentiels du domaine sans ambigüité permet d'avoir une syntaxe abstraite claire et formelle. Les syntaxes concrètes peuvent ainsi varier selon les éditeurs graphiques. De notre part, nous avons suivi un cycle de développement conforme à celui adopté actuellement pour les DSMLs.

Les modèles EQNM^{2}L sont échangés dans un format standard basé sur XML pour favoriser l'interopérabilité entre les différents outils de simulation et d'analyse. Ce format a servi aussi pour la transformation de modèle en vue de générer automatiquement le code de simulation. Nous avons opté pour XLT comme langage de transformation en se basant sur le standard. Les règles de transformation pour produire un code java selon la bibliothèque Japrosim ont été exprimé d'abord en langage naturel pour gagner en abstraction. Ensuite, ces mêmes règles ont été exprimé en XLT pour l'obtention du code. le même processus peut être adopté pour tout autre bibliothèque de simulation écrite dans n'importe quel langage de programmation et même pour les autres EMS et outils d'analyse ayant des format propriétaires.

L'environnement de modélisation graphique a été mis en œuvre utilisant la boite à outils GME. A partir du métamodèle en UML et les contraintes exprimées sous forme spécifique à GME, avec un support du langage OCL, un environnement intégré de modélisation graphique est généré automatiquement. Il permet la construction de modèles conformément au métamodèle et en respectant les contraintes préétablies. L'environnement permet aussi de bénéficier des opérations usuelles d'édition comme le copier/couper/coller, la suppression de composant, l'édition des attributs, ouverture/enregistrement et validation de modèles. Pour homogénéiser le projet, les règles de transformation peuvent être réécrite utilisant le langage GReAT soutenu par GME.

Conclusion générale et perspectives

Le travail réalisé dans cette thèse s'inscrit dans le cadre des DSMLs visuels qui visent à automatiser et simplifier la tâches des spécialistes du domaine. Les langages de modélisation dédiés permettent de spécifier la solution dans un haut niveau d'abstraction et à partir de laquelle le code complet de bas niveau est généré automatiquement. Ce domaine est en pleine expansion et d'un grand intérêt.

L'intérêt du travail réalisé réside dans les sources d'inspiration variées ainsi que leur application au domaine de l'évaluation des performances des systèmes à évènements discrets. Le formalisme graphique EQNM^{2}L développé étant un outil assez puissant qui se base sur la théorie des files d'attente en proposant de nouvelles extensions. Sa conception rigoureuse et le développement de l'environnement d'édition intégré selon une démarche bien claire permettent de valoriser encore les efforts déployés.

En s'inspirant des théories issues de la psychopédagogie concernant la représentation des connaissances sous forme de concept-maps, nous sommes convaincu que la meilleure façon de procéder est de construire des connaissances atomiques, de les lier de façon significative et constructive, ainsi que de prendre en compte l'utilisateur final. La théorie des langages visuels en informatique nous a aidé à mieux situer les concept-maps, qui en réalité sont une famille de langages diagrammatiques où les connaissances sont portées dans les attributs des nœuds et des liens qui les relient. La disposition spatiale n'a pas d'importance dans les concept-maps. Le champ d'application est très vaste allant de la représentation de connaissances à leur présentation, de la modélisation à un niveau extrêmement abstrait à la programmation de bas niveau. Du fait que nous nous intéressons au domaine de la modélisation des systèmes à évènements discrets, le domaine des langages dédiés nous a fournit les principes de base concernant l'architecture d'un DSML ainsi que les méthodologies de son développement. Cela nous a conduit dans notre étude à se concentrer sur les langages de modélisation dédiés visuels et plus particulièrement diagrammatiques.

Durant le développement du formalisme EQNM^{2}L que nous avons proposé pour l'élaboration des modèles conceptuels des systèmes à évènements discrets, l'IDM nous a servi de guide pour la spécification formelle sur le plan syntaxique et sémantique grâce à la méta-modélisation et la transformation de modèles. Le langage XML nous a servi de support pour développer un format d'échange standard permettant l'interopérabilité entre les outils de simulation et d'analyse des systèmes à évènements discrets.

L'environnement de développement intégré a été généré automatiquement à partir des spécifications formelles. Il permet l'édition de modèles conceptuels de façon graphiques, utilisant le formalisme EQNM^{2}L simple et intuitif, mais suffisamment puissant pour pouvoir exprimer une grande partie de la dynamique des systèmes à évènements discrets. Les modèles conçus sont interopérable sous forme XML. Le code de simulation est généré automatiquement utilisant la transformation de modèles. Théoriquement, il est possible d'utiliser n'importe quel langage de simulation. Pour cela il suffit juste d'intégrer les règles spécifiques. Même pour les outils d'analyse, il est possible de transformer le modèle dans leurs formats propriétaires.

Nous avons prouvé par le présent l'intérêt du développement d'un DSML graphique à partir des spécifications jusqu'à la génération automatique du code de façons automatisée. En exploitant la réutilisation, le processus de développement peut être itératif ainsi que la conception de nouvelles versions ou de nouveaux DSMLs de la même famille peut s'articuler sur ce qui a été réalisé. Le gain énorme en productivité est perçu non seulement au niveau de l'environnement de modélisation, mais également au niveau du code généré automatiquement.

Ce qui fait gagner aussi en abstraction, en maintenance, en interopérabilité et en extensibilité.

Le travail réalisé dans cette thèse donne lieu à quelques perspectives. Au cours des travaux effectués, nous avons proposé un nouveau formalisme visuel pour la modélisation des systèmes à évènements discrets en vue d'évaluer leurs performances. L'analyse se fera par résolution ou par simulation de modèles. Nous avons développé un environnement de modélisation graphique sous GME, un format d'échange basé sur XML et une bibliothèque de simulation à évènements discrets. Nous envisageons continuer à travailler sur divers plans :

✍ Améliorer la bibliothèque Japrosim en lui intégrant des modules pour :

1. La présentation graphiques des résultats de simulation.
2. La détection de la phase stable du fait que les performances calculées concernent en majorité celle-ci.
3. L'animation graphique des modèles.
4. L'explication et l'argumentation.
5. L'assistance à l'analyse des données d'entrée.
6. La simulation continue et hybride.

✍ Améliorer le formalisme EQNM²L sur les plans :

1. Expressivité : en intégrant de nouveaux concepts.
2. Degré de spécificité : se baser sur ce formalisme pour proposer d'autres adaptés à des domaines plus spécifiques, comme les réseaux de communication, les systèmes de production, systèmes réparties, …etc.
3. Développement d'un environnement intégré qui prend en charge toutes les phases depuis l'élaboration du modèle conceptuels jusqu'à la génération du code. Nous avons développé un ensemble de modules chacun pour une phase mais qui ne sont pas actuellement rassemblés en une seule application autonome.
4. Développer d'autres générateurs de code pour les outils de simulation et d'analyse les plus utilisés pour garantir encore le succès de déploiement.
5. Impliquer une large communauté dans le processus de développement et de tests[7].

7. Un site consacré à ce formalisme Open Source à été mis au point : http://eqnm2l.sourceforge.net/

Bibliographie

[Agrawal et al 06] Agrawal, A. Karsai, G. Kalmar, Z. Neema, S. Shi, F. and Vizhanyo, A. *The Design of a Language for Model Transformations.* Journal on Software and System Modeling, 5(3) :261-288, September 2006. 35

[Ambron 88] Ausubel, P. D. *Clustering : An interactive technique to enhance learning in biology.* Journal of College Science Teaching. vol 18,pp 122-144. 8, 9

[Ausubel 63] Ausubel, P. D. *The psychology of meaningful verbal learning.* New York : Grune and Stratton, 1963. 3

[Ausubel 68] Ausubel, P. D. *Educational Psychology : A Cognitive View.* New York : Holt, Rinehart and Winston, 1968. 3

[Backhouse 00] Backhouse Kevin. *Domain Specific Language Extensions.* PhD Dissertation, Oxford University, March 2000. 17

[Balasubramanian et al 04] Balasubramanian, K. Gokhale, A. Karsai, G. Sztipanovits, J. et Neema, S. *Developing Applications Using Model-Driven design Environments.* Computer, 39(2) : 33–40, 2006. 24

[Banks 91] Banks, J. *Selecting Simulation Software.* In Proceeding of the 1991 Winter Simulation Conference, ed. B.L. Nelson, W.D. Kelton, G.M. Clark, 15-20. Piscataway, New Jersey : Institute of Electrical and Electronics Engineers. 50

[Banks et al 05] Banks, J. Carson, J. S. Nelson, B. L. et Nicol, D. M. *Discrete-Event System Simulation.* Fourth Edition. Pearson Prentice Hall, 2005. 38

[Bardohl et Taentzer 97] Bardohl, R et Taentzer, G. *Defining Visual Languages by Algebraic Specification Techniques and Graph Grammars.* In Proc. Workshop on Theory of Visual Languages, pages 27-42, Capri, Italy, 27 September 1997. 24

[Belattar et Djoudi 00] Belattar, B. et Djoudi, M. *Integration of animation and explanation in modelling and simulation environment.* Actes du colloque CARI2000, Antananarivo, Madagascar, 11-15 octobre 2000, pp 455-462. 53

[Belattar et Laraba 97] Belattar, B. et Laraba, M. E. *Apport de l'explication en simulation.* Actes de la conférence MOSIM97, Rouen, France, pp 527,537. 53, 56

[Billington et al 03] Billington. J, Christensen. S, van Hee. K, Kindler. E, Kummer. O, Petrucci. L, Post. R, Stehno. C et Weber. M. *The Petri Net Markup Language : Concepts, technology, and tools.* Application and Theory of Petri Nets 2003, 24th International Conference. In W. van der Aalst and E. Best, editors, volume 2679 of LNCS, pp. 483-505. Springer, June 2003. 77

[Birta et Arbez 07] Louis G. Birta and Gilbert Arbez. *Modelling and Simulation : Exploring Dynamic System Behaviour.* Springer-Verlag London Limited 2007. 39

[Blay-Fornarino 06] Blay-Fornarino J.-M. F. J. E. M. *L'ingénierie dirigée par les modèles. Au-delà du MDA.* Hermes Sciences / Lavoisier, 2006 35

[Bolsh et al 06] Bolsh. G et al. *Queueing Networks and Markov Chains : Modeling and Performance Evaluation with Computer Science Applications.* Second Edition. John Wiley and Sons. 2006. 78, 80

[Boshernitsan et Downes 97] Boshernitsan. M, et Downes. M. *Visual Programming Languages : A survey.* University of California, Berkeley, Berkeley, 1997. 15

[Bothner 03] Bothner, P. *Kawa, the Java-based Scheme system.* 2003. v. 1.7, THE KAWA LANGUAGE FRAMEWORK : http ://www.gnu.org/software/kawa/. 10

[Bourouis 03] Bourouis Abdelhabib. *Intégration de la dimension explicative dans les modèles de simulation à évènements discrets.* Thèse de Magister, Université de Batna, 2003. v, 53, 57

[Bourouis et Belattar 06] Bourouis. Abdelhabib, et Belattar. Brahim. *Impact du choix de l'entité active sur les performances d'une simulation orientée processus multithreds basée sur Java.* Proceedings of the CIIA 06, Saida, Algeria, 15-16 Mai 2006. 65

[Bourouis et Belattar 08a] Bourouis. A, Belattar. B, *JAPROSIM : A Java Framework for Discrete Event Simulation.* Journal of Object Technology, vol. 7, no. 1, January-February 2008, pp. 103-119, (2008). . xi, 70, 74

[Bourouis et Belattar 08b] Bourouis. Abdelhabib, et Belattar. Brahim. *Using XML in Simulation Modelling : automatic code generation for XML-based models.* Proceedings of the CARI 08, pp 101-108. Tanger, Morocco, October 23-25, 2008. 62, 97

[Briggs et al 04] Briggs, G, D. A Shamma, A. J Cañas, R Carff, J Scargle, et J. D Novak. *Concept maps applied to Mars Exploration public outreach. Concept Maps : Theory, Methodology, Technology.* First International Conference on Concept Mapping. Pamplona, Spain, 2004. 3

[Budinsky et al 03] Budinsky, Frank. Steinberg, David. Merks, Ed. Ellersick, Raymond et Grose, Timothy J. *Eclipse Modeling Framework : A Developer's Guide.* Addison Wesley, 2003. 26, 34

[Buss 02] Buss. A. *Component Based Simulation Modeling with SimKit.* Proceedings of the 2002 Winter Simulation Conference, pages 243-249. 64

[Buzan 89] Buzan Tony. *Use Your Head.* London : BBC Books, 1989. 8

[Buzan 91] Buzan Tony. *Use Both Sides of Your Brain.* New York, Dutton (Penquin Books), 1991. 8

[Buzan 94] Buzan Tony. *The Mind Map Book.* New York, Dutton (Penquin Books), 1994. 8

[Calcinelli et Mainguenaud 94] Calcinelli, D, et M Mainguenaud. *Cigales, A Visual Query Language for a Geographical Information System : the User Interface.* Journal of Visual Languages and Computing, 1994 : 113-132. 15

[Carbogim et al 00] Carbogim, D. Robertson, D. et Lee, J. *Argument-based applications to knowledge engineering.* Knowledge Engineering Review, 15(2) :119.149. 2000. 8

[Caron 07] Caron Pierre-André *Ingénierie dirigée par les modèles pour la construction de dispositifs pédagogiques sur des plateformes de formation.* Thèse de doctorat de l'université des sciences et technologies de Lille. 18 juin 2007. 24

[Carson 05] Carson II, J. S. *Introduction to modeling and simulation.* Édité par M. E Kuhl, N. M Steiger, F. B Armstrong et J. A Joines. Proceedings of the 2005 Winter Simulation Conference. 2005. 16-23. 38

[Chalavadi 04] Chalavadi Uma Maheshwar *Automatic configuration of queueing network models from business process descriptions.* Master of Science Thesis. Oklahoma State University, December 2004. 77, 78

[Chesñevar et al 00] Chesñevar, C. I, Maguitman, A. et Loui, R. *Logical models of argument.* ACM Computing Surveys, 32(4) :337.383, 2000. 8

[Chung 04] Chung Christopher, A. *Simulation modeling handbook : a practical approach.* CRC Press LLC, 2004. 38, 40, 42, 43

[Clark et al 04] Clark, T. Evans, A. Sammut, P et Willans, J. *Applied Metamodelling : A Foundation for Language Driven Development* 2004, version 0.1. 34, 35

[Combemale et al 06] Benoît Combemale, Sylvain Rougemaille, Xavier Crégut, Frédéric Migeon, Marc Pantel et Christine Maurel. *Expériences pour décrire la sémantique en ingénierie des modèles.* Actes des 2èmes journées sur l'Ingénierie Dirigée par les Modèles IDM'06. Editeurs : Laurence Duchien et Cédric Dumoulin. Lille, Juin 2006. 34, 36

[Conklin et Begeman 87] Conklin, J, et M. L Begeman. *A Hypertext tool for Team Design Deliberation.* Édité par gIBIS. Hypertext'87. 247-251. 1987. 15

[Costagliola et al 97] Costagliola, G. De Lucia, A. Orefice, S. Tortora, G. *A Parsing Methodology for the Implementation of Visual Systems.* IEEE Transactions on Software Engineering. vol 23, pp 777-799. 1997. 33

[Cook et al 07] Cook Steve, Jones Gareth, Kent Stuart, Wills Alan Cameron. *Domain-Specific Development with Visual Studio DSL Tools.* Addison-Wesley. May 2007. 17

[Crimi et al 91] Crimi, C., Guercio, A., Nota, G., Pacini, G., Tortora, G. et Tuccit, M. *Relation Grammars and their Application to Multi-dimensional Languages.* Journal of Visual Languages and Computing 2(4) : 333-346. 1991. 33

[Erwig et Meyer 95] Erwig. M et Meyer. B. *Heterogeneous Visual Languages : Integrating Visual and Textual Programming.* 11th International IEEE Symposium on Visual Languages, September 5-9, 1995, Darmstadt, Germany, Proceedings. 15

[Frakes et al 98] Frakes, W., Prieto-Diaz R. et Fox, C. *DARE : Domain analysis and reuse environment.* Annals of Software Engineering,5 :125-141, 1998. 19

[Ferrucci et al 96] Ferrucci, F. Pacini, G. Satta, G. Sessa, M. Tortora, G. Tucci, M. et Vitiello, G. *Symbol -Relation Grammars : A Formalism for Graphical Languages.* Information and Computation, 131, 1-46, 1996. 33

[Gaines 91] Gaines, B. R. *An Interactive Visual Language for Term Subsumption Languages.* Sydney- Australia : IJCAI-91, 1991. 6, 15

[Garrido 01] Garrido J. M. *Object-oriented Discrete Event Simulation with Java.* Kluwer/Plenum, NY, September 2001. 63, 64

[Gerber et al 02] Gerber, A. Lawley, M. Raymond, K. Steel, J. Wood, A. *Transformation : The missing link of MDA. In : Graph Transformation.* Volume 2505 of Lecture Notes in Computer Science, Springer-Verlag (2002) 90-105 Proc. 1st International Conference. Graph Transformation 2002, Barcelona, Spain. 31

[Golin et Reiss 89] Golin, E. J. et Reiss, S. P. *The Specification of Visual Language Syntax.* Procs. of the IEEE Workshop on Visual Languages, Rome, Italy, pp. 105-110. October 1989. 33

[Greenfield et Short 04] Greenfield, J. et Short, K. *Software Factories : Assembling Applications with Patterns, Models,Frameworks, and Tools.* John Wiley and Sons, 2004. 24

[Grigorenko et al 05] Grigorenko, Pavel. Saabas, Ando et Tyugu, Enn. *Cocovila - compiler-compiler for visual languages.* Electr. Notes Theor. Comput. Sci., 141 :137-142, 2005. 34

[Gurevich 01] Gurevich Y. *The Abstract State Machine Paradigm : What Is in and What Is out.* Ershov Memorial Conference, 2001. 35

[Healy et Kilgore 97] Healy, K. J. et Kilgore, R. A. *SilkTM : A Java-Based Process Simulation Language.* Proceedings of the 1997 Winter Simulation Conference, pp. 475-482, December 7-10, 1997. 64

[Helm et Marriott 90] Helm, R. et Marriott, K. *Declarative Specification of Visual Languages.* Procs. of the IEEE Workshop on Visual Languages, Skokie, Illinois, 98-103. 1990. 24

[Helsgaun 00] Helsgaun Keld. *Discret Event Simulation in Java.* DATALOGISK SKRIFTER (writings on computer science), Roskilde University, 2000. 64

[Hlupic et Mann 95] Hlupic Vlatka et Mann, Amardeep. S. *SimSelect : A System for Simulation Software Selection.* In Proceedings if the 1995 winter Simulation Conference. 50

[Howell et McNab 98] Howell Fred et McNab Ross. *simjava : a discrete event simulation package for Java with applications in computer systems modelling.* First International Conference on Web-based Modelling and Simulation, San Diego CA, Society for Computer Simulation, Jan 1998. 64

[Fleurey 06] Fleurey Franck. *Langage et méthode pour une ingénierie des modèles fiables.* Thèse de Doctorat, Université de Rennes 1, France. 2006. 34

[Hudak 98] Hudak. P. *Modular domain specific languages and tools.* In P. Devanbu and J. Poulin, editors, Proceedings of the Fifth International Conference on Software Reuse (JCSR'98),pages 134-142. IEEE Computer Society, 1998. 17

[Janneck et Esser 01] Janneck. Jorn W. and Esser. Robert. *A predicate-based approach to defining visual language syntax.* In Symposia on Human-Centric Computing, IEEE Computer Society, 2001. 31

[Janssens et Rozenberg 80] Janssens, D. et Rozenberg, G. *On the Structure of Node-label Controlled Graph Languages.* Information Sciences 20, 191-216, 1980. 33

[Joines et Roberts 96] Joines, J. A et Roberts, S. D. *Fundamentals of Object-Oriented Simulation* Proceedings of the 1998 Winter Simulation Conference, pp. 141-149, December 13-16, 1998. 63

[Jouault et Bézévin 06] Jouault F. et Bézivin J. *KM3 : a DSL for Metamodel Specification.* Proceedings of 8th IFIP International Conference on Formal Methods for Open Object-Based Distributed Systems, Bologna, Italy, p. 171-185, 2006a. 34

[Kang et al 90] Kang, K. C, Cohen, S. G,Hess, J. A,Novak, W. E et Peterson, A. S. *Feature-oriented domain analysis (FODA) feasibility study.* Technical Report CMU/SEI-90-TR-21, Software Engineering Institute, Carnegie Mellon University, 1990. 19

[Kelly et Tolvanen 08] Kelly, Steven. et Tolvanen, Juha-Pekka. *Domain-Specific Modeling : Enabling Full Code Generation* John Wiley and Sons, Hoboken, New Jersey, 2008. 17, 20

[Kilgore 00] Kilgore, R. A. *Silk, Java and Object-Oriented simulation.* Proceedings of the 2000 Winter Simulation Conference, pp 246-252. December 2000. 57, 64

[Kleppe 06] Kleppe Anneke. *MCC : A Model Transformation Environment.* Proceedings of the First European Conference Model Driven Architecture. Foundations and Applications (ECMDA-FA), pp 173-187, Bilbao, Spain, juillet 2006. 29

[Klint 93] Klint, P. *A Meta-Environment for Generating Programming Environments.* ACM Transaction on Software Engineering Methodology Vol 2, April 1993. 19

[Kremer 97] Kremer Robert. *Constraint Graphs : A Concept Map Meta-Language.* PhD dissertation. Calgary, Alberta : University of Calgary, 1997. 15, 16

[L'Ecuyer 98] L'ecuyer Pierre. *Uniform Random Number Generator.* Proceedings of the 1998 Winter Simulation Conference, pp 97-104. December 1998. 67

[L'Ecuyer 99] L'ecuyer Pierre. *Good parameters and implementations for combined multiple recursive random number generators.* Operations Research, vol 47(1), pp 159-164. 1999. 67

[L'Ecuyer et al 02] L'Ecuyer P, Melian. L et Vaucher. J. *SSJ : A framework for stochastic simulation in Java.* Proceedings of the 2002 Winter Simulation Conference, pages 234-242. IEEE Press, 2002. 64

[L'Ecuyer et Panneton 05] L'ecuyer Pierre et Panneton François. *Fast Random Number Generators Based on Linear Recurrences Modulo 2 : Overview and Comparaison.* Proceedings of the 2005 Winter Simulation Conference, pp 110-119. December 2005. 67

[Lambiotte et al 84] Lambiotte, J. G, Dansereau, D. F, Cross, D. R. et Reynolds, S. B. *Multirelational Semantic Maps.* Educational Psychology Review 1(4) : 331-367. 1984. 15

[Langlois et al 07] Langlois Benoît, Consuela-Elena Jitia et Jouenne Eric *DSL Classification.* Proceedings of the 7th OOPSLA Workshop on Domain-Specific Modeling. October 21-22 2007. 21

[Law et McComas 01] Law, A. M, et M. G McComas. *How to build valid and credible simulation models.* Édité par B. A. Peters, J. S Smith, D. J Medeiros et M. W Rohrer. Proceedings of the 2001 Winter Simulation Conference. 2001. 22-29. 38

[Lazowska et al 84] Lazowska, E. D et al. *Quantitative System Performance : Computer System Analysis Using Queueing Network Model.* Prentice-Hall, Inc. 1984. 78

[Ledeczi et al 01] Ledeczi A., Maroti M., Bakay A., Karsai G., Garrett J., IV C. T., Nordstrom G., Sprinkle J. and Volgyesi P. *The Generic Modeling Environment.* Workshop on Intelligent Signal Processing, Budapest, Hungary, May 17, 2001. 34, 90

[Little 99] Little, M. C. *The JavaSim User's Manual.* Department of Computing Science, University of Newcastle upon Tyne, 1999. 64

[Lukose 93] Lukose , D. *Executable Conceptual Structures.* Édité par G. W Mineau, B Moulin et J. F Sowa. Conceptual Graphs for Knowledge Representation (Springer-Verlag), 1993. 6

[Macnamara 82] Macnamara, J. *Names for things : A study of human learning.* M.I.T. Press. Cambridge : Macnamara, 1982. 3

[Marriott 94] Marriott, K. *Constraint Multiset Grammars.* Procs. IEEE Symposium on Visual Languages, 118-125. 1994. 33

[Mernik et al 03] Mernik, M., Heering, J. et Sloane, A.M. *When and how to develop domain-specific languages.* Report Software Engineering SEN-E0309, Stichting Centrum voor Wiskunde en Informatica (CWI), Amsterdam, the Netherlands, December 8, 2003. 17, 19

[Miller et al 98] Miller, J. A, Ge, Y. et Tao, J. *Component Based Simulation Environments : JSIM as a Case Study Using Java Beans.* Proceedings of the 1998 Winter Simulation Conference, pages 373-381, Washington DC. 63, 64

[Minas 97] Minas, M. *Diagram Editing with Hypergraph Parser Support.* Procs. 13th IEEE Symposium on Visual Languages, Capri, Italy, 226-233, 1997. 33

[Minsky 68] Minsky Marvin. *Matter, mind, and models.* Semantic Information Processing, pages 425-432, 1968. 25

[Mosses 90] Mosses P. D. *Denotational Semantics.* Handbook of Theoretical Computer Science, Volume B : Formal Models and Sematics (B), p. 575-631, 1990. 35

[Musselman 92] Musselman, K. J. *Conducting a successful simulation project.* Édité par J. J Swain, D Goldsman, R. C Crain et J. R Wilson. Proceedings of the 1992 Winter. Piscataway : IEEE, 1992. Pp 115-121. 38

[Myers 90] Myers, B. A. *Taxonomies of Visual Programming and Program Visualization.* Journal of Visual Languages and Computing 1, n° 1 (1990) : 97-123. 14, 15

[Najork et Kaplan 93] Najork, M. A et Kaplan, S. M. *Specifying Visual Languages with Conditional Set Rewrite Systems.* Procs. IEEE Workshop on Visual Languages, 12-18, 1993. 33

[Novak et Canas 06] Novak, J. D, et Cañas, A. J. *The Origins of the Concept Mapping Tool and the Continuing Evolution of the Tool.* Information Visualization Journal, 2006 : 175-184. 2, 3, 4

[Novak et Canas 08] Novak, J. D, et Cañas, A. J. *The Theory Underlying Concept Maps and How to Construct and Use Them.* 2008, éd. Technical Report IHMC CmapTools 2006-01 Rev 01-2008. 4, 5

[Novak et Gowin 84] Novak, J. D et Gowin, D. B. *Learning How To Learn.* New York : Cambridge University Press, 1984. 6, 7

[Novak et Wandersee 91] Novak, J. D, et Wandersee, J. *special issue on concept mapping* Journal of Research in Science Teaching 28, n° 10 (1991) 5

[Nuutila et Törmä 04] Nuutila Esko et Törmä Seppo. *Text Graphs : Accurate concept mapping with well-defined meaning.* Concept Maps : Theory, Methodology, Technology Proceedings of the First International Conference on Concept Mapping. Édité par A. J. Cañas, J. D. Novak, F. M. González. Pamplona, Spain 2004 10

[Ormerod 01] Ormerod, R.J. *Viewpoint : the success and failure of methodologies - a comment on Connell.* Journal of the Operational Research Society 52, n° 10 (2001) : 1176-1179. 38

[Page et al 00] Page. B, Lechler. T et Claassen. S. *Objektorientierte Simulation in Java mitdem Framework DESMO-J (Object-Oriented Simulation in Java with the Framework DESMO-J,in German).* Libri Book on Demand, Hamburg, 2000. University of Hamburg, Faculty of Informatics. DESMO-J, 2006.Online http ://www.desmoj.de (in December 2006). 64

[Page et Kreutzer 05] Page. B et Kreutzer. W. *The Java Simulation Handbook : Simulating Discrete Event Systems with UML and Java.* Shaker Verlag GmbH, Germany (9 Nov 2005). 39, 43

[Paul et al 05] Paul, R, Eldabi, J. T, Kuljis, J et Taylor, S. J. E. *Is problem solving, or simulation model solving, mission critical ?* Édité par M. E Kuhl, N. M Steiger, F. B Armstrong et J. A Joines.Proceedings of the 2005 Winter Simulation Conference. 2005. 547-554. 38

[Pepper et Moore 01] Steve Pepper et Graham Moore. *XTM1.0, XML Topic Maps (XTM) 1.0 Specification* TopicMaps.Org. 2001, http ://www.topicmaps.org/xtm/1.0/ 10

[Prakken et Vreeswijk 02] Prakken, H. et Vreeswijk, G. *Logics for defeasible argumentation.* In D. Gabbay and F. Guenthner, editors, Handbook of Philosophical Logic, volume 4, pages 219.318. Kluwer Academic Publishers, Dordrecht, Netherlands, second edition, 2002. 8

[Rekers et Schürr 97] Rekers, J. et Schürr, A. *Defining and parsing Visual Languages with Layered Graph Grammars.* Journal of Visual Languages and Computing 8, 27-55, 1997. 33

[Rico 00] Rico Gabriel Lusser . *Writing the natural way.* New York : Tarcher ; Rev Sub edition, 2000.

[Risi 04] Risi Michele *Tools and Techniques for the Generation of Visual Environments.* PhD Thesis, University of Salerno, November 2004. 33

[Robinson 04] Robinson, Stewart. *Simulation : The Practice of Model Development and Use.* West Sussex : John Wiley & Sons Ltd, 2004. 38, 43

[Rothenberg et al 89] Rothenberg, J. Narain, S. Steeb, R, Hefley, C. et Shapiro, N. *Knowledge-Based Simulation : An Interim Report.* Technical report, The RAND Corporation, 1989. 55

[Sanchez-Ruiz et al 07] Sánchez-Ruíz, Arturo J. Saeki, Motoshi. Langlois, Benoît. Paiano, Roberto. *Domain-Specific Software Development Terminology : Do We All Speak the Same Language ?* The 7th OOPSLA Workshop on Domain-Specific Modeling. 18

[Sarantinos et Johnoson 91] Sarantinos, E. et Johnson, P. *Explanation dialogues : question desembiguation and text generation.* actes de la 11Ëme confÈrence Les systèmes experts et leurs applications, pp.109-122, Journèes Internationales d'Avignon, France, mai 1991. 55

[Shannon 98] Shannon, R. E. *Introduction to the art and science of simulation.* Édité par D. J Medeiros, E. F Watson, J. S Carson et M.S Manivannan. Proceedings of the 1998 Winter Simulation Conference. 1998. 7-14. 37, 38

[Schwetman 95] Schwetman, H. *Object-Oriented simulation modeling with C++/CSIM17.* Proceedings of the 1995 Winter Simulation Conference, pp. 529-533. December 1995. 63

[Smith 75] Smith, D. C. *PYGMALION : A Creative Programming Envoronment.* Stanford University, 1975. 14

[Soley et al 00] Soley, R. et l'équipe OMG. *Model-Driven Architecture.* OMG Document, novembre 2000. 24, 26

[Spinellis 01] Spinellis, D. *Notable design patterns for domain-speci ?c languages.* The Journal of Systems and Software, 56 :91-99, 2001. 17

[Taentzer et al 05] Taentzer, Gabriele. Ehrig, Karsten. Guerra, Esther. de Lara, Juan. Lengyel, Laszlo. Levendovszky, Tihamer. Prange, Ulrike. Varro, Daniel and Varro-Gyapay, Szilvia. *Model Transformation by Graph Transformation : A Comparative Study.* In Proc. Workshop Model Transformation in Practice, Montego Bay, Jamaica, October 2005. . 31

[Taha 09] Taha Walid. *Domain-Specific Languages.* International Conference on Computational and Experimental Engineering and Sciences, ICCES 25-27 Nov 2008. 17

[Taylor et al 95] Taylor, R. N, Tracz, W. et Coglianese. L. *Software development using domain-specific software architectures.* ACM SIGSOFT Software Engineering Notes, 20(5) :27-37, 1995. 19

[Teitelbaum et Reps 81] Teitelbaum, T and Reps, T. *The cornell program synthesizer : a syntaxdirected programming environment.* Communications of the ACM, 24(9) :563-573, Sept. 1981. 19

[Tewoldeberhan et Bardonnet 02] Tewoldeberhan, Tamrat W. et Bardonnet, Gilles. *An evaluation and selection methodology for discrete event simulation software.* Proceedings of the 2002 Winter Simulation Conference. 50, 51, 52

[Uskudarli et Dinesh 95] Uskudarli, S. M et Dinesh, T. B. *Towards a Visual Programming Environment Generator for Algebraic Specifications.* Procs. IEEE Symposium on Visual Languages, Darmstadt, Germany, 234-241, 1995. 24

[Van Der Zee et Van Der Vorst 07] Van der Zee, Durk-Jouke et Van der Vorst, Jack G.A.J. *Guiding principles for conceptual model creation in manufacturing simulation.* Proceedings of the 2007 Winter Simulation Conference. 44

[Van Emeren et al 96] Van Eemeren, F. H, Grootendorst, R. F. et Henkemans, F. S. *Fundamentals of Argumentation Theory : A Handbook of Historical Backgrounds and Contemporary Applications.* Lawrence Erlbaum Associates, Hillsdale NJ, USA, 1996. 8

[Van Gelder 03] Van Gelder Tim. *Enhancing deliberation through computer supported visualization.* In P.A. Kirschner, S.J.B. Shum, and C.S. Carr (Eds.), Visualizing argumentation : Software tools for collaborative and educational sense-making (pp. 97-115). New York. Springer. 2003. 9

[Vangheluwe et De Lara 02] Vangheluwe, H et De Lara, J. *Meta-models are models too.* Proceedings of the 2002 Winter Simulation Conference. (pp. 597-605). Eds : E. Yücesan, C.-H. Chen, J. L. Snowdon, and J. M. Charnes. 31

[Vangheluwe et De Lara 03] Vangheluwe, H et De Lara, J. *Computer automated multiparadime modelling : meta-modelling and graph transformation.* Proceedings of the 2003 Winter Simulation Conference. S. Chick, P. J. Sánchez, D. Ferrin, and D. J. Morrice, eds. 2003. 82

[Willemain 95] Willemain, T. R. *Model formulation : what experts think about and when.* Operations Research 43, n° 6 (1995) : 916-932. 38

[Wittenburg et Weitzman 98] Wittenburg, K. et Weitzman, L. *Relational Grammars : Theory and Practice in a Visual Language Interface for Process Modeling.* Visual Language Theory, Springer-Verlag, 193-217. 1998. 33