Institutional Repository - Research Portal
Dépôt Institutionnel - Portail de la Recherche
researchportal.unamur.be
University of Namur

THESIS / THÈSE

MASTER EN SCIENCES INFORMATIQUES

Définition de critères de comparaison des Ateliers de génie logiciel supportant le langage UML

Janssens, Stéphan

Award date:
2001

Link to publication

General rights
Copyright and moral rights for the publications made accessible in the public portal are retained by the authors and/or other copyright owners and it is a condition of accessing publications that users recognise and abide by the legal requirements associated with these rights.

- Users may download and print one copy of any publication from the public portal for the purpose of private study or research.
- You may not further distribute the material or use it for any profit-making activity or commercial gain
- You may freely distribute the URL identifying the publication in the public portal ?

Take down policy
If you believe that this document breaches copyright please contact us providing details, and we will remove access to the work immediately and investigate your claim.

Download date: 04. May. 2026

Facultés Universitaires Notre-Dame de la Paix, Namur
Institut d'Informatique
Rue Grandgagnage, 21, B-5000 Namur (Belgium)

« Définition de critères de comparaison des Ateliers de Génie Logiciel supportant le langage UML »

Stéphan Janssens

Mémoire présenté en vue de l'obtention
du grade de Licencié en Informatique

Année Académique 2000-2001

RÉSUMÉ

Ce travail a pour objectif d'établir une liste de critères permettant la comparaison d'outils de type « Ateliers de Génie Logiciel » qui supportent le langage U.M.L.

Ces critères sont classés par thèmes correspondants à des groupes de fonctionnalités.

Après avoir expliqué individuellement ces critères, le travail les synthétise sous forme d'une grille assez générale car ne tenant pas compte des spécificités de l'environnement dans lequel l'outil sera utilisé. Cette grille comporte également les paramètres qu'il faudra préciser pour obtenir une grille spécifique à l'environnement où l'outil sera utilisé.

Le travail reprend finalement deux grilles de critères correspondant à des environnements différents. A titre d'illustration, deux outils seront enfin évalués au regard de ces grilles.

The target of this work is to set-up a list of criteria which can be used to compare "Computer Aided System Engineering" tools that support the Unified Modelling Language.

The criteria have been grouped according to the different fields of functionality provided by such tools.

After explaining individually the different criteria, the work will be summarized in a grid that will not take into account the specificities of the environment where the tool will be used. We will specify the parameters that should be clarified to obtain a grid adapted to the environment where the tool will be used.

The work will, in the end, show two grids corresponding to different target environments for the tool. As an illustration, we will then evaluate two different tools with those grids.

AVANT-PROPOS

Je tiens à remercier ma collègue Monique Limbos, chargée du suivi de mon stage au sein de la Direction Informatique. Mon projet est venu s'ajouter à ses tâches habituelles alors qu'elle assumait plusieurs missions en parallèle.

Elle a su être disponible, attentive et critique. Sa connaissance des environnements techniques dans le domaine du développement de systèmes d'information m'a paru sans limite tangible.

Je remercie également mon Chef d'Unité, monsieur José Marin Navarro qui a su me fournir un cadre de travail adapté à la réalisation de mon mémoire. Chaque fois qu'il a pu me donner un coup de pouce en m'apportant un ouvrage de référence ou une coupure d'article, il n'a pas manqué l'occasion de le faire.

J'aurai également un petit mot pour les quelques collègues qui m'ont fourni des aides ponctuelles : Mario Adami pour les commandes de produits, Stefana Bruno pour mes problèmes de traitement de texte, Marleen Leunens pour les conseils concernant les styles dans Word et Antonio Sanchez Pareja, pour le prêt de l'une ou l'autre machine destinée aux tests des outils.

Je voudrais ensuite remercier Monsieur Dubois, d'avoir accepté d'être mon promoteur. Malgré notre emploi du temps fort chargé, les conseils promulgués suite à la lecture de mes ébauches se sont toujours révélés pertinents.

Sur un plan plus familial, je me dois de remercier mon épouse Cécile d'avoir accepté toutes les soirées d'exil dans mon bureau, où elle restait seule livrée aux griffes de nos trois enfants.

Je la remercie également d'avoir enduré la lecture de mon manuscrit, ce qui devait être un exercice particulièrement aride pour un non informaticien. Au même titre, je remercie deux amis qui ont accepté la tâche ingrate de la lecture : Serge Glibert et Daniel Philippe.

Pour conclure, je remercierai mes parents : ma maman d'avoir toujours donné la priorité à mes études, tout en respectant mes choix parfois surprenants, et mon papa de m'avoir plongé un peu accidentellement dans la marmite informatique à la glorieuse époque du TRS-80. C'est lui qui m'a appris à être à l'écoute de l'utilisateur et a fait naître une passion que je vis encore malgré qu'entre-temps j'en ai fait mon métier.

TABLE DES MATIÈRES

RÉSUMÉ 1

AVANT-PROPOS 3

TABLE DES MATIÈRES 5

GLOSSAIRE 9

TABLE DES ILLUSTRATIONS 13

1. INTRODUCTION 15

1.1. Contexte 15

1.1.1. Les spécificités de l'Orienté Objet (OO) 15

1.1.2. Ce qu'implique l'Orientation Objet sur la manière d'aborder le problème 15

1.1.3. Le cycle de vie Itératif. 16

1.1.4. Pourquoi UML ? 16

1.1.5. UML, un langage, pas une méthodologie 17

1.1.6. Rappel des différents diagrammes UML 17

1.1.7. Un peu d'histoire 18

1.1.8. Pourquoi un outil ? 20

1.2. Approche 20

2. DÉFINITION DES CRITÈRES D'ÉVALUATION DES OUTILS (ATELIERS DE GÉNIE LOGICIEL) 21

2.1. Support du langage 25

2.1.1. Version du langage 25

2.1.2. Diagrammes supportés 25

2.1.3. Stéréotypes 26

2.1.4. Extensibilité au type de projet 26

2.1.5. Extensibilité de l'outil (scripting) 27
2.1.6. Support d'autres notations 27
2.1.7. Vérification de la cohérence 28
2.2. La gestion des méta données 28
2.2.1. Documentation du schéma des méta données 28
2.2.2. Facilité d'accès aux données du métamodèle 28
2.2.3. Extensibilité du métamodèle 28
2.2.4. Possibilité d'exportation des méta données 29
2.2.5. Accessibilité simultanée au métamodèle 29
2.2.6. Notion de sites – réplication 30
2.2.7. Gestion des utilisateurs 30
2.2.8. Sauvegarde / Restauration du métamodèle 30
2.3. Gestion de la persistance 31
2.3.1. Persistance – gestion des objets persistants 31
2.3.2. Support des procédures stockées 31
2.4. Support et intégration de composants et architectures logiciels 32
2.4.1. Disponibilité de patterns, de modèles pour certains frameworks 32
2.4.2. Création de composants 33
2.5. L'outil et le cycle de vie du Projet 34
2.5.1. Contrôle de versions 34
2.5.2. La gestion du code 35
2.5.3. Génération de documentation sur base du projet 35
2.5.4. Gestion des exigences 37
2.5.5. Aide dans la mise en œuvre des tests 38
2.6. L'ergonomie offerte par l'outil 39
2.6.1. Propositions en fonction du contexte 39
2.6.2. Facilité de navigation 39

2.6.3. Adaptation de l'interface 39
2.6.4. Faire et défaire 40
2.6.5. Stable et performant 40
2.7. Support de différentes plate-formes 40
2.8. Aide à l'utilisation de l'outil 40
2.9. Ancrage de l'éditeur sur le marché 41
2.10. Coûts de l'outil 41
2.11. Méthodologie 42

3. UTILISATION DES CRITÈRES 43
3.1. Qualification des critères 43
3.2. Grilles d'Analyse 44
3.2.1. Grille générique 44
3.2.2. Grille spécifique 47

4. MISE EN PRATIQUE DE L'ÉVALUATION SUR DEUX OUTILS 65
4.1. L'outil « entreprise » : Rational Rose 2001 Entreprise 65
4.2. L'outil individuel : Argo UML 73

5. CONCLUSION 81

6. BIBLIOGRAPHIE 83
6.1. Ouvrages de référence 83
6.2. Sites Internet 83
6.2.1. UML 83
6.2.2. Spécifications XMI 83
6.2.3. Outils 83

ANNEXE 1 ILLUSTRATIONS DE L'ANALYSE DE RATIONAL ROSE ENTREPRISE 85

ANNEXE 2 ILLUSTRATIONS DE L'ANALYSE D'ARGO/UML 119

GLOSSAIRE

« Amigos »	Désigne les trois pères fondateurs d'UML : Grady Booch, James Rumbaugh, et Ivar Jacobson
A.G.L.	Atelier de Génie Logiciel – désigne un ensemble cohérent d'outils informatiques d'assistance dans toutes les phases du développement d'un système d'information
A.W.T.	Abstract Windows Toolkit – un ensemble d'objets Java permettant aux développeurs de créer des applications graphiques (avec des fenêtres, boutons, etc.…). Lié à Java 1.2.
Active-X	Un contrôle Active-X est l'équivalent d'une applet Java dans l'environnement Microsoft – un composant relativement indépendant qui peut faire partie d'un document composé, par exemple une page Web. L'Active-X peut être écrit dans de nombreux langages et est basé sur le modèle objet COM ou DCOM.
Artefact	Partie d'information utilisée ou produite lors du processus de développement d'un logiciel.
C.A.S.E. [Tools]	Computer Aided System Engineering – voir A.G.L.
COM	Component Object Model est la technologie Microsoft, permettant de développer par composants. Il a pour objectif de fournir les mêmes fonctionnalités que Corba. COM fournit des services de négociation d'interfaces, de gestion du cycle de vie des composants, d'enregistrement et de gestion des évènements, au sein d'une seule et même machine.
Corba	Common Object Request Broker Architecture pour créer, distribuer et gérer des objets répartis au sein d'un réseau. Il permet à des programmes s'exécutant à des endroits différents et développés par des équipes différentes de communiquer au travers d'un « Interface Broker ». Il est l'œuvre du consortium O.M.G.
D.D.L.	Data Definition Language – est un langage de définition permettant de maintenir le dictionnaire de données d'une base en y ajoutant, modifiant ou supprimant des éléments (tables, colonnes, contraintes…)

D.B.M.S.	Database Management System – un système qui permet à un ou plusieurs utilisateurs de créer et d'accéder à des données situées dans une base de données sans savoir où sont physiquement stockées les données et comment elles sont organisées. Le système assure également l'intégrité et l'accès concurrent aux données.
D.T.D.	Document Type Definition est une spécification issue de S.G.M.L. qui accompagne un document et identifie les différentes balises mises en œuvre et comment traiter l'information y associée. La définition des balises HTML constituent un DTD.
DCOM	Distributed Component Object Model – basé sur COM, cette technologie Microsoft ajoute un annuaire et d'autres fonctionnalités permettant aux composants de tourner sur des machines distribuées au sein d'un réseau
Dynamique [modélisation]	La modélisation dynamique du système a pour objectif de montrer l'interaction entre les objets du système, ou avec les systèmes extérieurs. Elle montre aussi l'évolution interne des objets.
E.J.B.	Entreprise Java Beans est une architecture permettant la mise en œuvre de composants écrits en Java et destinés à fournir des services. Pour déployer un EJB, il doit faire partie d'une application spécifique, appelée container.
Encapsulation	L'encapsulation est l'inclusion au sein d'un objet de tous les éléments nécessaires et suffisants à son fonctionnement – les données et les méthodes.
Ex-ante	Se dit d'une approche prédictive (estimant à priori)
Fonctionnelle [modélisation]	C'est une modélisation qui s'intéresse principalement au découpage du systèmes en fonctions. Les cas d'utilisation d'UML offrent une vue fonctionnelle du système
Forward engineer	Se dit de la technique qui consiste à partir de la modélisation pour générer du code.
Frameworks	Un framework est en un environnement mettant en œuvre des composants destinés à fournir des services dans un contexte donné.
Héritage	L'héritage est un concept qui permet aux objets d'hériter de certaines propriétés des objets qu'ils spécialisent
J.D.B.C.	Java DataBase Conectivity – un ensemble d'objets Java qui permettent de se connecter en Java à un grand nombre de base de données et de dialoguer avec celles-ci en SQL.

J2EE	Java 2 platform Entreprise Edition est une plate-forme Java destinée à offrir des services applicatifs orientés vers les grandes entreprises. J2EE est basée sur l'utilisation d'une couche client légère. Il met en œuvre des technologies basées sur des composants standardisés, modulaires et réutilisables.
O.C.L.	Object Constraint Language est un langage à expressions. Il permet d'exprimer des propriétés qui doivent être vérifiées dans les objets, en termes d'invariants
O.M.G.	Object Management Group est un consortium qui a été fondé en 1989 pour créer une architecture standard de composants distribués. Le premier « produit » de l'OMG sera Corba.
P.G.M.L.	Precision Graphics Markup Language est un langage pour la définition d'image 2D vectorielles. Il se base sur les modèles d'images présents dans Postscript et Portable Document Format
Patterns	Les patterns sont des abstractions qui décrivent des solutions récurrentes à des problèmes récurrents, complétés de multiples objectifs et contraintes
Persistance	Un mécanisme permettant aux objets du système d'information qui ne peuvent être éphémères de sauvegarder leur état dans un support externe à la mémoire (un SGBD par exemple)
PL/SQL	ProceduraL SQL est une extension procédurale (propriétaire à Oracle) au langage SQL.
Polymorphisme	Se dit de la capacité d'assigner des significations différentes à un opérateur en fonction de la classe d'objets sur lequel il est appliqué.
Prototypage	Le travail par prototypage consiste à développer des versions simplifiées de certaines fonctions du système qui seront validées ou invalidées en fonction de la réponse apportée par les utilisateurs.
R.D.B.M.S.	Relational Database Management System – est un programme qui permet de créer, mettre à jour et administrer une base de données relationnelle.
Reverse engineering	Le reverse engineering consiste à partir du code pour recréer les modèles. Cette technique est utilisée lorsqu'on est obligé de recourir à des composants dont on n'a plus de documentation.

Round-trip engineering	Le round-trip engineering consiste à générer du code à partir des modèles, à l'enrichir en dehors de l'AGL (par exemple en complétant le corps des méthodes) et à le réintroduire dans l'AGL afin de pouvoir continuer à faire évoluer les modèles tout en préservant le travail réalisé en dehors de l'outil
S.G.B.D.	Système de Gestion de Base de Données – voir D.B.M.S.
S.G.M.L.	Standard Generalized Markup Language – Langage normalisé permettand de décrire les relations entre le contenu d'un document sous forme électronique et sa structure (chapitre, paragraphes, table des matières…)
S.Q.L.	Structured Query Language est un standard interactif et de programmation pour récupérer de l'information d'une base de données et la mettre à jour. Bien que le langage soit standardisé ISO et ANSI, il existe de nombreuses extensions propriétaires au langage.
Statique [modélisation]	La modélisation statique du système présente la structure et les relations entre les objets utilisés par le système.
Stéréotype	Le stéréotype est un « qualificateur » qui joue un rôle prépondérant dans le mécanisme d'extensibilité d'UML : il permet de faire porter à un élément existant du langage une signification légèrement différente.
Stubs	Le Stub est une routine qui se substitue à un programme plus long, parce que ce programme n'est pas chargé en mémoire où qu'il sera exécuté sur une machine distante. Le stub accepte les requêtes provenant d'un programme client et les transmet au programme qu'il représente. Ce dernier transmet le résultat de son exécution au stub qui le remet au programme appelant.
Swing	Un ensemble de composants Java qui permettent aux développeurs de créer une interface graphique pour leurs applications (avec des boutons, ascenseurs etc.…). Swing est indépendant de l'interface graphique liée au système d'exploitation. Lié à Java 1.3.
Template	Un template est une sorte de « patron » que l'on peut appliquer quand on souhaite réaliser quelque chose utilisant des composants présents dans le « patron ».
Traçabilité	La traçabilité est la capacité à retrouver, tout au long du projet, la traduction des exigences du système au sein des artefacts

X.M.I.	X.M.I. est l'application de la technologie XML à l'échange de données entre des outils CASE.
X.M.L.	eXtensible Markup Language permet de créer des formats d'information communs et de partager à la fois le format et le contenu sur un réseau.

TABLE DES ILLUSTRATIONS

Figure 1 Le cycle de vie itératif 16

Figure 2 Evolution d'UML 19

Figure 3 Graphique illustrant les critères regroupés 23

Figure 4 Un nombre limité de diagrammes dans visio 2000 25

Figure 5 Rose 2000 s'adapte à de nombreux environnements 26

Figure 6 Exemple de script pour Rose 2000 27

Figure 7 Possibilité d'alterner entre différentes notations dans Rose 2000 27

Figure 8 Le site de patterns de together - http://www.togethercommunity.com 33

Figure 9 Garder à l'esprit l'ensemble des exigences tout au long du projet 38

1. INTRODUCTION

1.1. Contexte

La Commission Européenne a, comme beaucoup d'entreprises [publiques ou privées], intégré des environnements de développement d'applications Orientés Objets sans adapter ses méthodes d'analyse et de conception.

Elle a donc continué à utiliser des méthodes fonctionnelles et les outils (Ateliers de Génie Logiciels – « CASE tools ») qui les supportent..

Cette approche a progressivement montré ses limites. Son caractère fonctionnel ne permet pas de travailler au niveau d'abstraction requis pour obtenir un bon modèle Objet (par exemple, l'étude des structures de données et des traitements est fort dissociée). De plus, la méthode et les outils associés sont peu adaptés à une interaction plus forte avec l'utilisateur, à un travail par prototypage.

Comme les méthodes sont relativement difficiles à mettre en œuvre sans le support d'outils adaptés, un projet de sélection d'un outil CASE basé sur UML a été lancé. C'est dans ce cadre que s'est inscrite la présente étude.

1.1.1. Les spécificités de l'Orienté Objet (OO)

L'approche objet est basée sur cinq concepts fondateurs : les objets (et l'encapsulation), les messages, les classes, l'héritage et le polymorphisme.

« L'objet est une unité atomique formée de l'union d'un état et d'un comportement. Il fournit une relation d'encapsulation qui assure à la fois une cohésion interne très forte et un faible couplage avec l'extérieur. L'objet révèle son vrai rôle et sa vraie responsabilité lorsque, par l'intermédiaire de l'envoi de messages, il s'insère dans un scénario de communication. » [MULLER1997a].

1.1.2. Ce qu'implique l'Orientation Objet sur la manière d'aborder le problème

La « réutilisabilité des objets » et leur stabilité face à l'évolution des besoins, avancés comme les arguments majeurs en faveur des approches objets (pour le gain de temps qu'elles apportent) ne sont réellement praticables que si les objets ont été conçus en préservant une forte cohésion interne et un faible couplage avec l'extérieur. Pour pouvoir garantir ces caractéristiques, il est nécessaire de pouvoir analyser et concevoir le système sous une perspective objet.

Historiquement, les méthodes fonctionnelles se sont imposées. Elles transposaient dans une méthodologie la séparation des données et du code inhérente à l'architecture physique des ordinateurs dans leur mode de travail.

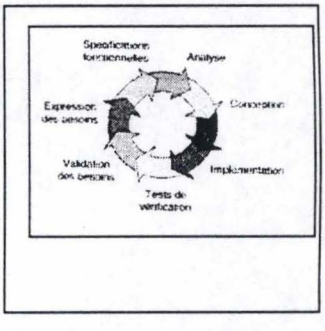
L'apparition de méthodes Orientées Objets s'est d'abord faite dans la phase de conception. Les informaticiens ont alors pris l'habitude de faire précéder celle-ci d'une analyse fonctionnelle. Malheureusement, cette manière de faire « manque très souvent d'abstraction et se limite à l'encapsulation des objets de bas niveaux, disponibles dans les environnements de réalisation et d'exécution » [MULLER1997a].

Aujourd'hui de [trop] nombreuses méthodes d'analyse et de conception prennent en charge l'entièreté du Cycle de développement Objet. C'est peut-être cette abondance qui a laissé les informaticiens perplexes et les a maintenus dans une approche classique mais peu adaptée au travail par Objets.

1.1.3. Le cycle de vie Itératif.

Les cycles de vie traditionnels (en cascade, en V, ...) ne sont, pour la plupart, pas itératifs. Il est donc difficile de gérer les changements (ou les ajustements) qui surgissent en cours de projet, ou dans le cadre de la maintenance. La modularité intrinsèque à une approche Orienté Objet permet plus facilement une approche itérative. L'implémentation est alors suivie par des tests et une validation des besoins qui peut déclencher une nouvelle expression des besoins si le produit n'est pas complet ou satisfaisant (qu'elles qu'en soient les raisons).

Figure 1 Le cycle de vie itératif



Le système peut donc être construit par morceaux (incréments) soit en réalisant une partie des fonctionnalités et en étendant les fonctionnalités couvertes au fur et à mesure, soit en raffinant les fonctionnalités déjà implémentées. Les composants déjà réalisés peuvent être présentés aux utilisateurs, à la manière du prototypage (alors que dans les cycles traditionnels, on attend plutôt la fin pour lever le voile sur l'ouvrage...). Ce cycle de vie intègre la maintenance (corrective ou évolutive).

Les avantages d'un cycle itératif sont : « La complexité n'est jamais insurmontable ; Un feed-back est généré très tôt, parce que l'implémentation se déroule rapidement pour une petite partie du système » [LARMAN1997]

1.1.4. Pourquoi UML ?

Une bonne modélisation objet nécessite, nous l'avons vu plus haut, un bon niveau d'abstraction. Les diagrammes qu'UML propose permettent de travailler à des niveaux d'abstraction différents. Une des forces d'UML est que les diagrammes restent les mêmes, quel que soit le niveau auquel on visualise les concepts. UML est également indépendant du langage de

programmation, du domaine d'application, ce qui en fait un langage universel.

UML facilite une implication des utilisateurs du système dans la construction d'un système parce que le caractère graphique du langage facilite son utilisation comme outil de communication. Tout en étant simple, UML a néanmoins été formalisé (grâce à son métamodèle), ce qui limite son ambiguïté.

On peut dire qu'UML tente de trouver un juste milieu entre l'accessibilité d'une représentation informelle et la rigueur d'un langage formel.

UML permet de représenter un système sous plusieurs vues complémentaires (statiques, dynamiques). Lorsqu'elles sont combinées, ces différentes vues offrent une vision complète du système. Les éléments communs et les zones de recouvrement facilitent l'assemblage des différents diagrammes.

1.1.5. UML, un langage, pas une méthodologie

« UML est qualifié de langage de modélisation, pas de méthode. La plupart des méthodes sont composées, en principe, d'un langage de modélisation et d'un processus. Le langage de modélisation est la notation (principalement graphique) que les méthodes utilisent pour exprimer les concepts. Le processus est ce qu'elles conseillent comme phases à enchaîner durant la conception. » [FOWLER1997].

1.1.6. Rappel des différents diagrammes UML

Diagramme Cas d'utilisation – « Use Cases »

Les cas d'utilisation (ou "Use Cases") mettent en évidence l'interaction entre l'utilisateur et le système d'information. Ils permettent de capturer des éléments essentiels dans la définition des besoins.

Diagramme de classes

Le diagramme de classes est un élément incontournable de l'orientation objet: il permet de définir les classes, leurs regroupements, et leurs relations qui sont à la base même de l'approche « objet ».

Diagrammes d'Objets

Appelés aussi diagrammes d'instance, ils montrent une instanciation du diagramme de classes. Ils sont utilisés pour illustrer la mise en œuvre d'une structure de données complexes (par exemple récursive).

Diagramme de séquence

Les diagrammes de séquence permettent de montrer la collaboration entre les objets, en mettant l'accent sur l'ordre (« séquence ») dans lequel cette interaction a lieu.

Diagramme de collaboration

Le diagramme de collaboration montre également la collaboration entre les objets mais l'accent est ici mis sur les relations entre les différents objets (l'ordre des messages passant au second plan).

Diagramme de paquetages – « Packages »

Ce diagramme sert à regrouper des éléments de la modélisation. Ils prennent toute leur importance dans des projets importants (par exemple lorsque les modèles ne tiennent plus sur une feuille A4).

Diagramme d'état

Les diagrammes d'état permettent de définir les différents états dans lesquels peuvent se trouver des objets donnés en fonction des cas d'utilisation où il se trouvent impliqués.

Diagramme d'activités

Les diagrammes d'activités permettent de mettre en évidence les responsabilités de différents composants collaborant au système (grâce aux couloirs de natation – « swimlanes ») ; par exemple, dans le cadre d'un cas d'utilisation.

Diagramme de déploiement

Ce diagramme permet de répartir les différentes composantes du système au sein de l'infrastructure (clients, serveurs…)

1.1.7. Un peu d'histoire

[d'après MULLER1997]

Les premières méthodes d'analyse d'un système apparaissent dans les années 1970 sous la forme de découpes fonctionnelles (hiérarchie de fonctions).

L'analyse est centrée sur les traitements. Dans les années 1980, des méthodes comme Merise voient le jour. Celles-ci privilégient une modélisation séparée des données et des traitements.

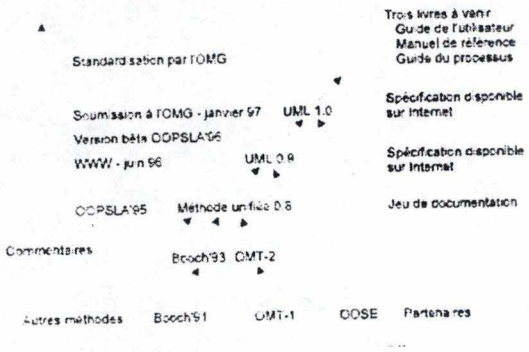
Au niveau de la programmation, Simula est le premier langage à amener les principes fondateurs de la programmation objets en 1967.

Au début des années 1990, on prend conscience de l'inadaptation d'une méthode séparant l'analyse des données de celle des traitements à déboucher sur une bonne conception orientée objet. Plus d'une cinquantaine de méthodes orientées objet voient le jour. Aucune ne se détache vraiment.

Au fil du temps, certaines notions semblent se dégager de ces méthodes ; notions qu'on retrouvera dans les méthodes portées par les fondateurs d'UML :

- OMT (James Rumbaugh) apporte les notions de classe et d'associations, les vues statiques, dynamiques et fonctionnelles d'un système.
- OOD (Grady Booch) apporte la partition en sous-systèmes, les vues logiques et physiques du système, la notion de package.
- OOSE (Ivar Jacobson) apporte l'expression des besoins à partir de l'interaction entre le système et l'utilisateur [use cases].

Figure 2 Evolution d'UML



L'évolution de ces méthodes (dans un premier temps OMT2 et Booch3) vont être telles que les différences vont s'amenuiser.

Booch et Jacobson vont, dès lors, décider de travailler ensemble à une méthode unifiée (appelée tout d'abord Unified Method) qui verra le jour dans sa première version (0.8) en 1995. Ils sont rejoints un an plus tard par Ivar Jacobson, ce qui donnera naissance à la version suivante (0.9 & 0.91). A partir de ce moment, le nom sera « Unified Modeling Language »). D'autres révisions ont suivi depuis pour aboutir à la version actuelle (1.4).

1.1.8. Pourquoi un outil ?

Maintenir la cohérence entre les différents artefacts [documents et produits (code logiciel,...)] composant un projet informatique peut être une tâche fastidieuse ; les outils C.A.S.E. (A.G.L. en français) les plus simples apportent au développement d'un projet informatique ce que le traitement de texte a apporté à la rédaction. L'effort à consentir pour maintenir la cohérence entre les différents artefacts sans ces outils est de plus en plus important au fur et à mesure de l'avancement de celui-ci. On peut aller jusqu'à dire que cette lourdeur contribue souvent à l'abandon de tout canevas méthodologique à un moment ou un autre du cycle de vie du projet (par exemple pour la maintenance).

Il faut noter que l'adoption des outils a un coût élevé, qu'on peut chiffrer (d'après [SOMMERVILLE1995]) à plus de deux millions trois cents mille francs belges (cinquante-huit mille €) par personne sur une période de cinq ans. Le calcul coût / bénéfice reste difficile à chiffrer mais il est certain qu'un approche itérative est difficile à pratiquer sans outil qui la supporte.

L'efficacité des outils C.A.S.E. est (toujours d'après [SOMMERVILLE1995]) variable selon le degré de formalisation de la phase qu'ils adressent. Ainsi, les phases d'ingénierie des exigences, de spécifications formelles et de maintenance ont toujours reçu un support plus faible au sein des outils parce qu'elles étaient peu formalisées.

1.2. Approche

Nous allons, dans la suite du texte, tenter de dégager des critères qui nous semblent pertinents dans la sélection d'un produit C.A.S.E. supportant le langage U.M.L. A cette fin, il nous a semblé important d'effectuer un bref rappel des différents diagrammes constituant le langage.

Nous exposerons ensuite les différents critères que nous avons retenus et les expliquerons. Sans que ce soit le seul regroupement possible, nous regrouperons les critères qui touchent à un thème commun.

Les critères seront ensuite regroupés au sein d'une grille que nous qualifierons de générique car devant être adaptée à l'environnement cible (celui où l'outil devra être utilisé).

Les critères seront ensuite instanciés afin de les adapter au contexte spécifique de l'entreprise (ou de l'individu) qui doit choisir un outil. Ce processus donnera naissance à une grille que nous qualifierons de « spécifique ».

Enfin, nous utiliserons nos critères pour approcher deux outils, dans des contextes différents, à des fins d'illustration. Nous considérons que la comparaison des différents outils ne fait pas partie de nos objectifs, encore moins la sélection d'un outil. Celle-ci est trop dépendante de la culture d'entreprise, de l'approche méthodologique choisie,...

2. DÉFINITION DES CRITÈRES D'ÉVALUATION DES OUTILS (ATELIERS DE GÉNIE LOGICIEL)

Nous avons exposé plus haut pourquoi les ateliers de génie logiciel nous semblaient importants et ce, particulièrement dans le cadre d'un cycle de vie itératif comme proposé par les approches orientées objet. Nous y avons également évoqué le fait que l'investissement (pas seulement initial et pas uniquement financier) pouvait être très important. Partant de là, il nous restait donc à identifier un ensemble de critères de sélection. Nous avons trouvé pertinent de les regrouper comme suit :

- Support d'UML : Tout ce qui est lié au langage, à ses versions, les différents modèles qui le composent, ses possibilités d'extension...

- (Méta-)Données : Les modes de stockage de l'information concernant les modèles (données sur les données), les diagrammes, les possibilités d'import / export de ces données, la problématique de leur accessibilité simultanée...

- Gestion de la persistance : Dans les systèmes modélisés au sein de l'outil, la capacité à mettre en relation des concepts objets avec des système de gestion de données qui souvent ne le sont pas.

- Support et intégration dans des frameworks, environnements... : Comme on construit rarement un système ex nihilo, le système doit pouvoir permettre la réutilisation tant de composants « maisons » que d'environnements « commerciaux ».

- Cycle de vie : Les phases de la vie du projet dans lesquelles ont été regroupées les versions du projet, le passage vers le code, la documentation, les tests et la traçabilité. Nous avons sous divisé ce groupe en « Gestion des versions », « Gestion du code », « Documentation », « Traçabilité » et « Gestion des tests ».

  Il nous faudra noter que des évolutions peuvent avoir lieu durant la vie d'un projet (au cours de ses cycles). Il s'en suivra une évolution des exigences qui impliquera des changements au niveau de la documentation, du code et des tests. La maintenance est également à prendre en compte dans ce cycle de vie.

  La gestion de ces changements est grandement facilitée par l'utilisation d'outils. Sans outils, le changement est beaucoup plus lourd à gérer.

- Ergonomie : Tout ce qui est lié à l'utilisation « agréable » de l'outil.

- Aide : Tout ce qui permet d'aider à l'utilisation de l'outil, surtout lors de la prise en main.

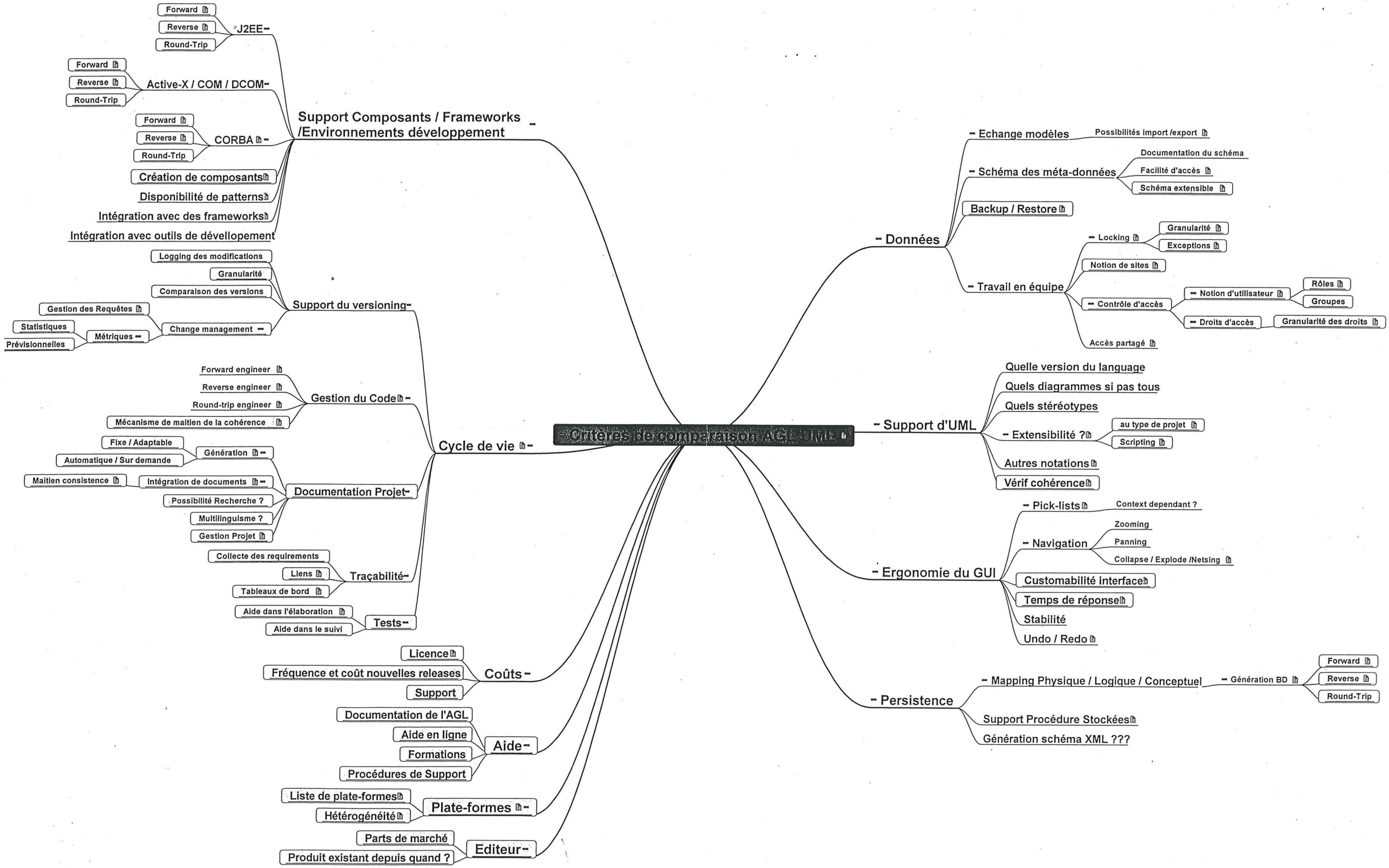
- Plates-formes : La disponibilité de l'outil dans des environnements techniques spécifiques.

- Coût : Les aspects qui touchent au prix d'acquisition, d'utilisation et de maintenance de l'outil.

- Editeur : Les questions d'ancrage de l'outil dans le marché.

Avant de détailler les critères, nous présentons, pour plus de clarté, leur regroupement sous forme graphique.

Figure 3 Graphique illustrant les critères regroupés



2.1. Support du langage

2.1.1. Version du langage

Le langage lui-même n'est pas figé. Il évolue en fonction des besoins ressentis par la communauté qui l'utilise. Le consortium O.M.G. fait évoluer le langage en invitant ses membres à faire des propositions d'améliorations.

Il est important que l'outil que l'on sélectionne soit donc en phase avec la version considérée comme courante du langage [pour l'instant, 1.3]. Ce critère est à considérer comme « indispensable ».

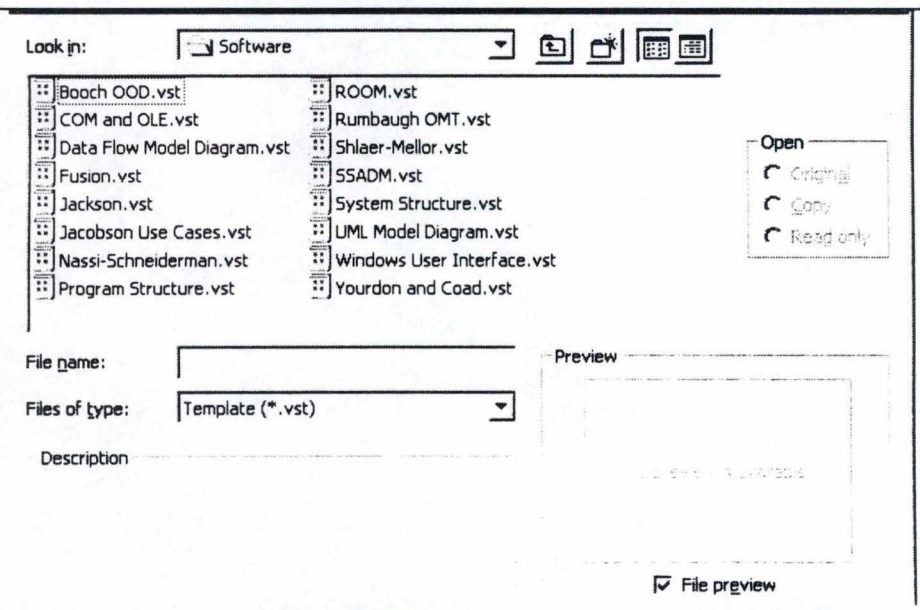
Il faut aussi que l'éditeur montre sa capacité à mettre à jour son produit quand la norme évolue. (Par exemple, ArgoUML – un produit « open source » – déclare utiliser une librairie spéciale de Novosoft pour implémenter le métamodèle d'UML... Ils peuvent donc s'adapter à de nouvelles versions du standard UML en quelques jours. Paradoxalement, certaines options avancées des diagrammes UML ne sont pas implémentées...).

L'utilisation d'O.C.L. permet d'ajouter une notation plus formelle dans certains diagrammes. Son utilisation a été rendue possible dans la version 1.1 d'UML » [LOPEZ1998]. La version actuelle des spécifications du langage UML inclut la formalisation de la syntaxe O.C.L. La sémantique n'étant pas encore formalisée, l'utilisation faite par les outils reste encore limitée.

2.1.2. Diagrammes supportés

Certains outils qui annoncent supporter U.M.L. n'implémentent pas l'ensemble des 9 diagrammes (par exemple, Visio 2000 ne supporte, dans sa version standard, que les Use Cases – appelés « Jacobson Use Cases » et le diagramme de classes – appelé « UML model diagram »).

Figure 4 Un nombre limité de diagrammes dans visio 2000



Le support d'un nombre limité de diagrammes restreint l'utilisation que l'on peut faire du produit pour des projets complexes (ArgoUML ne supporte pas dans sa version courante [0.8] le diagramme de séquence).

Certains produits sont d'ailleurs disponibles en version d'évaluation avec une limitation sur les diagrammes supportés (par exemple « Together Whiteboard 4.2 » ne supporte que la création de diagrammes de classes).

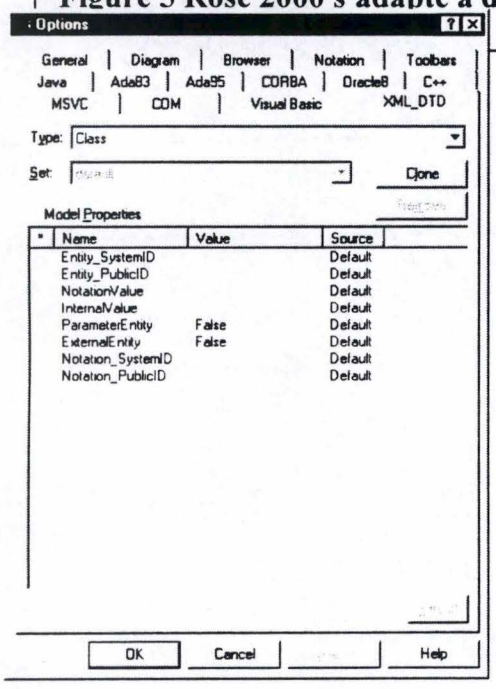
2.1.3. Stéréotypes

Le support des stéréotypes est intimement lié à la capacité du langage à exprimer de nouveaux besoins sans devoir spécialiser les diagrammes. Il permet d'enrichir le modèle en lui ajoutant une information que sa sémantique ne permet pas d'exprimer en standard (par exemple « signal », « interface » pour un diagramme de classes). De nouveaux stéréotypes peuvent être créés par l'utilisateur si le besoin s'en fait sentir. C'est donc une sémantique puissante qu'il faut bien maîtriser pour ne pas l'utiliser inutilement. Leur support est, sans nul doute, important ; reste à savoir ce que leur utilisation dans l'outil permet (simple texte ou élément à part entière ayant des conséquences sur la génération de code ou la vérification de la cohérence) ?

2.1.4. Extensibilité au type de projet

La capacité du langage à pouvoir s'adapter à de nouveaux contextes de développement (sites webs, XML, ...) est un avantage intrinsèque d'UML. Toutefois, si un produit a été trop dirigé vers la production de code pour un environnement donné, il est difficile de l'utiliser dans de nouveaux contextes. L'évolution récente des technologies de développement (le monde des applications basées sur la technologie Web) démontre cette nécessaire capacité d'adaptation.

Figure 5 Rose 2000 s'adapte à de nombreux environnements



2.1.5. *Extensibilité de l'outil (scripting)*

Au fur et à mesure de l'utilisation d'un outil quel qu'il soit, il est probable que l'utilisateur soit amené à souhaiter de petites améliorations du produit. La disponibilité d'un macro langage et la possibilité d'accéder à des données internes de l'outil (parcourir les classes, ...) permet d'enrichir celui-ci d'une manière aisée.

Il est souhaitable de pouvoir insérer ces petits modules dans les menus existants. La disponibilité de scripts exemples et/ou la possibilité de modifier des fonctions standards par l'intermédiaire d'un script sont des avantages certains (par exemple adapter la génération de code).

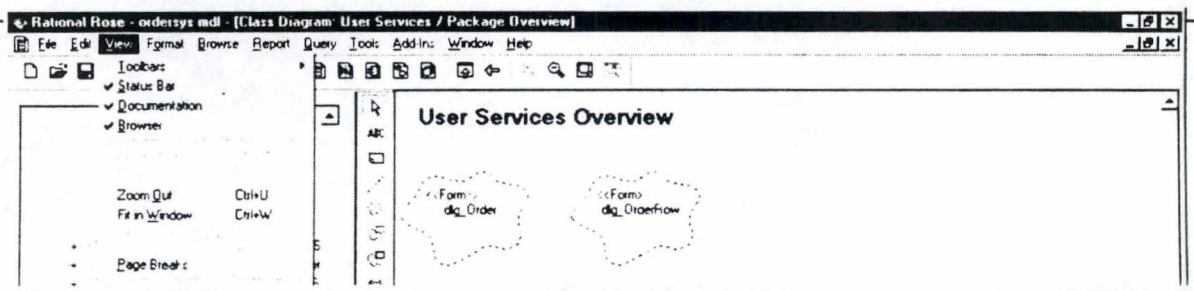
Figure 6 Exemple de script pour Rose 2000

```
' Select all and copy from the ViewPort, paste into text file,
import into EXCEL
' Writing directly to EXCEL is easy, but not worth the
trouble.

Sub Main
  Dim myClass As Class
  Dim myAttribute As Attribute

  ' On Error Resume Next  <- use this if needed

  ViewPort.Open
  ViewPort.Clear

  For i% = 1 To RoseApp.CurrentModel.GetAllClasses.Count
    Set myClass = RoseApp.CurrentModel.GetAllClasses.GetAt(i%)
    Print myClass.Name
    For j% = 1 To myClass.Attributes.Count
      Set myAttribute = myClass.Attributes.GetAt(j%)
      Print myClass.Name;",";myAttribute.Name
    Next j%
  Next i%
End Sub
```

2.1.6. *Support d'autres notations*

Si les utilisateurs du produit ont déjà pratiqué d'autres notations proches de la sémantique UML (Booch, ...), ou que l'entreprise travaille sur des projets avec des partenaires utilisant une telle notation, il peut être important de permettre d'autres vues d'un même modèle.

Figure 7 Possibilité d'alterner entre différentes notations dans Rose 2000



2.1.7. Vérification de la cohérence

Il est nécessaire que les diagrammes faisant partie d'un même projet soient cohérents entre eux. Comme il n'est pas évident pour le concepteur d'avoir une vue d'ensemble lorsque le projet est important, un outil est utile pour aider à découvrir les incohérences éventuelles.

Cet outil peut, par exemple, apporter une aide en vérifiant que les modèles ne font pas référence à des packages externes non définis.

2.2. La gestion des méta données

2.2.1. Documentation du schéma des méta données

Les méta données contiennent de l'information sur les entités manipulées dans les modèles (objets, attributs, opérations, ...). La structure des méta données doit être documentée et accessible. Cela permet à l'utilisateur d'extraire des informations de ce métamodèle.

Le script présenté plus haut dans l'illustration de l'extensibilité de l'outil (scripting) accède au métamodèle pour imprimer une liste des classes et de leurs attributs.

2.2.2. Facilité d'accès aux données du métamodèle

Le fait d'avoir une bonne documentation des informations contenues dans le métamodèle n'implique pas nécessairement une grande facilité à y accéder. La disponibilité d'un modèle objet des méta données accessible via le scripting de l'outil rend les choses plus aisées, dispensant par exemple d'accéder directement (en SQL) aux tables du repository.

2.2.3. Extensibilité du métamodèle

L'entreprise qui exploite l'outil pourrait ne pas trouver à l'intérieur du métamodèle toute l'information qui lui semble pertinente. Suivant le choix qui a été fait par le concepteur de l'outil, il est ou non possible d'étendre le schéma des méta données – en ajoutant, par exemple, des attributs à certaines classes.

Cette caractéristique est plus aisée à réaliser si l'outil se base sur un repository stocké dans une base de données, par exemple, un base de données relationnelle.

2.2.4. Possibilité d'exportation des méta données

La possibilité d'échanger les méta données entre différents outils est souhaitable car elle permet de ne pas perdre toute l'information concernant les projets réalisés sous un outil lorsqu'on décide de l'abandonner au profit d'un autre.

Il s'agit d'un cas typique d'échange d'information structurée (sauf pour les diagrammes mêmes qui transportent une information graphique). Le standard XML est donc le plus adapté à organiser ces informations.

Il faut toutefois associer un schéma standardisé à l'information XML pour permettre l'échange entre les différents outils. Ce format est défini au sein du MOF (« Meta Object Facility ») qui standardise la syntaxe abstraite des méta modèles.

XMI est, ainsi le format d'échange défini par l'O.M.G. pour l'échange de méta données. Ils se base sur XML et le MOF.

Le tableau suivant permet de clarifier les différents niveaux d'abstractions manipulés par les concepts présentés [OMG2000].

Niveaux d'abstraction	Termes MOF	exemples
M3	Méta-métamodèle	Le modèle MOF
M2	Méta-métadonnées	Le métamodèle UML
M1	Méta données	Des modèles UML
M0	Données	Le système modélisé

2.2.5. Accessibilité simultanée au métamodèle

Pour certains outils, aucune machine n'assume le rôle de serveur de stockage des méta données. Les données ne sont dès lors pas accessibles de tous les clients en même temps, mais d'un seul qui prend alors « possession » de cette partie du projet. Cette méthode est souvent moins lourde et, par conséquent, plus performante et facile à mettre en œuvre dans l'outil.

En contrepartie, cette méthode offre, en principe, une granularité plus faible lors du travail en équipe. Par défaut, c'est souvent le projet en entier qui est visé. Plus aucun membre de l'équipe de projet ne peut alors modifier celui-ci. Cette limitation est probablement acceptable dans des projets de petite taille ; elle le devient nettement moins sur des projets à large échelle.

Il faut pouvoir gérer des situations d'exception : un plantage de l'outil ou une coupure de courant peut arriver. Il doit alors être aisé de faire sauter les

verrous qui empêchent la modification du projet par deux personnes en même temps. Cette fonction peut être assignée à quelqu'un qui a un rôle d'administrateur du repository ou du projet.

2.2.6. Notion de sites – réplication

Certaines entreprises peuvent être amenées à réaliser des projets sur un ensemble d'implantations géographiquement éloignées disposant d'une faible interconnexion. Quelle que soit l'organisation du repository en groupe de fichiers ou dans une base de données, il est alors nécessaire que l'outil intègre une notion de site et permette une gestion de la réplication et du locking entre les sites.

2.2.7. Gestion des utilisateurs

Intégré ou non avec le système d'exploitation ou le SGBD, le repository qui doit gérer un accès concurrent doit se baser sur la notion d'utilisateur. Celle-ci s'accompagne idéalement de la notion de groupes d'utilisateurs.

Des droits d'accès doivent être définis (par exemple permettre uniquement la lecture, la modification, suppression, l'ajout de nouveaux éléments, entamer un nouveau cycle…). La granularité des droits d'accès, à savoir leur étendue (ensemble du projet, phases…) et la diversité des droits disponibles peuvent être importants, surtout pour des gros projets.

Le système doit alors permettre de définir des rôles au sein du projet (chef de projet, administrateur, réviseur, …). Ces rôles sont aux droits d'accès ce que les groupes sont aux utilisateurs.

Reste enfin, à pouvoir mettre en relation des rôles avec des droits d'accès et assigner ceux -ci à des groupes [d'utilisateurs].

2.2.8. Sauvegarde / Restauration du métamodèle

La disponibilité d'un outil spécifique de sauvegarde et de restauration du métamodèle peut être intéressant pour des projets de grande ampleur. Il serait alors possible de recharger dans le méta modèles une partie des informations qui s'y trouvait, sans en affecter les autres. Cette restauration sélective nécessiterait toutefois un contrôle de cohérence après sa mise en œuvre.

2.3. Gestion de la persistance

2.3.1. Persistance – gestion des objets persistants

Comme on utilise aujourd'hui peu de SGBD Orientés Objet, il faut pouvoir transformer les modèles (diagramme de classes dont certaines doivent être persistantes) en unités de stockage (par exemple tables relationnelles) adaptées au SGBD qui sera utilisé dans le système en cours de développement.

L'outil doit donc pouvoir générer un schéma de base de données. Elle peut le faire de manière directe (manipulation sur le dictionnaire des données du S.G.B.D.) ou par l'intermédiaire d'un script D.D.L. Quoi que ce dernier ne soit pas entièrement normalisé, cela reste la manière qui offre l'indépendance du S.G.B.D. au meilleur prix.

Comme dans le contexte de la génération de code, il y a lieu de se poser la question du type d'interaction entre le schéma qui évolue dans l'outil C.A.S.E. et l'élément externe qu'est la base de données :

- Forward engineering : il représente la génération, à partir des modèles contenus dans l'outil, d'un artefact. Dans ce cas-ci, il s'agit de la génération de la base de données, en passant ou non par un script D.D.L.

- Reverse engineering : il est l'opposé du précédent dans le sens où l'artefact existe mais le modèle qui en est l'expression n'existe pas. L'outil C.A.S.E. part alors de la base de données ou du script D.D.L. pour en dériver un modèle. Ce n'est évidemment pas un cas optimal et le modèle qui en résulte risque de manquer d'abstraction.

- Round-Trip engineering : il est la combinaison des deux interactions précédentes ; le modèle a débouché, par forward engineering, sur l'artefact. Celui-ci a évolué indépendamment du modèle et est remis en phase avec les modèles du projet par reverse engineering afin de permettre une nouvelle itération du cycle de vie objet.

Vu l'importance acquise par X.M.L. dans le développement de solutions impliquant la collaboration d'applications, l'outil apporte un plus s'il est à même de générer en parallèle à la Base de Données, son schéma X.M.L.

2.3.2. Support des procédures stockées

La base de données ne se cantonne pas aujourd'hui dans le rôle passif d'un simple container de données. Il contient de plus en plus souvent des fragments de code qui sont soit directement liés à la gestion des données que la base accueille (typiquement des triggers) mais aussi du code plus général (procédures stockées).

Ces fonctionnalités sont peu normalisées et on retrouve donc tant des langages propriétaires (PL/SQL chez Oracle) que des initiatives plus ouvertes (Java toujours chez Oracle dans les versions 'i').

Si les diagrammes de déploiement offrent, à priori, la possibilité de décrire comment (où) seront placés les composants, toute la partie génération de code ou de la persistance doit prendre en compte l'existence de fragments de code à cet endroit.

2.4. Support et intégration de composants et architectures logiciels

Les environnements dans lesquels les projets sont développés sont en constante évolution.

Certaines entreprises ont choisi de développer leur projets en se basant sur des modèles d'architectures logicielles telles Active-X, COM, DCOM, .NET ou, dans les monde Java, avec AWT, Swing, J2EE. D'autres environnements spécifiques nécessitent l'utilisation de Corba.

Tous ces environnements ont leurs spécificités conceptuelles qui nécessitent soit des modèles intégrant une série de concepts spécifiques – qu'on gagnerait à déjà être utilisables dans ces modèles (ne pas réinventer la roue) – soit un comportement spécifique de l'outil à leur égard (p.ex. forward / reverse / round trip engineering des stubs Corba).

Les environnements cités ci avant démontrent l'évolution constante des architectures logicielles. Opter pour un outil démontrant une bonne ouverture c'est donc protéger son investissement à long terme.

2.4.1. Disponibilité de patterns, de modèles pour certains frameworks

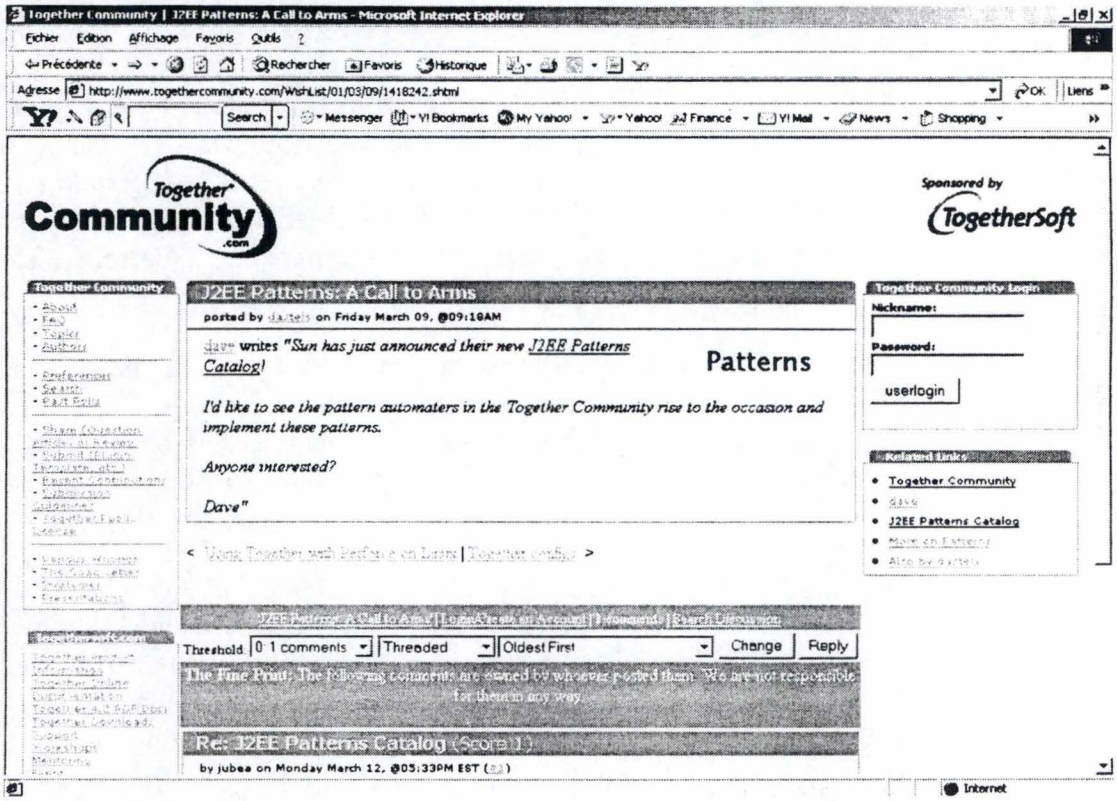
A côté des architectures logicielles, on retrouve également des sortes de bibliothèques de composants.

Elles peuvent prendre la forme d'un framework qui est une sorte de mécano permettant d'assembler des composants prédéfinis pour composer son application dans un certain domaine d'application.

Elles peuvent aussi, à un niveau plus abstrait, prendre la forme de patterns, sortes de schémas génériques qu'on peut retrouver dans des applications totalement différentes.

Certains outils ont développé, à côté de leur site internet dédié aux informations commerciales et de support, un site d'échange de tels éléments de modèles.

Figure 8 Le site de patterns de together - http://www.togethercommunity.com



2.4.2. Création de composants

S'il est intéressant de pouvoir disposer de « librairies » de patterns permettant d'intégrer des patterns standard dans ses applications (pour ne pas « réinventer la roue »), il est encore plus important de pouvoir extraire à son tour des systèmes d'information que l'on conçoit au sein de sa propre entreprise des concepts typiques de celle-ci qu'on puisse réutiliser dans d'autres projets.

Il faut donc que l'outil permette d'extraire certains morceaux des artefacts et de les regrouper au sein de librairies externes au projet courant.

Dans une grande entreprise, il faut pouvoir s'informer sur les librairies existantes via l'interrogation d'un conteneur central, une sorte de repository des composants.

On peut imaginer que ces repositories dépassent les frontières de l'entreprise, soit parce qu'on fait appel à de la sous-traitance soit si on souhaite aussi être partenaire d'une communauté dépassant le cadre de l'entreprise.

2.5. L'outil et le cycle de vie du Projet

2.5.1. Contrôle de versions

Le projet a une vie et l'ensemble des diagrammes ainsi que les documents, le code qui y est associé vont subir au long de cette vie des changements. Parfois, ces changements qui visent un mieux en termes de fonctionnalités ou de corrections de défauts se révèlent être insatisfaisants. Un système de contrôle de versions permet de revenir en arrière, en défaisant ainsi les changements apportés.

La mise en œuvre est souvent basée sur une action explicite de l'utilisateur qui déclare que le système est dans un état bien connu « A » et que les changements apportés par après permettent de déclarer le système dans l'état « B ». L'équipe de projet a, dès lors, tout le loisir de décider si les changements apportés pour passer de « A » à « B » étaient pertinents ou s'il est préférable de les défaire (revenir à l'état « A »).

Il ne faut pas confondre ceci avec un simple système de « undo » qui permet de défaire les modifications récentes (d'un niveau plus élémentaire). Ce type de fonctionnalité est ergonomiquement intéressant mais la signification sémantique de l'opération se pose dans un contexte bien plus élémentaire, sans rapport avec celle du projet.

En outre, ces systèmes permettent également de comparer deux états donnés du système et d'en déduire les différences. C'est alors à l'utilisateur d'apporter la signification de celles-ci dans la sémantique du projet.

2.5.1.1.La gestion des changements

Le projet donne généralement lieu à des demandes de changement, parce que les besoins ont été mal estimés ou que le contexte a évolué durant la construction du projet.

C'est un phénomène normal et inhérent à tout projet, quelle que soit sa taille. C'est d'ailleurs pour cette raison que le cycle de vie objet prend en compte la dimension d'évolution d'un système d'information après sa livraison aux utilisateurs.

Il est donc utile de posséder un système qui permette d'enregistrer chaque demande, d'accompagner celle-ci de documents en expliquant l'origine, la suite donnée et de mettre celle-ci, en cas de réalisation, en relation avec une version du projet.

Il est utile d'accompagner ces changements de données statistiques, coûts, nombre de modèles affectés, impact sur le code source, la base de donnée, coût d'implémentation.

Par ailleurs, l'existence de telles statistiques doit faciliter la capacité à estimer l'impact sur le projet et donc le coût que de nouveaux changements

impliqueront pour le système. Certains outils permettent d'ailleurs, d'assister à cette estimation ex ante.

2.5.2. La gestion du code

Pour l'instant, on n'échappe pas, en tous cas dans les systèmes d'information au sens classique du terme, à un codage dans un langage de développement. Il est donc utile de posséder un outil qui permette de travailler dans le langage utilisé par l'entreprise et plus particulièrement pour le projet. Comme dans le cadre de la gestion de la persistance, il y a lieu de distinguer le :

- Forward engineering : il consiste à produire, sur base des modèles exprimés dans le projet, du code dans le langage ciblé pour le développement du système d'information. Cet aspect doit donc tenir compte des choix technologiques établis par la société en la matière.

- Reverse engineering : Il consiste à intégrer au sein du projet des éléments de code pour lesquels il n'existe pas de modèles et qu'il est impensable de recréer de toutes pièces (manque de temps, de documentation, ...). Lorsque de tels cas sont amenés à se présenter, il faut que l'outil supporte les choix historiques de l'entreprise en terme de langages (d'environnements) de développement.

- Round Trip engineering : Suite à la génération de code par l'outil, le code est complété (la génération "forward" ne pouvant pas tout déduire des modèles). Il y a donc lieu, à l'itération suivante de resynchroniser le code avec le modèle car on souhaite naturellement préserver ce qu'on a ajouté entre deux itérations.

Pour permettre le « Round trip », les outils utilisent certaines méthodes permettant de faire la différence entre ce qui a été généré sur base des modèles par l'outil et ce qui a été ajouté par après. La connaissance de ces mécanismes est intéressante.

Il faut rester assez pragmatique et comprendre que la génération de code est, dans l'état actuel de l'art, une aide à la mise en œuvre (en-têtes des classes, certains éléments de contrôle...). L'outil CASE ne va pas supprimer la phase de mise en œuvre.

2.5.3. Génération de documentation sur base du projet

Si on recherche dans l'outil CASE un efficacité accrue, il ne faut toutefois pas négliger les moyens de communication à l'extérieur de l'équipe projet : il faut pouvoir donner accès à des artefacts appartenant au projet à des personnes qui ne font pas partie de l'équipe projet, qui ne sont pas utilisateurs de l'outil et ont peu d'intérêts à le devenir.

Il est donc utile de pouvoir définir des rapports types destinés à certaines catégories d'utilisateurs : les membres de la cellule de pilotage du projet qui se contenteront d'informations générales sur son avancement (quand tout va bien...), des utilisateurs plus concernés voudront d'autres catégories d'information.

Tous les outils ne permettent pas d'adapter ce que les rapports contiennent. Il faut parfois passer par un module additionnel.

Reste à préciser sur base de quel événement l'outil décide s'il faut mettre à jour les rapports : est-ce uniquement à la demande ou peut-on planifier une mise à jour journalière ou hebdomadaire.

Enfin, le format du rapport importe également : il est probablement avantageux de pouvoir disposer des rapports en format html, ou mieux, encore, que l'outil intègre la gestion d'un site web prenant en charge la communication extérieure du projet.

2.5.3.1.Intégration de documents externes dans le projet

Au sein du projet, il est intéressant de pouvoir maintenir le lien avec des documents « externes » tels des procès verbaux de réunions tenues dans le cadre du projet. Ces rapports se matérialiseront très probablement sous forme de textes, tableaux dans un format spécifique à l'outil utilisé pour leur rédaction.

Il faut que l'outil puisse intégrer ceux-ci tout en leur préservant une certaine dynamique : s'ils sont mis à jours en dehors de l'outil, il faut que le document présent dans l'outil reflète ces changements. Plus loin on notera l'utilité d'établir des liens entre des éléments présents dans l'objet externe et des objets internes à l'outil CASE (par exemple un diagramme ou une nouvelle version d'un diagramme qui est le résultat d'une décision prise en réunion...)

Il est utile que l'outil prévoie un système de recherche... cette recherche prendra t'elle en compte ces objets « étrangers » ou seront-ils « oubliés » ?

2.5.3.2.Multilinguisme de la documentation du projet

Un critère que peu d'outils seront à même d'intégrer est la dimension multilingue. Il peut être nécessaire de disposer de plusieurs versions linguistiques d'un même document, parce que les personnes qui doivent le lire ne parlent pas la même langue.

Il faut alors maintenir pour chaque artefact une information sur la langue dans laquelle il est rédigé, permettre de gérer la cohérence des versions (si on modifie l'anglais... alors il faut d'autres versions du français, ...).

Les diagrammes eux-mêmes devraient, pour bien faire, exister dans de multiples versions linguistiques...

2.5.3.3.Lien avec un système de gestion de projets

L'outil CASE n'intègre pas, en principe, des fonctionnalités de gestion de projet (affectation des ressources, durée des tâches, suivi de leur complétude, calcul des chemins critiques...). Si les documents produits par un tel système peuvent être intégrés dans l'outil, selon le principe décrit au paragraphe 2.5.3.1 ci-dessus, cette approche n'est pas la plus intéressante.

En effet, des activités qui sont initiées dans l'outil (nouvelle itération du cycle objet, nouvelles spécifications des « use cases ») ont clairement un impact sur le travail à réaliser, particulièrement si des estimations de charge de travail (nécessitée par un changement) sont disponibles dans l'outil, comme expliqué au paragraphe 2.5.1.1 ci-dessus. L'outil CASE pourrait donc offrir une meilleure disposition à collaborer avec un gestionnaire de projet classique.

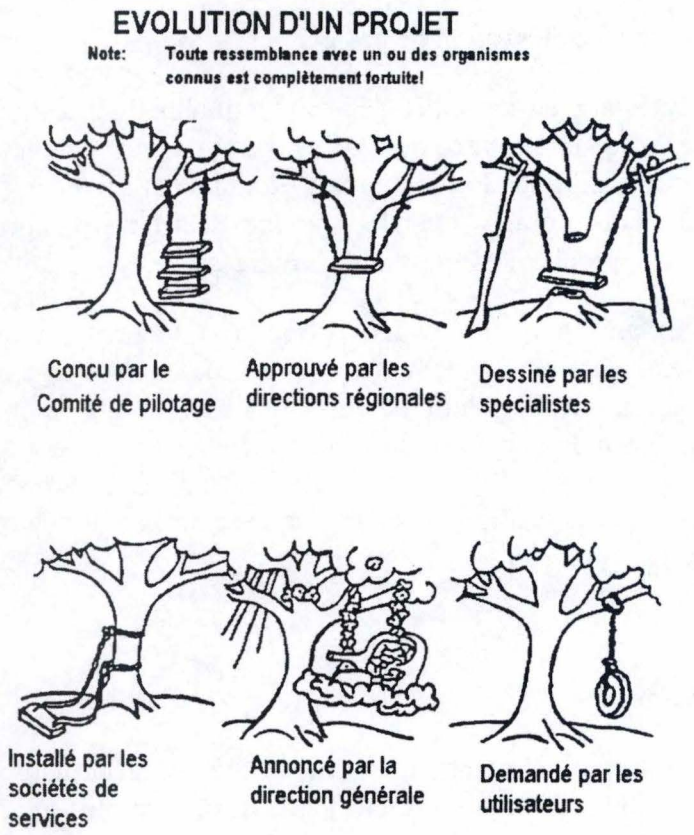
2.5.4. Gestion des exigences

Tous les cours d'Ingénierie du Logiciel abordent le thème de la distorsion des exigences du client tout au long du projet. Il est certain que plus le projet est important et plus l'équipe de projet est importante (ce qui va souvent de pair), plus il est difficile de garder à l'esprit les objectifs du projet qui tiennent à cœur à l'utilisateur du système.

L'outil CASE peut aider à garder à l'esprit ces exigences en offrant un système de gestion. Il convient donc de lister les exigences dans une partie ad-hoc de l'outil (avec d'autres informations comme la personne qui a exprimé cette exigence, sa portée, des liens éventuels vers des documents « externes » tels des procès verbaux de réunions,...) et de pouvoir associer des parties des artefacts du projet (par exemple des modèles) avec les exigences qu'ils mettent en œuvre.

Finalement, l'outil doit pouvoir générer des tableaux de bord permettant de vérifier la mise en œuvre des exigences répertoriées à chaque phase du projet. Ces tableaux peuvent revêtir la forme de tableaux dont l'abscisse liste les cycles et les phases du projet et l'ordonnée les différentes exigences.

Figure 9 Garder à l'esprit l'ensemble des exigences tout au long du projet



2.5.5. Aide dans la mise en œuvre des tests

Pour produire un système d'une bonne qualité, il est essentiel que le système soit testé. L'outil CASE peut apporter une aide dans deux aspects des tests : le suivi des procédures de tests et l'élaboration des tests eux-mêmes.

Le premier aspect est le plus aisé. En effet, pour assurer le suivi des tests, il faut, un peu à la manière de la gestion des exigences, pouvoir entrer la liste des tests à accomplir, les phases du projet concernées par ces tests.

D'autre part, il faut pouvoir signaler à l'outil qu'un test a été effectué et avec quel niveau de succès, en précisant éventuellement des remarques.

Finalement, une sorte de tableau de bord doit permettre de se rendre compte des tests restant à réaliser durant les différentes phases du cycle objet en cours et parmi les tests déjà effectués, lesquels le sont avec succès.

En ce qui concerne le second aspect, l'aide à la définition des tests, les outils peuvent se baser sur l'information contenue dans les diagrammes « cas d'utilisation » et « de séquence ». Les premiers permettent de capturer les différents scénarii d'utilisation, les seconds de définir les « contrats » concernant les messages échangés entre les objets.

2.6. L'ergonomie offerte par l'outil

2.6.1. Propositions en fonction du contexte

L'ergonomie est assez difficile à quantifier en terme de critères. D'autant qu'il est certainement possible de s'attacher à la définition de l'analyse de la tâche en long et en large, ce qui ne fait clairement pas partie des objectifs du présent document.

Néanmoins, le recours à des listes contextuelles lors de la réalisation de diagrammes est assez agréable. Par exemple, lorsqu'on souhaite préciser le type d'un attribut, avoir à sa disposition la liste des types scalaires qu'on enrichit d'autres types objets correspondant au modèle en cours de définition. De même lors de l'établissement d'un cardinalité, ...

2.6.2. Facilité de navigation

Un des problèmes communs à tous les outils CASE est la question de la navigation au sein des modèles. Au début, il est relativement aisé de visualiser l'ensemble des éléments du diagramme qui est en cours de visualisation. Au fur et à mesure que le diagramme se complète, l'écran devient trop exigu. Il faut pouvoir alors zoomer en avant et en arrière et faire du panning, déplaçant ainsi la portion qu'on visualise au sein de l'ensemble du diagramme.

Cependant, si ce mode de fonctionnement est suffisant pour un plan d'architecte, il peut se révéler inapproprié dans certains diagrammes. Pouvoir recourir à l'imbrication (nesting) de diagrammes est alors utile (par exemple un diagramme de séquence qui présente des opérations de haut niveau et où on peut obtenir une décomposition en opérations plus élémentaires en cliquant sur un des symboles d'une opération. C'est alors un autre diagramme de séquence qui apparaît, détaillant plus finement le séquencement de cette opération. Cette fonction ne s'applique qu'à certains diagrammes.

On peut également combiner cette opération avec des fonctions « collapse » et « explode » qui tantôt montrent, tantôt cachent les diagrammes des niveaux « inférieurs ».

Ces fonctions qui sont souhaitables dans le cadre du travail au sein de l'outil peuvent l'être tout autant lors de l'impression des diagrammes (imprimer une hiérarchie de diagrammes imbriqués).

2.6.3. Adaptation de l'interface

Comme nous avons tous nos habitudes de travail, il est important que l'interface puisse, à un certain degré, s'adapter à des choix posés par

l'utilisateur (et que ces choix soient stockés éventuellement au sein de la définition même de l'utilisateur au niveau du repository). On peut, par exemple, adorer ou détester les menus dont certaines options s'escamotent automatiquement. Il convient de laisser l'utilisateur en décider.

2.6.4. Faire et défaire...

Il est important de pouvoir annuler et répéter (annuler une annulation) un nombre raisonnable des dernières opérations réalisées. Comme cela a été expliqué au niveau de la gestion des versions, on parle ici d'opérations « atomiques » sur les objets qu'on manipule actuellement : tracer une association, ajouter un attribut ou une méthode à un objet mais pas d'opérations complexes qui ont une signification plus opérationnelle pour le système en cours de développement (« ajouter la possibilité de visualiser la photo de l'article commandé », ...).

2.6.5. Stable et performant

Ces critères sont pour le moins subjectifs et ne sont généralement pas documentés. C'est seulement à l'usage et sur des modèles d'une taille respectable que l'on pourra juger si l'outil est stable, c'est à dire qu'il ne disparaît pas sans crier gare, tout en perdant les changements réalisés, ce qui est une circonstance aggravante... C'est également à l'usage qu'on verra si le temps de réponse n'est pas trop affecté par le nombre d'objets présents dans le modèle (temps de chargement, de vérification de cohérence, ...) et qu'on peut donc qualifier le système de performant.

2.7. Support de différentes plate-formes

Il faut que l'outil soit disponible sur les stations de travail des membres du projet. Celle-ci pourraient être hétérogènes : certaines sous Unix, d'autres sous Windows. Il faut alors que l'outil offre un bon niveau d'interopérabilité : les éléments du projet doivent être accessibles indifféremment de l'une ou l'autre plate-forme, sans perte de contenu. Si cela est souhaité, le mieux est de procéder alors à quelques tests.

Une interopérabilité plus traditionnelle est celle du repository : les stations de travail exploitent toute la même plate-forme (Windows par exemple) mais le serveur qui doit servir de repository est une machine Unix sur laquelle tourne un SGBD Oracle. Ceci pose généralement moins de problèmes.

2.8. Aide à l'utilisation de l'outil

Les outils CASE sont complexes à utiliser, également parce que leur utilisation implique en même temps que l'apprentissage de l'outil, celui du

langage UML et, nous y reviendrons, un processus d'approche du problème : une méthodologie, en somme.

L'aide à l'utilisation de l'outil doit donc se baser sur une bonne documentation de référence, incluant si possible un ouvrage de type « tutorial ».

L'outil lui-même se doit de fournir une aide en ligne adaptée au contexte courant, c'est à dire contextuelle. Cette aide peut être efficacement complétée par un site Internet avec des « trucs et astuces » d'utilisations de l'outils sous formes de FAQ (« questions souvent posées »).

Le recours à un service de support efficace est également très utile, sans doute plus orienté vers des questions d'utilisation de l'outil pendant les premières semaines et ensuite seulement vers de la maintenance corrective.

Enfin, il y a lieu qu'à côté de l'outil, le vendeur offre un éventail de formations. Certains vendeurs proposent d'ailleurs de faire suivre celles-ci par la présence d'un consultant aidant à traduire les concepts vus au cours en actions dans son propre environnement de travail la semaine qui suit le cours.

2.9. Ancrage de l'éditeur sur le marché

L'achat d'un outil CASE est un investissement important, tant en coûts directs (voir ci-après) qu'en coûts indirects : la formation, l'apprentissage,... Il faut donc s'assurer de la pérennité d'un tel investissement. Sans que ce critère ne prenne une trop grande place, il faut donc voir depuis quand le produit existe et quelle part de marché il a acquis en comparaison aux autres outils CASE supportant UML.

2.10. Coûts de l'outil

Les coûts des licences de tels outils sont importants.

Certains ont une structure de prix assez complexe car l'offre est constituée de nombreuses options. Il faut donc veiller à comparer des choses comparables : si ont met de nombreuses fonctionnalités à l'avantage d'un produit comme Rational Rose, il faut additionner le coût des options ainsi sélectionnées.

Par ailleurs, il faut se demander à quelle fréquence l'éditeur propose de nouvelles versions de ses produits et quelles sont les conditions pour en disposer.

Enfin, quel est le coût du support (droit d'appeler un help desk, de disposer de correctifs...).

Ces deux derniers aspects vont souvent de pair (le support donne droit aux nouvelles versions).

2.11. Méthodologie

Nous dirions que l'aspect méthodologique est un aspect important que nous n'aborderons pas dans notre travail. Pourtant, lors de l'utilisation des outils, nous nous sommes rendus compte de l'importance de l'existence d'un processus de travail, d'une méthodologie.

Comparer les méthodologies objet n'était clairement pas dans les objectifs de ce travail. Certains éditeurs proposent d'utiliser leur outil avec une méthodologie propriétaire ; par exemple RUP (Rational Unified Process) chez Rational. Le choix de l'outil peut donc infléchir celui de la méthodologie... ou le contraire.

3. UTILISATION DES CRITÈRES

Après avoir identifié et explicité les critères qu'il nous semblait pertinent d'utiliser, nous aborderons maintenant comment les utiliser.

A cette fin, nous allons développer une grille regroupant nos critères d'évaluation. Cette grille devra ensuite être adaptée (nous dirons « instanciée ») à l'environnement d'utilisation cible.

3.1. Qualification des critères

Nous avons présenté ci-dessus une liste de critères génériques. Il importe maintenant d'en dériver une liste de critères spécifiques à son entreprise.

Pour arriver à cela, il convient d'abord de dériver certaines caractéristiques des critères.

Ainsi, tout d'abord, il faut se demander quelle importance relative possède chacun des critères :

- Indispensable / must have : l'outil qu'on souhaite sélectionner doit avoir ces fonctionnalités, sans quoi il ne sera pas sélectionné. Attention que cette approche peut parfois entraîner une trop grande sélectivité et déboucher soit sur la sélection d'outils très chers, soit sur l'inexistence d'un outil répondant à tous les critères requis. Il faut alors revoir la portée de ces critères et les transformer en hautement souhaitable / highly desirable

- Souhaitable / Nice to have : ces critères procurent un avantage au produit mais n'ont pas le caractère "hautement souhaitable" des critères précédents. On leur donnera toutefois une importance plus grande que les critères « normaux ».

- Utile : ces critères ne revêtent pas une importance particulière pour l'entreprise. On y prêtera peu d'attention.

- Inutile : les critères ne s'appliquent pas à l'environnement de l'entreprise. C'est un raccourci pour dire « non utile » qui serait sans doute une formulation plus adéquate.

Ensuite, les critères peuvent être explicites en eux-mêmes ou bien être trop génériques pour être évaluables.

Il en va ainsi d'une série de critères qu'il n'était pas utile de spécialiser lorsque nous les avons énoncés plus haut mais qui doivent l'être pour évaluer un produit (exemple : le forward, reverse ou round-trip engineering dans la génération de code – 2.5.2 concerne forcément un langage de développement – pas nécessairement le même dans les trois cas ; on ne

saurait évaluer la capacité d'un outil à faire du reverse sans préciser que c'est en Cobol ni faire du round trip sans préciser que ce sera en Java).

Ces constatations nous amènent à faire un travail complémentaire à celui de l'énoncé des critères : instancier la grille générique dont nous avons tracé l'ébauche au paragraphe 2 pour en établir l'importance et en préciser, lorsqu'il y a lieu les paramètres spécifiques.

3.2. Grilles d'Analyse

3.2.1. Grille générique

Partant du typage des critères énoncés ci-dessus, nous pouvons synthétiser ceux-ci sous forme d'une première grille.

On y indique si l'importance du critère est généralement la même quelle que soit l'entreprise (dans le cas contraire, il faudra préciser cette importance lors de l'instanciation de la grille) et lorsqu'il faut préciser certains paramètres d'environnement lors de l'instanciation.

Critère	Importance	Préciser
Langage		
Support de la version courante du langage	indispensable	Version (1.3 actuellement)
Support de tous les diagrammes	indispensable	
Extensibilité du langage (stéréotypes)	indispensable	
Possibilité de s'adapter à de nouveaux types de projets		
Extensibilité de l'outil (scripting)		
Support d'autres notations (OMT Booch)		Autres notations souhaitées
Outil de vérification de la cohérence		
Méta données		
Documentation du schéma des méta données		
Facilité d'accès au schéma des méta données		
Extensibilité du métamodèle		
Sauvegarde / Restauration intelligente du métamodèle		
Exportation XMI / PTL		
Importation XMI / PTL		

Critère	Importance	Préciser
Travail de groupe		
Accessibilité simultanée		
Notion de sites et de réplication inter sites		
Notion d'utilisateurs		
Notion de groupes d'utilisateurs		
Possibilité de droits d'accès sur le projet		
Notions de rôles		
Gestion de la persistance		
Mapping objet vers physique		SGBD cible
Support des procédures stockées		SGBD cible : type de procédures
Création / Modification de la base		
Forward		via SQL / en direct
reverse		via SQL / en direct
Round Trip		via SQL / en direct
Support et intégration dans des frameworks, travail par composants		
Disponibilité de modèles pour frameworks spécifiques		frameworks concernés
Disponibilité de patterns		
Support d'architectures		Corba / Com / Dcom ...
Intégration avec des environnements de développements		environnements concernés
Création de composants		
Cycle de vie		
Gestion des versions		
Journalisation des modifications avec points de reprises		
Comparaison de versions		
Gestion des requêtes de changement		
Analyse d'impact d'une modification		
Gestion du code		
Forward		Langage(s) cible(s)
Reverse		Langage(s) source(s)
Round Trip		Langage(s) de travail

Critère	Importance	Préciser
Gestion de la documentation		
Génération de documentation sur base du projet		format de sortie
Publication de la documentation vers un site Web du projet		
Adaptabilité du format de la documentation générée		
Intégration et synchronisationde documents externes dans le projet		format des documents
Multilinguisme dans la documentation du projet		langues visées
Lien avec un gestionnaire de projets		nom du gestionnaire
Gestion des exigences		
Collecte des exigences		
Liens entre les exigences et les artefacts du projet		
Tableaux de bord		
Gestion des tests		
Collecte des tests à réaliser		
Aide à la définition des tests (scénarii / harnesses...)		
Tableaux de bord d'avancement / succès des tests		
Ergonomie		
Propositions faites en fonction du contexte (pick lists...)		
Facilité de navigation (zooming, panning, nesting, ...)		
Adaptabilité de l'interface aux souhaits de l'utilisateur		
Fonctions défaire et refaire		
Stabilité		
Performance		
Plates-formes		
Tourne sur les plates-formes de l'entreprise		nom des plates-formes
Fonctionnement dans une configuration hétérogène (clients)		nom des plates-formes
Fonctionnement avec un repository sur une autre plate-forme		nom de la plate-forme (SGBD) du repository

Critère	Importance	Préciser
Aide à l'utilisation de l'outil		
Aide intégrée contextuelle (avec fonction de recherche)		
Documentation papier		
Tutorial		
Formation		
Editeur		
Quote-part du marché (%)		
Produit existe depuis (années)		
Coût		
Acquisition / utilisation des licences		
Coût des mises à jour		
Contrat de support		niveau de support à préciser
Méthodologie		
Existence d'une méthodologie d'utilisation de l'outil (maison ou autre)		
Adaptabilité		méthodologies souhaitées
Méthodologie maison disponible		

3.2.2. Grille spécifique

Nous avons choisi de montrer l'instanciation de la grille générique dans deux contextes différents :

- Le cas d'un outil visant le support de projets de grande envergure. L'outil visé sera donc multi utilisateurs, avec de nombreuses fonctionnalités et possibilités d'évolution.

- Un outil léger, permettant une approche personnelle du langage et de son utilisation avec un AGL. Ce dernier visera l'étudiant ou le professionnel au début de sa formation à UML.

3.2.2.1.Cas de l'outil d'entreprise

Nous allons instancier la grille énoncée au point 3.2.1 pour créer une grille adaptée à la situation générale telle que nous l'avons perçue au sein de la Commission Européenne par le biais d'un groupe de travail « outils UML » regroupant des représentants de nos différentes Directions Générales.

L'outil devra permettre le suivi de projets de bout en bout, offrir des mécanisme de vérification de la cohérence du projet et de l'adéquation des solutions proposées aux besoins exprimés.

Il devra permettre de documenter les projet, en détaillant les solutions effectivement mises en œuvre mais également les alternatives, les décisions prises et les motifs des choix effectués.

L'outil devra offrir les fonctions traditionnelles d'un outil CASE en facilitant la génération de code et les structures de persistance. Il devra le faire en respectant les choix technologiques de l'Institution (RDBMS Oracle, architecture distribuée J2EE, langage Java, client léger basé sur le browser et la production dynamique de pages HTML).

L'outil devra aider les concepteurs et les architectes à capturer les concepts essentiels et atteindre le niveau d'abstraction adéquat. Il permettra d'utiliser une méthodologie orientée objet couvrant l'ensemble du cycle de vie d'un projet, sans l'imposer.

L'outil sera utilisé pour communiquer des artefacts à des acteurs externes à l'Institution (par exemple des sous traitants chargés de la réalisation d'un Système d'Information). Il devra également pouvoir s'adapter à des projets non informatiques.

En traduisant ces buts de l'outil dans des caractéristiques correspondant aux critères évoqués avant, nous obtenons l'instanciation qui suit.

Le support de la dernière version du langage est obligatoire. Il démontre indirectement la capacité de l'éditeur du logiciel à suivre de près ce qui se fait dans la communauté UML.

Si le support de tous les diagrammes n'est pas strictement nécessaire pour « se faire la main », on ne peut s'en priver pour des projets réels où l'absence d'un type de diagramme pourrait devenir un handicap. Il en va de même pour les stéréotypes.

Dans le même ordre d'idées, on ne connaît pas les technologies qu'on utilisera dans deux ans ; il est donc souhaitable que l'outil puisse s'adapter à de nouveaux modes d'organisations de l'information, de nouvelles architectures et de nouveaux modes d'organisation des équipes de projet.

Le scripting pourrait être utile pour réaliser des fonctions « maison » qui ne seraient pas possibles en standard ou que l'on souhaiterait voir automatisées.

Il n'y a pas eu, auparavant, d'utilisation systématique d'une méthodologie Objet ni de recommandation dans ce domaine. L'adoption du langage UML est une première initiative dans ce sens. Le support d'autres notations ne semble pas utile car, même dans l'échange avec les fournisseurs, la

Commission est un client suffisamment puissant que pour insister sur l'utilisation d'UML.

Un outil de vérification de cohérence est une aide indispensable, puisqu'il permettra de nous aider à détecter nos erreurs de jeunesse...

Critère	Importance	Commentaires
Langage		
Support de la version courante du langage	Indispensable	1.3
Support de tous les diagrammes	Indispensable	
Extensibilité du langage (stéréotypes)	Indispensable	
Possibilité de s'adapter à de nouveaux types de projets	Souhaitable	
Extensibilité de l'outil (scripting)	Utile	
Support d'autres notations (OMT Booch)	Inutile	
Outil de vérification de la cohérence	Indispensable	

S'il n'est pas prévu, à priori, d'étendre le métamodèle, il pourra toutefois s'avérer nécessaire d'en extraire certaines données. La documentation du schéma du repository est donc souhaitable, ainsi que des facilités d'accès à son contenu.

La sauvegarde et la restauration modulables des méta données pourraient s'avérer utiles à long terme mais ne revêtissent pas une importance capitale.

Par contre l'exportation et l'importation de modèles sont importantes pour l'échange d'information depuis et vers les fournisseurs externes (dans le cadre d'une collaboration avec un partenaire qui utiliserait un autre outil). Elle permet également de récupérer certaines informations lors d'un éventuel changement d'outil.

Il est indispensable de pouvoir travailler en groupe sur des projets, avec des notions d'utilisateurs regroupés sous des rôles différents, avec des droits d'accès spécifiques.

Le besoin de recourir à la notion de site est superflu vu les facilités importantes dont nous bénéficions en matière de télécommunications entre nos sites.

Critère	Importance	Commentaires
Méta données		
Documentation du schéma des méta données	souhaitable	
Facilité d'accès au schéma des méta données	souhaitable	
Extensibilité du métamodèle	inutile	
Sauvegarde / Restauration intelligente du métamodèle	utile	
Exportation XMI / PTL	souhaitable	permet l'interaction avec des fournisseurs externes ; pérennité des projets
Importation XMI / PTL	souhaitable	
Travail de groupe		
Accessibilité simultanée	indispensable	grands projets avec équipes importantes
Notion de sites et de réplication inter sites	inutile	
Notion d'utilisateurs	indispensable	
Notion de groupes d'utilisateurs	indispensable	grands projets avec équipes importantes
Possibilité de droits d'accès sur le projet	indispensable	grands projets avec équipes importantes
Notion de rôles	indispensable	Sans notion de rôles, la gestion des droit d'accès est un cauchemar

Nous n'utilisons pas de SGBD Orientés Objet. Nous avons donc besoin d'une aide importante pour parcourir le chemin entre la vue objet et la persistance traduite en terme de concepts relationnels. Notre architecture est basée sur le SGBD Oracle. A ce titre, nous souhaiterions que l'outil gère les procédures stockées, en Java parce que cela représente l'avenir de notre SGBD mais aussi en PL/SQL pour des raisons de compatibilité avec l'existant.

L'outil devrait permettre d'intégrer dans un projet la vision d'un base de données existante et surtout d'en générer une nouvelle de toutes pièces à partir des données du projet. Comme tous les projets ne seront pas UML du jour au lendemain, les bases de données vont continuer à évoluer en dehors de l'outil. Le Round Trip est donc également souhaité.

Critère	Importance	Commentaires
Gestion de la persistance		
Mapping objet vers physique	indispensable	Oracle 8.1
Support des procédures stockées	souhaitable	Java et PL/SQL

Critère	Importance	Commentaires
Création / Modification de la base		
forward	indispensable	via DDL, mieux en direct
reverse	souhaitable	via DDL, mieux en direct
Round Trip	souhaitable	via DDL, mieux en direct

L'Institution n'utilise pas, pour l'instant, d'environnements de type framework. La disponibilité d'un site permettant d'enrichir l'outil avec des patterns est souhaitable.

La Commission va utiliser comme standard l'architecture Java J2EE (un « Application Server » est en cours de sélection). Certains environnements spécifiques (TAXUD – gestion des douanes) font appel à Corba. L'environnement pour des petits projets (« workgroup ») est basé sur COM.

Nous n'avons pas d'environnement de développement standard à proprement parler. Les différentes communautés de développeurs (Directions Générales…) devraient pouvoir échanger aisément des composants (typiquement des « Entreprise Java Beans »).

Critère	Importance	Commentaires
Support et intégration dans des frameworks, travail par composants		
Disponibilité de modèles pour frameworks spécifiques	inutile	
Disponibilité de patterns	souhaitable	
Support d'architectures	indispensable	J2EE, COM et Corba souhaitables
Intégration avec des environnements de développements	utile	non encore décidé
Création de composants	indispensable	EJB

Dans un environnement de projets de grande taille comme ceux que l'Institution est amenée à gérer, un mécanisme de gestion de versions est souhaitable (éventuellement avec points de reprise).

Ces projets, de par leur ampleur et leur étalement, subissent des requêtes de modifications en cours de route. Pouvoir assurer le suivi de celles-ci, les mettre en relation avec les versions des artefacts du projet est souhaitable. Projeter les implications d'un changement serait un plus.

Critère	Importance	Commentaires
Gestion des versions		
Journalisation des modifications avec points de reprises	utile	
Comparaison de versions	souhaitable	
Gestion des requêtes de changement	souhaitable	
Analyse d'impact d'une modification	utile	

L'Institution possède du code de programmation dans une pléthore de langages. Néanmoins, comme exposé ci avant, notre environnement cible est Java. Il est donc indispensable de pouvoir générer et cycler en Java.

Toutefois, comme les environnements les plus récents étaient PowerBuilder et Visual Basic, avoir l'option d'acquérir des modules de reprise (reverse) dans ces langages serait un plus

Critère	Importance	Commentaires
Gestion du code		
Forward	Indispensable	Java
Forward	Utile	Visual Basic
Reverse	Indispensable	Java
Reverse	Utile	Visual Basic
Reverse	Utile	PowerBuilder
Round Trip	Indispensable	Java

Les projets ont jusqu'à présent été documentés en utilisant des documents Microsoft Office (principalement Word ...). L'intégration et la génération de tels documents est donc indispensable. Il serait souhaitable de pouvoir adapter le format des documents générés.

Un site Web (Intranet, Extranet...) nous semble un outil indispensable dans la politique de communication avec les partenaires des projets.

Le multilinguisme (actuellement 11 langues) serait un plus car certains projets critiques pourraient requérir une disponibilité de certains documents dans toutes les versions linguistiques. Un outil qui pourrait donc gérer cette dimension serait un plus, mais notre expérience montre que cela n'est pas commun.

Une intégration avec un gestionnaire de ressources du projet serait un plus (de préférence Microsoft Project 2000).

Critère	Importance	Commentaires
Gestion de la documentation		
Génération de documentation sur base du projet	indispensable	Word, RTF ou HTML et XML + DTD ou schéma
Publication de la documentation vers un site Web du projet	indispensable	
Adaptabilité du format de la documentation générée	Souhaitable	
Intégration et synchronisation de documents externes dans le projet	indispensable	HTML et XML + DTD ou schéma
Multilinguisme dans la documentation du projet	Utile	ES, DA, DE, EL, EN, FR, IT, NL, PT, FI, SV
Lien avec un gestionnaire de projet	Utile	MS Project 2000

Dans beaucoup de projets, on a tendance à perdre de vue les exigences de base telles qu'elles ont été exprimées par les représentants du client. Notre Institution n'échappe pas à cette règle. Un outil qui offre la possibilité de collecter ces exigences et de produire des tableaux de bord, permettant de se rendre compte dans quelle mesure les artefacts du projet prennent en compte à tout moment les exigences, est souhaitable.

Critère	Importance	Commentaires
Gestion des exigences		
Collecte des exigences	souhaitable	
Liens entre les exigences et les artefacts du projet	souhaitable	
Tableaux de bord	souhaitable	

Il est souhaitable d'avoir un outil aidant à administrer les tests (collecter les tests à réaliser, en gérer le statut en fonction des itérations, produire des statistiques de complétude...).

Eventuellement, il serait utile que le produit aide à la définition des tests à réaliser, mais cela n'est pas indispensable, surtout dans le cas de projets sous-traités.

Critère	Importance	Commentaires
Gestion des tests		
Collecte des tests à réaliser	souhaitable	
Aide à la définition des tests (scénarii / harnesses...)	utile	
Tableaux de bord d'avancement / succès des tests	souhaitable	

Les critères d'ergonomie sont assez classiques... ajoutons que, vu la taille des projets gérés, le produit doit être stable et relativement performant.

Critère	Importance	Commentaires
Ergonomie		
Propositions faites en fonction du contexte (pick lists...)	souhaitable	
Facilité de navigation (zooming, panning, nesting, ...)	souhaitable	
Adaptabilité de l'interface aux souhaits de l'utilisateur	indispensable	
Fonctions défaire et refaire	indispensable	
Stabilité	indispensable	
Performance	souhaitable	

L'Institution utilise des postes clients sous NT4 (et passera probablement d'ici un an à Windows 2000 / XP). C'est la seule plate-forme vraiment nécessaire pour tourner l'outil. Toutefois, comme des expériences pilotes de stations de développement sous Unix / Linux ont lieu, il serait utile que l'outil puisse tourner dans cet environnement. Si l'outil utilise une Base de Données comme repository, notre standard est Oracle 8.1 sous Unix et il serait utile de ne pas invalider ce choix d'architecture.

Critère	Importance	Commentaires
Plates-formes		
Tourne sur les plates-formes de l'entreprise	Indispensable Utile	NT4, Windows 2000 Linux
Fonctionnement dans une configuration hétérogène (clients)	inutile	
Fonctionnement avec un repository sur une autre plate-forme	utile	Oracle 8.1.7 sous Unix

Il est indispensable que l'outil dispose d'une bonne aide contextuelle intégrée dans le produit. Garder un support papier reste souhaitable pour des questions de commodité. Un tutorial sur papier ou format électronique serait apprécié, il serait toutefois complété par des formations classiques que la société doit être à même de fournir.

Critère	Importance	Commentaires
Aide à l'utilisation de l'outil		
Aide intégrée contextuelle (avec fonction de recherche)	indispensable	
Documentation papier	souhaitable	
Tutorial	Utile	
Formation	indispensable	

L'éditeur doit, idéalement, avoir une bonne présence sur le marché (Européen) tant en volume qu'en nombre d'années de présence, ceci afin de garantir une pérennité de l'investissement (financier et humain) consenti dans l'outil.

On tiendra évidemment compte du facteur coût dans le cadre de l'évaluation. On ne peut toutefois donner à ce critère de valeur à priori. Il servira à comparer deux produit qui satisferaient tous deux à nos exigences.

Critère	Importance	Commentaires
Editeur		
Quote-part du marché (%)		
Produit existe depuis (années)		
Coût		
Acquisition / utilisation des licences		
Coût des mises à jour		
Contrat de support		bug fixes, aide, nouvelles versions

Comme nous n'avons pas de méthodologie Orientée Objet en place et qu'il est difficile d'utiliser un tel outil sans un support méthodologique, il est indispensable qu'une méthodologie soit disponible.

A défaut d'avoir un outil adaptable, une méthodologie « propriétaire » de l'éditeur pourrait faire l'affaire. Il est souhaitable que cette méthodologie ait fait ses preuves afin de ne pas en essuyer les plâtres et devoir en changer sous peu.

Critère	Importance	Commentaires
Méthodologie		
Existence d'une méthodologie d'utilisation de l'outil (maison ou autre)	Indispensable	L'outil sans la méthodologie est peu utile
Adaptabilité	Souhaitable	Méthodologie importe peu mais jouissant d'une notoriété et éprouvée
Méthodologie maison disponible	Utile	

3.2.2.2.Cas de l'outil individuel

Le contexte d'utilisation de cet outil est fort différent de l'outil d'entreprise. Nous visons ici un outil qui sera utilisé par une personne ne travaillant pas au sein d'une équipe autour d'un projet.

L'objectif principal de la personne est une familiarisation avec U.M.L. et, sur un plan secondaire à l'utilisation d'un outil de type A.G.L. supportant U.M.L.

Cette personne pourrait être un étudiant mais aussi un professionnel qui souhaite se faire la main. Comme ces profils sont assez larges, et en fonction de leur environnement spécifique, il se peut qu'une caractéristique qui nous a parue inutile, se retrouve utile, voire souhaitable.

L'outil devrait supporter une version récente du langage, si possible la dernière. Idéalement, tous les diagrammes devraient être supportés mais cela n'est pas critique. Les diagrammes les plus utilisés (diagrammes de classes et use cases) constituent le strict minimum. L'extensibilité du langage est utile sans être critique car l'utilisateur novice devrait avant tout essayer d'exprimer les choses dans les limites du langage lui-même avant de penser à utiliser ses extensions. L'extensibilité à de nouveaux types de projets reste toutefois souhaitable vu l'évolution actuelle des environnements de développement.

Le support du scripting ne nous paraît pas utile. Celui d'autres notations n'est utile que dans le cas de personnes ayant une expérience avec d'autres représentations Orientées Objet, ce qui leur permettrait alors de passer d'une vue à l'autre.

La présence d'un outil de vérification de cohérence nous paraît un outil souhaitable pour nous assister dans la découverte du langage et de l'outil.

Critère	Importance	Commentaires
Langage		
Support de la version courante du langage	Souhaitable	
Support de tous les diagrammes	Souhaitable	
Extensibilité du langage (stéréotypes)	Utile	
Possibilité de s'adapter à de nouveaux types de projets	Souhaitable	
Extensibilité de l'outil (scripting)	Inutile	
Support d'autres notations (OMT Booch)	Inutile	Sauf si expérience dans une autre représentation OO
Outil de vérification de la cohérence	Souhaitable	

S'il peut éventuellement être utile d'accéder au schéma du métamodèle et, par conséquent, de disposer de la documentation de celui-ci et d'une méthode d'interrogation, son extensibilité et la sauvegarde / restauration ne nous paraissent pas utiles, vu le profil d'utilisation de l'outil.

Pouvoir exporter ses modèles dans un format permettant la reprise dans un autre outil nous semble utile (par exemple si notre utilisateur évolue plus tard vers un outil d'entreprise). Quant à l'importation d'autres formats, elle trouve un intérêt didactique car des exemples de modèles seraient sans doute plus aisés à trouver dans un format devenu standard d'échange.

Le besoin d'accès concurrent nous semble inutile, vu l'utilisation à titre individuel de l'outil. Il en va de même pour la notion de sites. La notion d'utilisateurs, de groupes d'utilisateurs, de rôles et de droits d'accès n'a pas de sens, au même titre.

Critère	Importance	Commentaires
Méta données		
Documentation du schéma des méta données	Utile	
Facilité d'accès au schéma des méta données	Utile	
Extensibilité du métamodèle	inutile	
Sauvegarde / Restauration intelligente du métamodèle	Inutile	
Exportation XMI / PTL	utile	
Importation XMI / PTL	utile	
Travail de groupe		
Accessibilité simultanée	Inutile	
Notion de sites et de	Inutile	

Critère	Importance	Commentaires
réplication inter sites		
Notion d'utilisateurs	Inutile	
Notion de groupes d'utilisateurs	Inutile	
Possibilité de droits d'accès sur le projet	Inutile	

La gestion des objets vers un stockage de l'information dans une base de données relationnelle nous semble important. On ne peut pas s'aligner sur un SGBD standard, ne sachant pas ce que l'utilisateur type peut utiliser. Il vaut mieux privilégier un travail vers et à partir de D.D.L. standard. Pour une initiation, l'utilisation de procédure stockées ne nous semble pas utile.

Il faut donc pouvoir générer du D.D.L. à partir des modèles et éventuellement en reprendre ou cycler.

Critère	Importance	Commentaires
Gestion de la persistance		
Mapping objet vers physique	Souhaitable	standard
Support des procédures stockées	Inutile	
Création / Modification de la base		
forward	Souhaitable	via DDL
reverse	Utile	via DDL
Round Trip	Utile	via DDL

L'utilisateur type que nous avons identifié plus haut ne devrait pas être demandeur du support pour des frameworks ou d'environnements de développement spécifiques. De même, la réutilisabilité des composants ne nous semble pas une caractéristique importante car elle est synonyme de production, pas de formation. Elle reste toutefois utile et nous optons pour EJB, vu l'orientation prise vers Java.

Pouvoir télécharger des patterns pour apprendre à les utiliser dans ses modèles nous semble, par contre, utile. De même, le support d'architectures (par exemple J2EE) devrait être une aide réelle à l'utilisateur.

Critère	Importance	Commentaires
Support et intégration dans des frameworks, travail par composants		
Disponibilité de modèles pour frameworks spécifiques	Inutile	

Critère	Importance	Commentaires
Disponibilité de patterns	Utile	
Support d'architectures	Utile	J2EE par exemple
Intégration avec des environnements de développements	Inutile	
Création de composants	Utile	EJB

Les projets gérés au sein de l'outil par nos utilisateurs types à des fins de formation devraient être d'une taille restreinte. L'utilisation à titre pédagogique devraient en limiter les requêtes de changements et l'évolution des fonctionnalités demandées en cours de projet.

Nous considérerons donc que la gestion des versions, sa journalisation, la gestion et projection des requêtes de changement comme des luxes inutiles dans le contexte qui nous occupe.

Critère	Importance	Commentaires
Gestion des versions		
Journalisation des modifications avec points de reprises	Inutile	
Comparaison de versions	Inutile	
Gestion des requêtes de changement	Inutile	
Analyse d'impact d'une modification	Inutile	

Pour la gestion du code, nous considérerons un seul langage cible : Java nous semble être suffisamment en phase avec les besoins de l'entreprise aujourd'hui pour être ce candidat unique.

Il est souhaitable de pouvoir générer du code afin de toucher au moins l'aspect implémentation. Le Round Trip serait utile mais pas critique. Le Reverse engineering en soi n'est pas utile.

Critère	Importance	Commentaires
Gestion du code		
Forward	Souhaitable	Java
Reverse	Inutile	Java
Round Trip	Utile	Java

Il faut pouvoir générer de la documentation dans un traitement de texte (RTF ou MS Word par exemple). Une génération en HTML pourrait faire l'affaire.

La personnalisation du format de sortie, la gestion d'un site intranet de projet, la synchronisation avec des documents externes à l'outil, le support du multilinguisme, de l'intégration avec un gestionnaire de projet n'est pas utile.

Critère	Importance	Commentaires
Gestion de la documentation		
Génération de documentation sur base du projet	indispensable	Word, RTF ou HTML
Publication de la documentation vers un site Web du projet	Inutile	
Adaptabilité du format de la documentation générée	Inutile	
Intégration et synchronisationde documents externes dans le projet	Inutile	
Multilinguisme dans la documentation du projet	Inutile	
Lien avec un gestionnaire de projets	Inutile	

Même si la prise en compte des exigences est importante, la gestion de la collecte de celles-ci ne nous paraît pas devoir être intégrée à l'outil car elle est en dehors de l'objectif poursuivi dans ce cas de figure.

Critère	Importance	Commentaires
Gestion des exigences		
Collecte des exigences	Inutile	
Liens entre les exigences et les artefacts du projet	Inutile	
Tableaux de bord	Inutile	

De même que pour la collecte des exigences, nous ne saurions insister assez sur l'importance des tests comme facteurs de succès des projets. Toutefois, cela n'intervient pas directement dans l'objectif poursuivi par ce cas de figure.

Critère	Importance	Commentaires
Gestion des tests		
Collecte des tests à réaliser	Inutile	

Critère	Importance	Commentaires
Aide à la définition des tests (scénarii / harnesses...)	Inutile	
Tableaux de bord d'avancement / succès des tests	Inutile	

Nous serons ici, d'une manière générale, moins exigeants quant à l'ergonomie du produit[1]. Le manque de performance et l'instabilité occasionnelle du produit ne devraient pas être trop critiques. L'adaptabilité de l'interface aux souhaits de l'utilisateur et les propositions intelligentes [contextuelles] revêtent moins d'importance que dans un outil utilisé à long terme.

Nous pensons qu'il est toutefois souhaitable de pouvoir revenir facilement en arrière sur un changement opéré (défaire) et pouvoir se déplacer aisément dans un diagramme qu'on ne pourrait plus visualiser complètement sur l'écran.

Critère	Importance	Commentaires
Ergonomie		
Propositions faites en fonction du contexte (pick lists...)	Utile	
Facilité de navigation (zooming, panning, nesting, ...)	Souhaitable	
Adaptabilité de l'interface aux souhaits de l'utilisateur	Utile	
Fonctions défaire et refaire	Souhaitable	
Stabilité	Utile	
Performance	Utile	

Nous ne pensons pas que l'utilisateur voudra utiliser l'outil dans des configurations hétérogènes. Vu le caractère mono utilisateur de l'outil, sa disponibilité sur la plate-forme de l'utilisateur est la seule exigence. Un outil écrit en Java permettrait de rencontrer aisément cette exigence (il serait dès lors portable sur toutes les plates-formes ayant une Java Virtual Machine disponible).

[1] En restant raisonnable : il ne faut pas que l'expérience de l'outil soit tellement négative qu'on en ressorte dégoûté.

Critère	Importance	Commentaires
Plates-formes		
Tourne sur les plates-formes de l'entreprise	indispensable	Celle de l'utilisateur
Fonctionnement dans une configuration hétérogène (clients)	inutile	
Fonctionnement avec un repository sur une autre plate-forme	Inutile	

Une aide intégrée (ou en ligne) dans l'outil est souhaitable. Si l'existence d'une documentation papier (ou d'un fichier imprimable) nous semble utile, c'est la disponibilité d'un tutorial (même en ligne) qui prend ici tout son sens, vu l'objectif de formation poursuivi. Pouvoir être formé à l'outil pourrait s'avérer utile si le démarrage de l'utilisateur s'avère difficile.

Critère	Importance	Commentaires
Aide à l'utilisation de l'outil		
Aide intégrée contextuelle (avec fonction de recherche)	Souhaitable	
Documentation papier	Utile	
Tutorial	Indispensable	
Formation	Utile	

Ce que nous visons avec cet outil est la formation de la personne à l'utilisation du langage UML et à l'utilisation d'<u>un</u> outil (pas <u>cet</u> outil).

La pérennité de cet outil, sa place au sein du marché a peu d'importance. Par contre, son coût d'acquisition, de mise à jour et de support doit être nul ou proche de zéro.

Critère	Importance	Commentaires
Editeur		
Quote-part du marché (%)	Inutile	
Produit existe depuis (années)	Inutile	
Coût		
Acquisition / utilisation des licences	~0	
Coût des mises à jour	~0	
Contrat de support	~0	

Il est souhaitable que l'outil soit le plus indépendant possible d'une méthodologie, afin de pouvoir l'utiliser quelle que soit celle choisie. La disponibilité d'une méthodologie « propriétaire » ne peut se concevoir que si l'outil ne permet pas de s'adapter à n'importe quelle méthodologie.

Critère	Importance	Commentaires
Méthodologie		
Existence d'une méthodologie d'utilisation de l'outil (maison ou autre)	Utile	L'outil sans la méthodologie est peu utile
Adaptabilité	Souhaitable	Méthodologie importe mais jouissant d'une notoriété et d'un historique
Méthodologie maison disponible	Utile	

4. MISE EN PRATIQUE DE L'ÉVALUATION SUR DEUX OUTILS

Dans le prolongement des deux grilles instanciées ci-avant, nous avons trouvé utile de démontrer le fonctionnement de chacune des grilles sur un outil correspondant au segment de marché visé (entreprise versus individuel).

Les grilles seront munies d'une nouvelle colonne montrant si le niveau désiré a été atteint par l'outil pour ce critère. Nous utiliserons le symbole 👍 pour montrer la satisfaction et le symbole 👎 pour dénoter l'échec. Lorsque la situation est plus nuancée, nous noterons ~. Si les fonctionnalités peuvent être atteintes par l'achat d'un module additionnel, nous l'indiquerons par le mot « option ».

Pour mettre en évidence les fonctionnalités importantes faisant défaut dans l'outil, nous utiliserons le symbole « stop » 🛑 si le critère était considéré comme indispensable, le symbole « danger » ⚠ si celui-ci était jugé souhaitable. Les critères jugés non utiles seront hachurés pour montrer leur caractère accessoire.

La première colonne du tableau fait référence à la section des annexes qui illustre le plus souvent la fonctionnalité concernée.

4.1. L'outil « entreprise » : Rational Rose 2001 Entreprise

L'offre commerciale de Rational est très complète. Nous avons choisi d'évaluer l'outil « Rose Entreprise » dans sa version la plus récente (2001A).

Lorsque des fonctionnalités évoquées dans la grille feront l'objet de produits séparés, disponibles chez l'éditeur ou un de ses partenaires mais à acheter séparément, nous mentionnerons leur existence, sans toutefois rentrer en détail dans leur évaluation.

	Critère	Importance	Vérifié	Commentaires
Langage				
1.1	Support de la version courante du langage	Indispensable	👍	La version implémentée est 1.3
1.2	Support de tous les diagrammes	Indispensable	👍	Officiellement pas de diagramme objet Simulé par diagramme de collaboration
1.3	Extensibilité du langage (stéréotypes)	Indispensable	👍	Stéréotypes prédéfinis dans fichier ini ; possibilité d'en rajouter
1.4	Possibilité de s'adapter à de nouveaux types de projets	Souhaitable	👍	Nombreux templates disponibles en standard

	Critère	Importance	Vérifié	Commentaires
1.5	Extensibilité de l'outil (scripting)	Utile	👍	L'outil dispose d'un langage de scripting « à la Visual Basic for Application ». Deux manuels (user et reference) sont dédiés à ce qu'ils nomment l' « extensability » L'outil est également un serveur COM et peut être étendu via des DLL ActiveX
1.6	Support d'autres notations (OMT Booch)	Inutile	👍	OMT / Booch
1.7	Outil de vérification de la cohérence	Indispensable	👍	La vérification de cohérence vise les références à des packages non existants.

L'outil répond à tous les critères concernant le support du langage.

	Critère	Importance	Vérifié	Commentaires
Méta données				
2.1	Documentation du schéma des méta données	Souhaitable	👍	REI (Extensability) documente les objets du métamodèle
2.2	Facilité d'accès au schéma des méta données	Souhaitable	👍	Scripting à la VBA
2.3	Extensibilité du métamodèle	inutile	👎	Repository constitué de fichiers propriétaires
2.4	Sauvegarde / Restauration intelligente du métamodèle	Utile	👎	Dépendances entre les fichiers propriétaires.
2.5	Exportation XMI / PTL	Souhaitable	👍	XMI possible grâce à un add-on gratuit PTL est natif
2.6	Importation XMI / PTL	Souhaitable	👍	XMI possible grâce à un add-on gratuit PTL est natif
Travail de groupe				
2.7	Accessibilité simultanée	Indispensable	👍	Le découpage est à organiser sur base des packages (manuellement) en « Controlled Units »
2.8	Notion de sites et de réplication inter sites	Inutile	option	Clearcase & Clearquest multi-sites

	Critère	Importance	Vérifié	Commentaires
2.9	Notion d'utilisateurs	Indispensable		Laissé à la discrétion du système d'exploitation sauf avec l'utilisation d'un outil optionnel : Clearcase
2.10	Notion de groupes d'utilisateurs	Indispensable		Ibid
2.11	Possibilité de droits d'accès sur le projet	Indispensable		Ibid
2.12	Notion de rôles	Indispensable		Ibid

Si la documentation du schéma et son accessibilité sont très bons, le support est basé sur un format propriétaire. L'importation et exportation vers XMI est supporté via un module à télécharger du site de l'éditeur. Seul bémol, on ne récupère pas les diagrammes.

L'accès concurrent est possible mais doit faire l'objet d'un travail préalable (définition des Controlled Units) pour éviter l'accès exclusif de l'entièreté du projet par une seule personne.

Rational a choisi l'approche de la légèreté en laissant au système d'exploitation le soin de gérer la problématique des accès. Il en résulte un contrôle d'accès qui n'est non seulement pas intégré à l'outil mais qui risque fort de manquer de granularité ou de concepts en relation avec les objets manipulés au sein de l'outil. C'est une grosse lacune pour de gros projets qu'il est possible de contourner en utilisant un outil optionnel de gestion de versions.

	Critère	Importance	Vérifié	Commentaires
Gestion de la persistance				
3.1	Mapping objet vers base de données	Indispensable		Le Data Modeler
3.2	Support des procédures stockées	Souhaitable	~	Uniquement en SQL
Création / Modification de la base				
3.3	Forward	Indispensable		Possible vers DDL avec enchaînement de l'exécution
3.4	Reverse	Souhaitable		Possible à partir du DDL ou en Direct
3.5	Round Trip	Souhaitable		Possible avec DDL ou en Direct

Un outil (Data Modeler) prend en charge la question de la persistance des classes. Il permet d'associer celles-ci avec les tables d'un RDBMS. L'outil intègre la notion de procédures stockées (composants) mais ne supporte que les procédures SQL. L'interaction avec le RDBMS (création, modification, reprise des tables) peut se faire par DDL ou en direct.

	Critère	Importance	Vérifié	Commentaires
Support et intégration dans des frameworks, travail par composants				
4.1	Disponibilité de modèles pour frameworks spécifiques	Inutile	option	Des templates sont disponibles pour certains environnements via RUP (option)
4.2	Disponibilité de patterns	Souhaitable	👍	Patterns « Gang of 4 » intégrés. D'autres disponibles sur le site rational.net
4.3	Support d'architectures	Indispensable	👍	Bon support de J2EE, de COM et Corba
4.4	Intégration avec des environnements de développement	Utile	👍	Non encore décidé En Java : Forté, Jbuilder, VisualCafé et Visual Age prévus.
4.5	Création de composants	Souhaitable	👍	A isoler dans des controlled units

L'outil est en phase avec les tendances du marché. Il est accompagné de « design patterns », supporte les environnements de développement et les frameworks courants.

Les composants qu'on souhaite rendre publics peuvent être isolés dans des packages ad-hoc et faire l'objet de controlled units spécifiques. Il ne dispose toutefois pas d'un outil « portail » permettant d'en publier la disponibilité.

	Critère	Importance	Vérifié	Commentaires
	Gestion des versions			
5.1	Journalisation des modifications avec points de reprises	Utile	Option	Fonctionnalité du produit optionnel « Clearcase »
5.2	Comparaison de versions	Souhaitable	Option	Ibid
5.3	Gestion des requêtes de changement	Souhaitable	Option	Fonctionnalité du produit optionnel « Clearquest »
5.4	Analyse d'impact d'une modification	Utile	~	Cette analyse est possible ex post

Les fonctionnalités de gestion des versions des artefacts d'un projet sont disponibles en recourant à un produit optionnel : Clearcase. Ce produit permet également de comparer des versions supposées différentes et de gérer l'introduction et le suivi des requêtes de changement. Il ne permet pas de se baser sur des métriques pour estimer le coût d'une modification du système.

	Critère	Importance	Vérifié	Commentaires
	Gestion du code			
6.1	Forward	Indispensable	👍	Java
6.2	Forward	Souhaitable	👍	Visual Basic
6.3	Reverse	Indispensable	👍	Java
6.4	Reverse	Utile	👍	Visual Basic
6.5	Reverse	Utile	option	Powerbuilder disponible via un add-in développé par la société Metex
6.6	Round Trip	Indispensable	👍	Java

La génération de code et le reverse engineering de code existant sont disponibles pour les langages Visual Basic et Java. Le round trip en Java l'est également. Il est possible de reprendre du code Powerbuilder en achetant un produit d'une société tierce.

	Critère	Importance	Vérifié	Commentaires
	Gestion de la documentation			
7.1	Génération de documentation sur base du projet	Indispensable	~	Pas de génération Word. Génération HTML possible mais sans tags XML.
7.2	Publication de la documentation vers un site Web du projet	Indispensable	👍	Sous Tools / Web Publisher
7.3	Adaptabilité du format de la documentation générée	Souhaitable	option	Nécessite SoDA
7.4	Intégration et synchronisation de documents externes dans le projet	Indispensable	👍	Raccourcis vers les documents ou URL créés dans l'outil.
7.5	Multilinguisme dans la documentation du projet	Utile	👎	N'est pas prévu
7.6	Lien avec un gestionnaire de projets	Utile	👎	Pas d'intégration mais des templates existent pour RUP

La génération standard est orientée vers la production d'un site web du projet (des documents HTML hiérarchisés au sein d'une structure). La production de documents d'un autre format (traitement de texte) ou avec un contenu personnalisé nécessite un outil optionnel. Il n'est, pour l'instant, pas possible de générer du XML en lieu et place du HTML, ce qui est dommage.

Il est aisé d'intégrer dans l'outil des références à des documents (ou URL) externes. Le multilinguisme des artefacts ne fait l'objet d'aucun traitement particulier. Il n'existe pas à proprement parler d'intégration avec un gestionnaire de projet mais des templates RUP (option) existent.

	Critère	Importance	Vérifié	Commentaires
	Gestion des exigences			
8.1	Collecte des exigences	Souhaitable	Option	Fait l'objet du logiciel optionnel « Requisite Pro »
8.2	Liens entre les exigences et les artefacts du projet	Souhaitable	Option	Ibid - Manuel
8.3	Tableaux de bord	Souhaitable	Option	Ibid - Traceability tree & matrix

Un outil spécifique est disponible en option pour la gestion des exigences.

	Critère	Importance	Vérifié	Commentaires
	Gestion des tests			
9.1	Collecte des tests à réaliser	Souhaitable	Option	Produit optionnel Testmanager
9.2	Aide à la définition des tests (scénarii / harnesses...)	Utile	Option	Produit optionnel QualityArchitect
9.3	Tableaux de bord d'avancement / succès des tests	Souhaitable	Option	Produit optionnel Testmanager

La gestion des tests et de leur statut, la génération de tableaux de bords est possible moyennant l'acquisition du produit optionnel « Testmanager ». Pour pouvoir gérer l'exécution de ces tests (domaines de réponses attendus, paramètres en entrées...) il faut acquérir un autre outil : QualityArchitect.

	Critère	Importance		Commentaires
Ergonomie				
10.1	Propositions faites en fonction du contexte (pick lists...)	Souhaitable	👍	Propositions contextuelles
10.2	Facilité de navigation (zooming, panning, nesting, ...)	Souhaitable	👍	Zooming et panning. L'imbrication de diagrammes est possible avec Diagramme d'Activités et d'Etat

	Critère	Importance		Commentaires
10.3	Adaptabilité de l'interface aux souhaits de l'utilisateur	Indispensable	👍	Panneau de configuration très complet
10.4	Fonctions défaire et refaire	Indispensable	~	Les fonctions ne sont pas toujours disponibles et ne fonctionnent pas sur plusieurs niveaux.
10.5	Stabilité	Indispensable	👍	Le produit est très stable
10.6	Performance	Souhaitable	👍	L'outil est long à charger mais performant pour le reste

Le produit dispose d'un bon confort d'utilisation général. On regrette toutefois que les fonctions « faire et défaire » ne soient pas toujours disponibles et qu'elle ne concernent que la dernière action.

	Critère	Importance	Vérifié	Commentaires
Plates-formes				
11.1	Tourne sur les plates-formes de l'entreprise	Indispensable Utile	👍	NT4 / Windows 2000 Unix / Linux
11.2	Fonctionnement dans une configuration hétérogène (clients)	Inutile	👍	Format PTL identique sous Unix et Windows
11.3	Fonctionnement avec un repository sur une autre plate-forme	Utile	👍	Le repository est constitué de fichiers partagés

Le produit est disponible sur les plates-formes courantes. L'interopérabilité est bonne, tant pour l'échange entre clients s'exécutant sur des plates-formes différentes que pour le partage (l'échange) de modèles.

	Critère	Importance	Vérifié	Commentaires
Aide à l'utilisation de l'outil				
12.1	Aide intégrée contextuelle (avec fonction de recherche)	Indispensable	👍	Help de type Windows
12.2	Documentation papier	Souhaitable	👍	11 ouvrages + fichiers PDF sur le CD
12.3	Tutorial	Utile	👍	Sur CD d'installation + ouvrage de Terry Quatrany
12.4	Formation	Indispensable	👍	Disponible chez l'éditeur

Tous les aspects d'aide et de documentation sont pleinement satisfaisants.

	Critère	Importance	Vérifié	Commentaires
Editeur				
13.1	Quote-part du marché (%)		👍	28,7 %
13.2	Produit existe depuis (années)		👍	Le produit est apparu dès 1992. Il s'est aligné sur UML en 1996
Coût				
13.3	Acquisition / utilisation des licences		👍	4220 € (node locked) 7380 € (floating)
13.4	Coût des mises à jour		👍	Inclus dans le contrat de support (voir ci-dessous)
13.5	Contrat de support		👍	845 € (node locked) 1480 € (floating)

Les coûts d'acquisition et de maintenance de l'outil sont forts élevés, mais cela est caractéristique de tous les outils CASE commerciaux.

	Critère	Importance	Vérifié	Commentaires
Méthodologie				
14.1	Existence d'une méthodologie d'utilisation de l'outil (maison ou autre)	Indispensable	👍	Existence de RUP (Rational Unified Process)
14.2	Adaptabilité	Souhaitable	👍	L'approche est souple : basé sur des templates
14.3	Méthodologie propre à l'éditeur disponible	Utile	👍	RUP

Une méthodologie propriétaire existe. Elle est bien intégrée à l'outil, tout en ne s'imposant pas.

En conclusion, nous pouvons dire que Rational Rose Entreprise rencontre la plupart des souhaits d'une entreprise, soit intrinsèquement, soit par l'adjonction de produits optionnels qui en complètent les fonctionnalités.

Nous regrettons seulement que le travail en groupe ne soit pas plus facile à mettre en œuvre et qu'il nécessite un produit additionnel pour pouvoir

réellement gérer des utilisateurs, des groupes d'utilisateurs, des rôles et des droits d'accès.

4.2. L'outil individuel : Argo UML

Cet outil est disponible à l'adresse http://argouml.tigris.org

	Critère	Importance	Vérifié	Commentaires
Langage				
1.1	Support de la version courante du langage	Souhaitable	👍	1.3 L'outil utilise une librairie pour implémenter le métamodèle
1.2	Support de tous les diagrammes	Souhaitable	👎 ⚠	Pas de sequence diagram L'object diagram doit être simulé par un diagramme de collaboration
1.3	Extensibilité du langage (stéréotypes)	Utile	👍	16 stéréotypes supportés
1.4	Possibilité de s'adapter à de nouveaux types de projets	Souhaitable	👎 ⚠	Peu adapté
1.5	Extensibilité de l'outil (scripting)	Inutile	👎	Pas de scripting : en Java (sources)
1.6	Support d'autres notations (OMT Booch)	Inutile	👎	Sauf si expérience dans une autre représentation OO
1.7	Outil de vérification de la cohérence	Souhaitable	👍	Les « critiques » offrent une certaine aide à la cohérence

L'outil ne rencontre pas deux critères souhaitables : deux diagrammes ne sont pas présents (mais le diagramme objet peut être simulé et le diagramme de séquence offre une vue différente du diagramme de collaboration qui est disponible).

L'adaptabilité de l'outil à gérer de nouveaux environnements (sites webs, XML...) n'est pas bonne. L'outil est orienté Java pur.

	Critère	Importance	Vérifié	Commentaires
Méta données				
2.1	Documentation du schéma des méta données	Utile	👍	En format source (DTD et DDL)
2.2	Facilité d'accès au schéma des méta données	Utile	~	C'est possible... en Java et JDBC si en mode SGBD
2.3	Extensibilité du métamodèle	inutile	👍	En mode SGBD, ne devrait pas poser de problème
2.4	Sauvegarde / Restauration intelligente du métamodèle	Inutile	👎	Rien n'est prévu
2.5	Exportation XMI / PTL	utile	~	XMI / PGML est la sauvegarde standard. PTL via produit tiers
2.6	Importation XMI / PTL	utile	~	Ibid
Travail de groupe				
2.7	Accessibilité simultanée	Inutile	👎	Non prévu, même en mode SGBD
2.8	Notion de sites et de réplication inter sites	Inutile	👎	Ibid
2.9	Notion d'utilisateurs	Inutile	👎	Ibid
2.10	Notion de groupes d'utilisateurs	Inutile	👎	Ibid
2.11	Possibilité de droits d'accès sur le projet	Inutile	👎	Ibid
2.12	Notion de rôles	Inutile	👎	Ibid

On peut dire que les critères concernant les méta données jugés utiles sont, d'une manière générale, rencontrés par l'outil. Il est clairement pour l'instant un outil à caractère individuel.

	Critère	Importance	Vérifié	Commentaires
Gestion de la persistance				
3.1	Mapping objet vers physique	Souhaitable	👎 ⚠	Rien n'est prévu
3.2	Support des procédures stockées	Inutile	👎	
Création / Modification de la base				
3.3	Forward	Souhaitable	👎 ⚠	Aucune aide pour la création de la base

	Critère	Importance	Vérifié	Commentaires
3.4	Reverse	Utile	👎	Aucune aide pour reprise base existante
3.5	Round Trip	Utile	👎	Aucune aide pour interaction avec le design d'une base

L'outil n'apporte malheureusement aucune aide à la gestion de la persistance des objets définis au sein des modèles.

	Critère	Importance	Vérifié	Commentaires
Support et intégration dans des frameworks, travail par composants				
4.1	Disponibilité de modèles pour frameworks spécifiques	Inutile	👎	Absent
4.2	Disponibilité de patterns	Utile	👎	Pas de modèles disponibles
4.3	Support d'architectures	Utile	👎	Pas présent dans l'outil
4.4	Intégration avec des environnements de développements	Inutile	👎	
4.5	Création de composants	Utile	👎	Pas disponible

Malheureusement, une infrastructure (communauté, groupes de discussions...) qui permettrait d'échanger des fragments de modèles, des « templates » fait défaut... Par ailleurs, l'outil ne favorise pas, il nous semble, cette approche.

	Critère	Importance	Vérifié	Commentaires
	Gestion des versions			
5.1	Journalisation des modifications avec points de reprises	Inutile	👎	Absent
5.2	Comparaison de versions	Inutile	👎	Absent
5.3	Gestion des requêtes de changement	Inutile	👎	Peut être simulé avec To Do
5.4	Analyse d'impact d'une modification	Inutile	👎	Absent

Aucun système de gestion de versions n'est intégré à l'outil.

	Critère	Importance	Vérifié	Commentaires
	Gestion du code			
6.1	Forward	Souhaitable		Java
6.2	Reverse	Inutile		Java
6.3	Round Trip	Utile		Java

La génération des classes Java fonctionne de manière satisfaisante. L'outil ne dispose pas dans la version 0.8 (cela semble prévu dans 0.9) d'une fonction de Reverse Engineering. Le Round Trip n'est pas disponible dans l'outil.

	Critère	Importance	Vérifié	Commentaires
	Gestion de la documentation			
7.1	Génération de documentation sur base du projet	indispensable		Word, RTF ou HTML
7.2	Publication de la documentation vers un site Web du projet	Inutile		Absent
7.3	Adaptabilité du format de la documentation générée	Inutile		Absent puisque aucune documentation n'est générable
7.4	Intégration et synchronisation de documents externes dans le projet	Inutile		Pas possible
7.5	Multilinguisme dans la documentation du projet	Inutile		Absent
7.6	Lien avec un gestionnaire de projets	Inutile		Absent

La possibilité de générer une documentation du projet est, malheureusement, absente du projet et cela nous semble une grosse lacune car elle force l'utilisateur à maintenir lui-même une telle documentation en dehors de l'outil (pour interagir avec les autres acteurs du projet).

	Critère	Importance	Vérifié	Commentaires
	Gestion des exigences			
8.1	Collecte des exigences	Inutile		Pas prévu mais peut être approché avec des to do listes
8.2	Liens entre les exigences et les artefacts du projet	Inutile		Absent
8.3	Tableaux de bord	Inutile		Absent

L'outil ne gère pas de collecte et traçage des exigences mais le mécanisme des listes « to do » peut aider pour de petits projets.

	Critère	Importance	Vérifié	Commentaires
Gestion des tests				
9.1	Collecte des tests à réaliser	Inutile	~	Pas prévu ; peut être simulé avec des « to do » listes
9.2	Aide à la définition des tests (scénarii / harnesses...)	Inutile	👎	Rien de prévu
9.3	Tableaux de bord d'avancement / succès des tests	Inutile	👎	Pas de gestion des tests

Non prévu mais peut être simulé pour de petits projets avec les listes « to do ».

	Critère	Importance		Commentaires
Ergonomie				
10.1	Propositions faites en fonction du contexte (pick lists...)	Utile	~	Une certaine gestion contextuelle est présente mais faible
10.2	Facilité de navigation (zooming, panning, nesting, ...)	Souhaitable	👎 ⚠	Pas de zoom, navigation fatigante sur de grands diagrammes
10.3	Adaptabilité de l'interface aux souhaits de l'utilisateur	Utile	👎	Pas de sauvegarde des paramètres de travail
10.4	Fonctions défaire et refaire	Souhaitable	👎 ⚠	Présent dans le menu mais grisé
10.5	Stabilité	Utile	👍	Bonne stabilité
10.6	Performance	Utile	👍	Bonne performance au sein de l'outil

Le confort ergonomique de l'outil est raisonnable mais souffre de l'absence de fonction « undo » et de l'impossibilité de zoomer sur des diagrammes importants en taille.

	Critère	Importance	Vérifié	Commentaires
Plates-formes				
11.1	Tourne sur les plates-formes de l'entreprise	indispensable	👍	Toute machine ayant un Java Runtime Environment
11.2	Fonctionnement dans une configuration hétérogène (clients)	inutile	👍	Echange de fichiers entre plates-formes sans surprises
11.3	Fonctionnement avec un repository sur une autre plate-forme	Inutile	👍	Possible en mode SGBD (JDBC)

Le fait que l'outil soit écrit en Java lui confère une disponibilité sur quasi toutes les plates-formes. De plus l'architecture Java lui confère une excellente interopérabilité.

	Critère	Importance	Vérifié	Commentaires
Aide à l'utilisation de l'outil				
12.1	Aide intégrée contextuelle (avec fonction de recherche)	Souhaitable	👎 ⚠	Pas d'aide intégrée
12.2	Documentation papier	Utile	👎	Pas disponible, même en format électronique
12.3	Tutorial	Indispensable	👍	Très sommaire mais meilleur chez Gentleware
12.4	Formation	Utile	👍	Possible chez Gentleware

Malheureusement, la documentation n'est vraiment pas au rendez-vous. Sans doute la start-up Gentleware était consciente de cette lacune et heureusement aide à relever le niveau, même pour l'utilisation d'Argo/UML.

	Critère	Importance	Vérifié	Commentaires
Editeur				
13.1	Quote-part du marché (%)	Inutile	~	Pas de chiffres disponibles (88.000 téléchargements)
13.2	Produit existe depuis (années)	Inutile	👍	Depuis 1998
Coût				
13.3	Acquisition / utilisation des licences	~0	👍	Gratuit
13.4	Coût des mises à jour	~0	👍	Gratuit

	Critère	Importance	Vérifié	Commentaires
13.5	Contrat de support	~0	👍	Gratuit via newsgroups mais peu rapide. Payant chez Gentleware

Le coût d'utilisation de l'outil rencontre nos attentes (gratuité).

	Critère	Importance	Vérifié	Commentaires
Méthodologie				
14.1	Existence d'une méthodologie d'utilisation de l'outil (propre à l'éditeur ou autre)	Utile	👍	Petite méthodologie intégrée dans l'outil
14.2	Adaptabilité	Souhaitable	~	Possible mais pas aisé
14.3	Méthodologie propre à l'éditeur disponible	Utile	👍	Embryon de méthodologie

Un embryon de méthodologie est inclus dans l'outil. On peut adapter celle-ci ; on n'est pas forcé de l'utiliser (on peut la désactiver).

En conclusion, cet outil possède tout ce qu'il est raisonnable d'attendre d'un outil gratuit (et même plus).

Malheureusement, il manque de documentation permettant à un débutant d'aborder l'outil et le langage. La communauté active autour de l'outil n'a pas développé un réseau d'échange de modèles, patterns, ... comme cela existe pour d'autres produits « concurrents ». Cela ne facilite pas le démarrage.

L'outil ne permet pas la génération d'une documentation ayant pour objectif la présentation du projet au monde extérieur. C'est un handicap sérieux pour l'utiliser en production.

Il ne fournit pas non plus d'aide à la mise en place d'une Base de Données pour assurer la persistance des objets manipulés par le produit qu'on développe.

Par contre, on ne peut que saluer la portabilité de l'outil, son respect des standards, ses conseils intégrés, sa performance et sa stabilité. L'outil Poséidon, basé sur Argo, gratuit dans sa « comunity edition » en cours de développement me semble mériter une attention.

5. CONCLUSION

Nous avons tout d'abord montré les spécificités d'une approche « Orienté Objet ». Ensuite, nous avons souligné que les méthodes fonctionnelles n'étaient pas à même de déboucher sur un bon modèle Objet.

UML est, nous l'avons expliqué, le langage qui a réussi, grâce à ses qualités intrinsèques, à mettre d'accord les principaux acteurs du monde « Orienté Objet ».

Grâce à UML et à l'approche « Orienté Objet », il est désormais plus aisé de mettre en œuvre un cycle de vie itératif qui devrait permettre d'être plus en phase avec les souhaits des « utilisateurs ».

Pour nous guider tout au long des projets, nous avons exprimé le besoin d'une méthodologie mais aussi d'un outil pour gérer tous les artefacts et leurs différentes versions.

Notre travail s'est alors recentré sur la définition d'une hiérarchie de critères reprenant les fonctionnalités susceptibles d'être fournies par un outil CASE.

Les catégories abordées concernaient le langage UML, les méta données, la persistance des données du système en cours de modélisation, l'interaction avec des environnements de développements, le cycle de vie du projet, les plates-formes d'exécution, l'aide à l'utilisation, le support méthodologique, le coût et la notoriété de l'éditeur.

Ces catégories ont alors été regroupées en une grille à instancier en fonction du contexte d'utilisation de l'outil à évaluer.

Nous avons également illustré l'instanciation de la grille dans deux contextes d'utilisation différents.

Finalement, nous avons utilisé chacune de ces grilles instanciées pour évaluer un outil pressenti comme adapté au profile d'utilisation correspondant.

Trois aspects évoqués dans notre travail mériteraient, à notre avis, une investigation plus poussée :

- La génération de code – notre travail n'a pas considéré la gestion de code comme la finalité des outils mais plutôt comme un moyen ; un travail plus centré sur ce que les outils pourraient capturer comme information dans les modèles pour générer du code (et vice-versa) permettrait une analyse plus fine concernant la qualité de l'interaction de l'outil avec le code.

- Les transformations de schéma de données – Nous nous sommes contentés de l'existence d'un module aidant à cette démarche alors qu'il serait possible d'étudier bien plus finement comment passer d'un diagramme de classes à un schéma relationnel (et l'inverse).

- Les méthodologies – nous sommes convaincus de l'importance d'associer une méthodologie « Orienté Objet » à l'utilisation de l'outil. Idéalement, le choix d'une méthodologie devrait se faire avant celui de l'outil. Le support de la méthodologie sélectionnée (ou l'existence d'un template, d'un add-in…) pourrait alors devenir un des critères décisifs dans les choix de l'outil.

6. BIBLIOGRAPHIE

6.1. Ouvrages de référence

Pierre-Alain Muller, *Modélisation Objet avec UML*, Editions Eyrolles, Paris 1997 (ISBN 2-212-08966-X)

Pierre-Alain Muller, *Instant UML*, Editions Wrox, Birmingham 1997 (ISBN 1-86-1000-87-1)

Nathalie Lopez, Jorge Migueis et Emmanuel Pichon, *Intégrer UML dans vos projets*, Editions Eyrolles, Paris 1998 (ISBN 2-212-08952-X)

Joseph Schmuller, *Sams teach yourself UML in 24 hours*, Editions Sams, Indianapolis 1999 (ISBN 0-672-31636-6)

Craig Larman, *Applying UML and Patterns*, Editions Prentice-Hall, Upper Saddle River 1997 (ISBN 0-137-48880-7)

Ian Sommerville, *Software Engineering*, Editions Addison Wesley, Harlow 1995 (ISBN 0-201-42765-6)

Martin Fowler, Kendall Scott, *UML distilled*, Editions Addison Wesley, Harlow 1997 (ISBN 0-201-32563-2)

Collectif , *XML Metadata Interchange – XMI specification version 1.1*, OMG, Needam 2000

6.2. Sites Internet

6.2.1. UML

http://uml.free.fr

http://www.omg.org/technology/uml/

6.2.2. Spécifications XMI

http://www.omg.org/cgi-bin/doc?formal/2000-11-02

6.2.3. Outils

Rational Rose http://www.rational.com/products/rose/index.jsp

Argo UML http://argouml.tigris.org/

Annexe 1
Illustrations de l'analyse de Rational Rose Entreprise

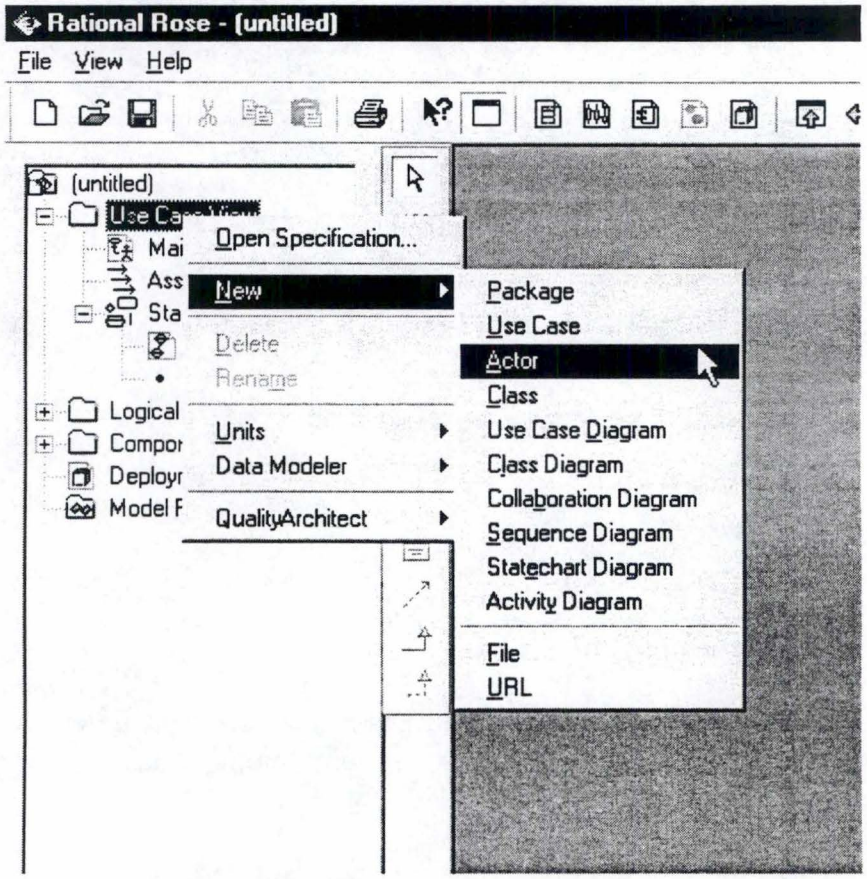
1. LANGAGE

1.1. Support de la version courante du langage

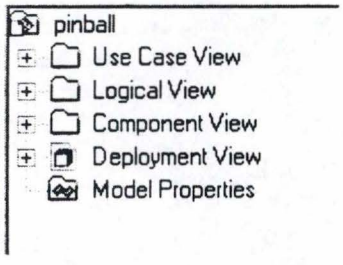
Rose implémente la version 1.3 d'UML. Il n'y aura probablement jamais d'implémentation 1.4 car ils travaillent sur la réécriture de leur moteur en vue du support de UML 2.0.

1.2. Support de tous les diagrammes

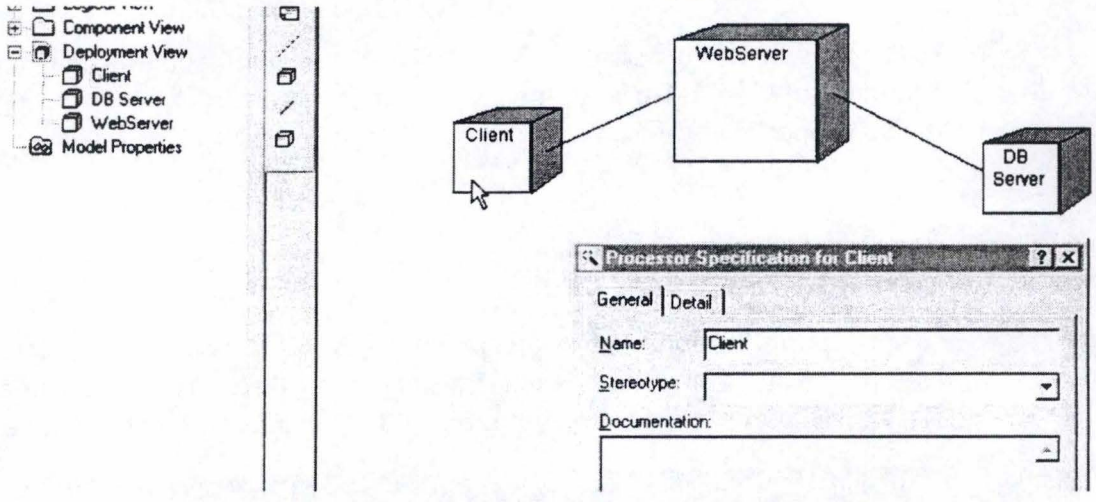
Rose supporte l'ensemble des diagrammes UML. On peut voir directement dans le menu contextuel « new » les diagrammes de Cas d'Utilisation, de Classes, de Séquence, de Collaboration, de Paquetage, d'État et d'Activité.



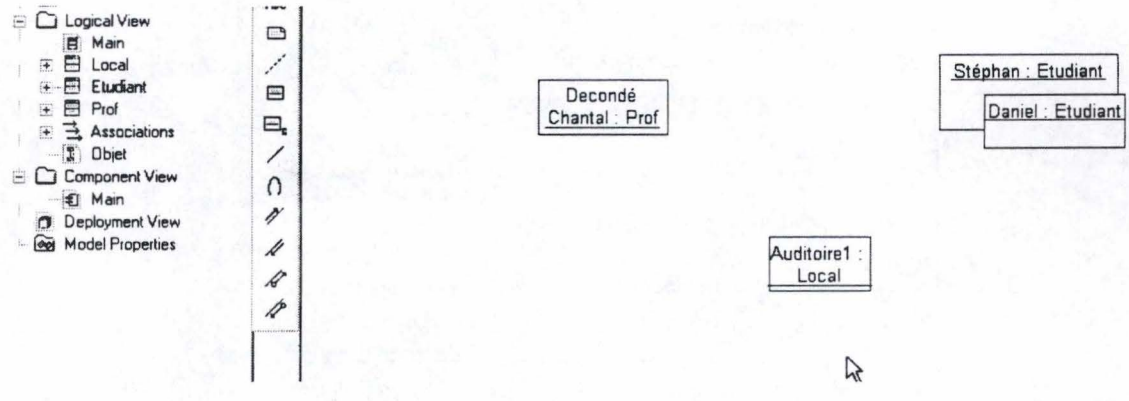
Le diagramme de déploiement fait l'objet d'une vue spécifique dans l'explorer de Rose (deployment view) :



C'est dans cette vue qu'il convient de créer le diagramme de déploiement :



Pour créer un diagramme objet, il faut user d'une astuce en créant un diagramme de collaboration :

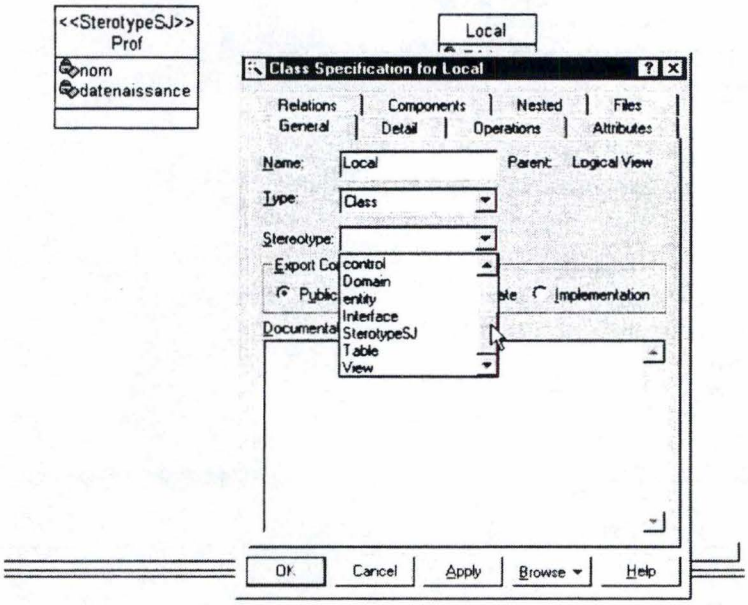


1.3. Extensibilité du langage

Les stéréotypes sont supportés par l'outil. On peut en créer de nouveaux. Ils sont alors disponibles dans le projet. Ils vont rejoindre les stéréotypes prédéfinis qu'on peut sélectionner dans la liste :

- Classes: Interface, Control, Boundary, Entity, Business Actor, Business Entity, View, Domain, Table, Database
- Associations : Extend, Include, Communicate, Subscribe, Realize
- Components : EXE, DLL, ActiveX, Application, Applet
- Use Case: business use case, use-case realization
- Use Case Package: Organization Unit, Subsystem, Layer
- Logical Package: Organization Unit, Subsystem, Layer, Domain Package

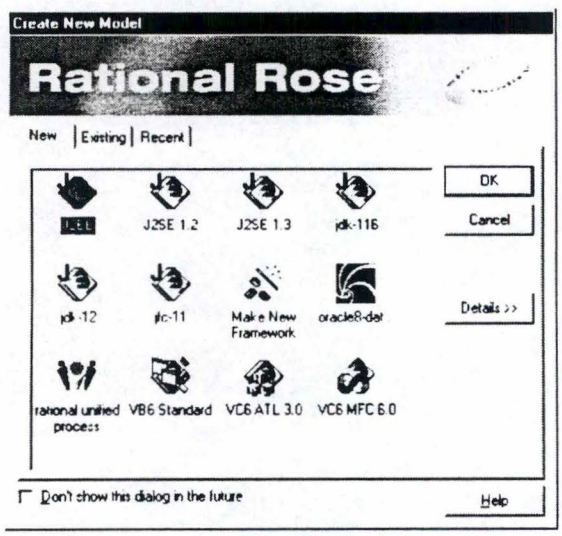
- Dependency Relation: refine, extend, include, derive

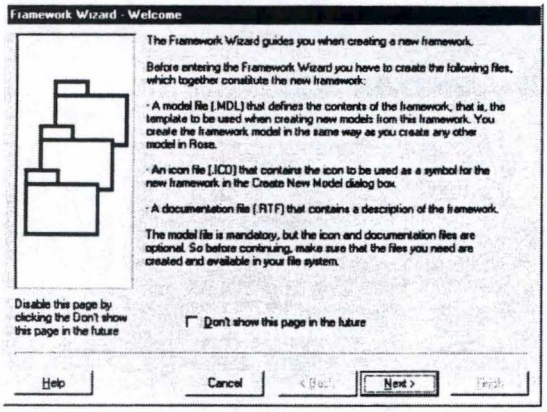


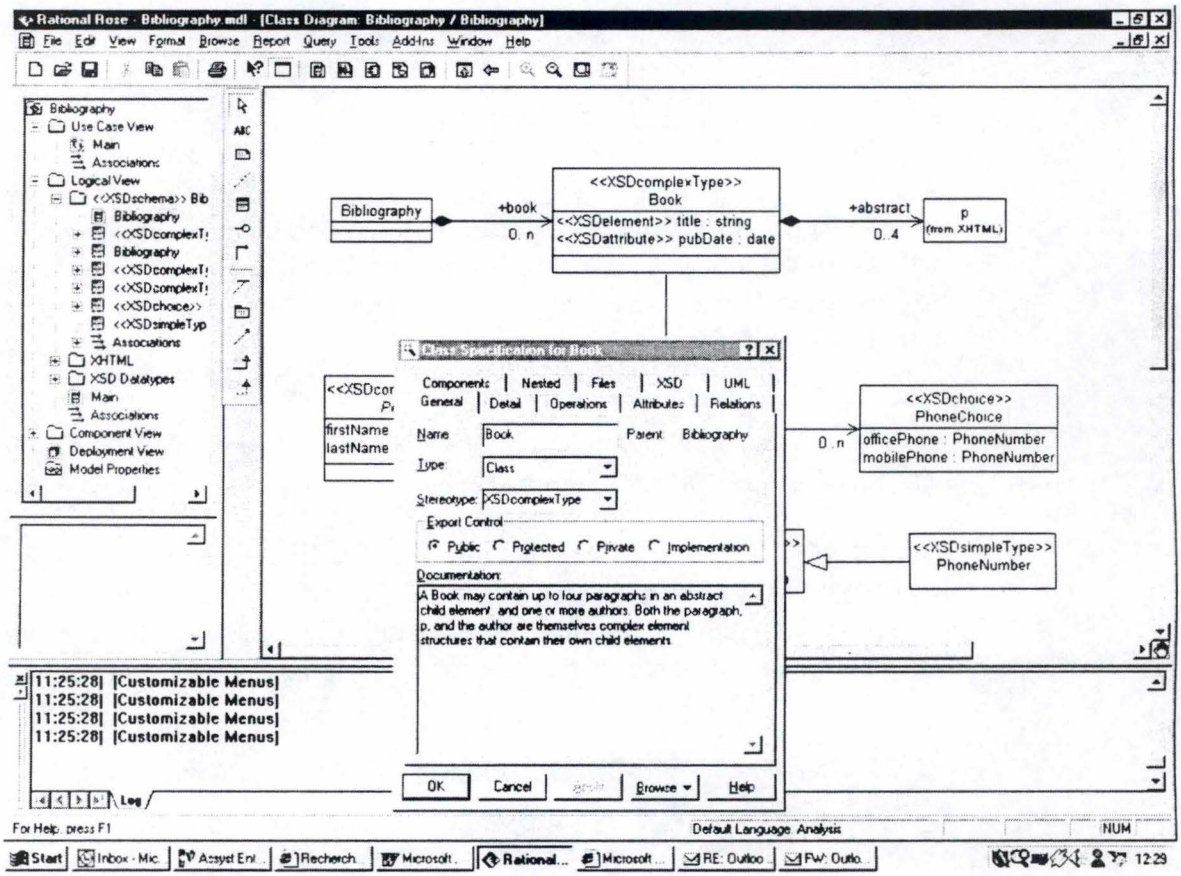
Le chapitre 12 du manuel utilisateur explique comment rendre de nouveaux stéréotypes disponibles pour tous les projets. Des icônes spécifiques peuvent être assignées aux stéréotypes. C'est déjà le cas pour une série des stéréotypes prédéfinis.

1.4. Possibilité d'adaptation à de nouveaux types de projets

La souplesse d'utilisation souhaitable de l'outil est démontrée. Lors du démarrage, l'outil propose de partir d'une série de templates standards « frameworks ». Il est également possible d'enrichir la liste proposée. L'extensibility interface (voir ci-dessous) permet un grand niveau de souplesse.

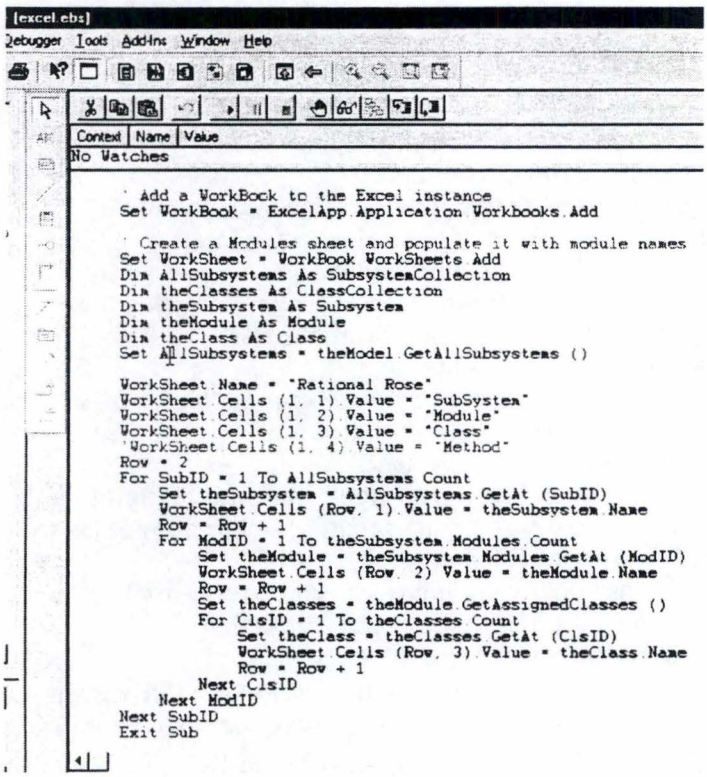



Framework Wizard - Welcome
The Framework Wizard guides you when creating a new framework.
Before entering the Framework Wizard you have to create the following files, which together constitute the new framework:
- A model file (.MDL) that defines the contents of the framework, that is, the template to be used when creating new models from this framework. You create the framework model in the same way as you create any other model in Rose.
- An icon file (.ICO) that contains the icon to be used as a symbol for the new framework in the Create New Model dialog box.
- A documentation file (.RTF) that contains a description of the framework.
The model file is mandatory, but the icon and documentation files are optional. So before continuing, make sure that the files you need are created and available in your file system.
Disable this page by clicking the Don't show this page in the future
Don't show this page in the future
Help
Cancel
< Back
Next >
Finish


Rational Rose - Bibliography.mdl - [Class Diagram: Bibliography / Bibliography]
File Edit View Format Browse Report Query Tools Add-Ins Window Help
Bibliography
Use Case View
Main
Associations
Logical View
Bibliography
XHTML
XSD Datatypes
Component View
Deployment View
Model Properties
Bibliography
+book
0..n
<<XSDcomplexType>>
Book
<<XSDelement>> title : string
<<XSDattribute>> pubDate : date
+abstract
0..4
p
(from XHTML)
Class Specification for Book
Components | Nested | Files | XSD | UML
General | Detail | Operations | Attributes | Relations
Name: Book
Parent: Bibliography
Type: Class
Stereotype: XSDcomplexType
Export Control
Public Protected Private Implementation
Documentation:
A Book may contain up to four paragraphs in an abstract child element and one or more authors. Both the paragraph, p, and the author are themselves complex element structures that contain their own child elements.
OK
Cancel
Browse
Help
firstName
lastName
0..n
<<XSDchoice>>
PhoneChoice
officePhone : PhoneNumber
mobilePhone : PhoneNumber
<<XSDsimpleType>>
PhoneNumber
11:25:28| [Customizable Menus]
11:25:28| [Customizable Menus]
11:25:28| [Customizable Menus]
11:25:28| [Customizable Menus]
Log
For Help, press F1
Default Language: Analysis
NUM
Start
Inbox - Mic...
Assyst Ent...
Recherch...
Microsoft...
Rational...
Microsoft...
RE: Outloo...
FW: Outlo...
12:29

1.5. Extensibilité par scripting

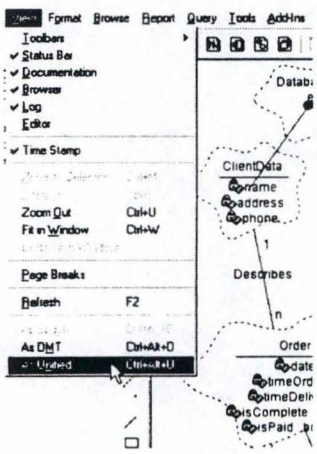
Cette fonctionnalité utile est très bien développée dans deux manuels parlant de la « Rose Extensibility Interface ». Des scripts d'exemples sont livrés dans le répertoire « scripts » :



Rose est également serveur COM (il est écrit en Visual C++). Pour certaines fonctions comme des menus contextuels (clique droit de souris), il faut passer par l'écriture de DLL et on ne peut se contenter d'un script.

1.6. Support d'autres notations

Jugé non utile ; l'outil permet toutefois d'alterner des vues OMT et Booch.



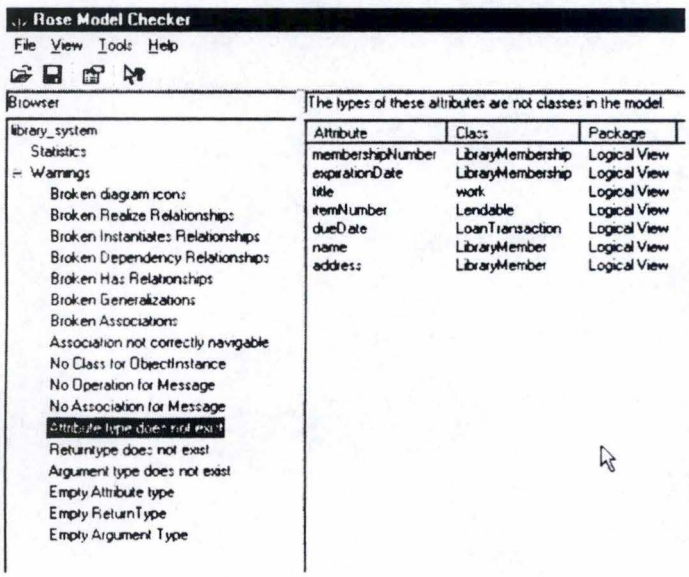
1.7. Outil de vérification de la cohérence

Un outil « Check Model » dans le menu Tools permet de vérifier s'il n'y a pas de références à des packages non existants :

```
13:42:46|      to Item with name ::sun::rmi::server
13:42:46|      by Dependency "<unnamed>".
13:42:46| Error: Unresolved reference from Package "activation"
13:42:46|      to Item with name ::sun::security::action
13:42:46|      by Dependency "<unnamed>".
13:42:46| Error: Unresolved reference from Package "activation"
13:42:46|      to Item with name ::sun::rmi::transport
13:42:46|      by Dependency "<unnamed>".
13:42:46| Error: Unresolved reference from Package "rmi"
13:42:46|      to Item with name ::sun::rmi::server
13:42:46|      by Dependency "<unnamed>".
13:42:46| Error: Unresolved reference from Component "BigInteger"
13:42:46|      to Item with name ::sun::security::action::LoadLibraryAction
13:42:46|      by Dependency "<unnamed>".
13:42:46| Error: Unresolved reference from Package "math"
13:42:46|      to Item with name ::sun::security::action
13:42:46|      by Dependency "<unnamed>".
13:42:46| Error: Unresolved reference from Component "DriverManager"
13:42:46|      to Item with name ::sun::security::action::GetPropertyAction
```

Un outil additionnel mais gratuit « Rose Model Checker » est disponible sur le site http://www.rationalrose.com/modelchecker/downloads.htm

Il vérifie une série d'incohérences (comme les types classes inexistants : http://www.rationalrose.com/modelchecker/warnings.htm)



2. MÉTA DONNÉES

2.1. Documentation du schéma des méta données.

Le modèle objet englobant le schéma est documenté dans le manuel « Rose Extensibility Reference ».

Des diagrammes objet viennent compléter la description textuelle.

2.2. Facilité d'accès

Le langage « script » utilisé est facile d'accès. Des exemples d'accès au schéma sont fournis dans le répertoire « scripts ».

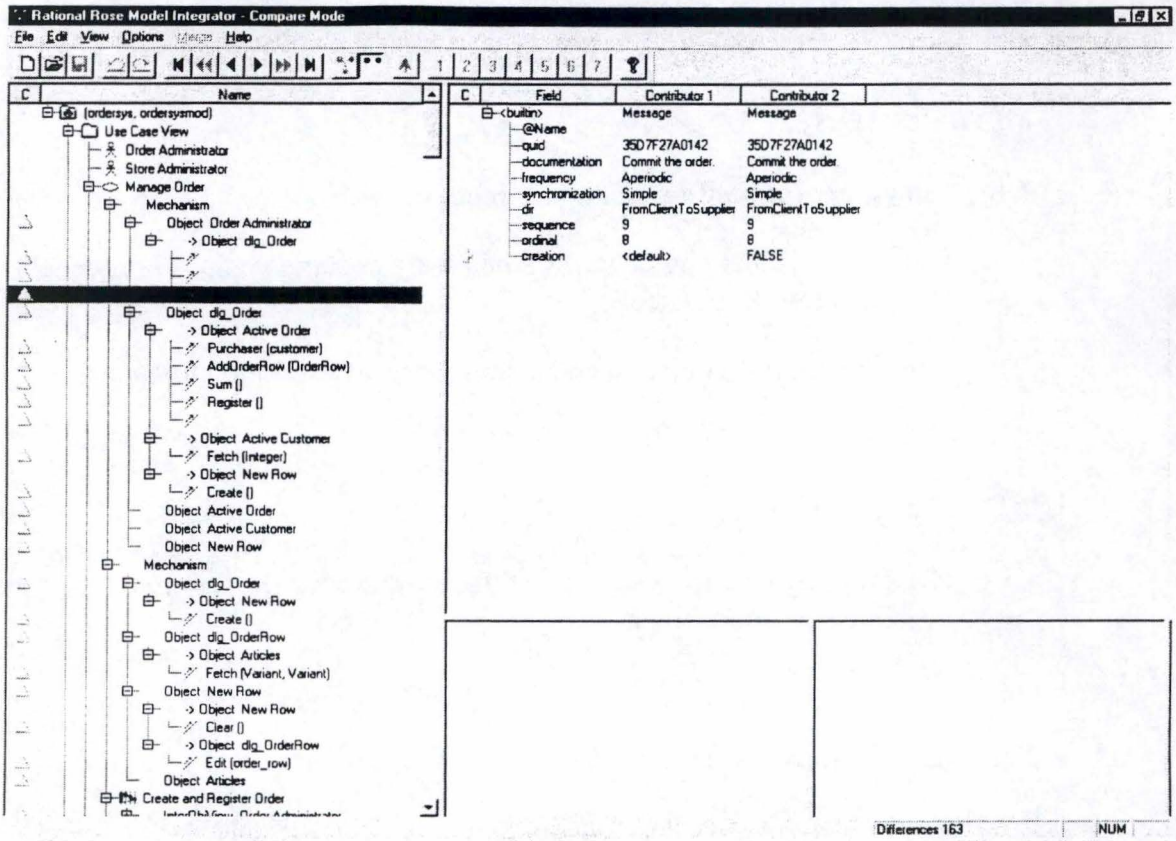
2.3. Extensibilité du métamodèle.

Jugée non utile, cette fonctionnalité semble difficile à rencontrer, vu le format propriétaire et l'utilisation de fichiers par l'outil. L'utilisation de l'extensability (REI) permet toutefois d'ajouter des tags spécifiques et de les stocker dans les fichier MDL.

2.4. Sauvegarde/ Restauration du métamodèle.

Les modèles sont stockés dans un fichier propriétaire. En configurant le projet pour un travail de groupe, on peut répartir les données sous plusieurs fichiers, en fonction des « Controlled Units ». Il existe toutefois une dépendance entre les fichiers d'un même projet.

Mieux vaut donc restaurer l'entièreté du projet et utiliser ensuite le model integrator pour déterminer les différences et fusionner les deux instances du projet en une seule. L'utilisation d'un outil de versioning peut également fournir une alternative.

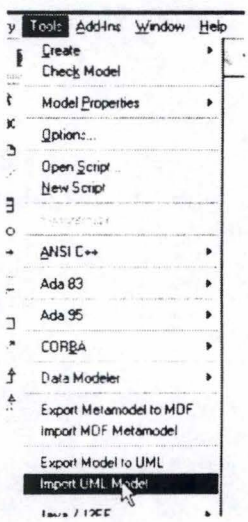


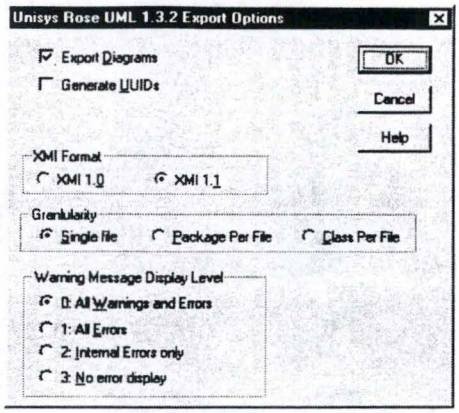
2.5. Exportation vers XMI / PTL

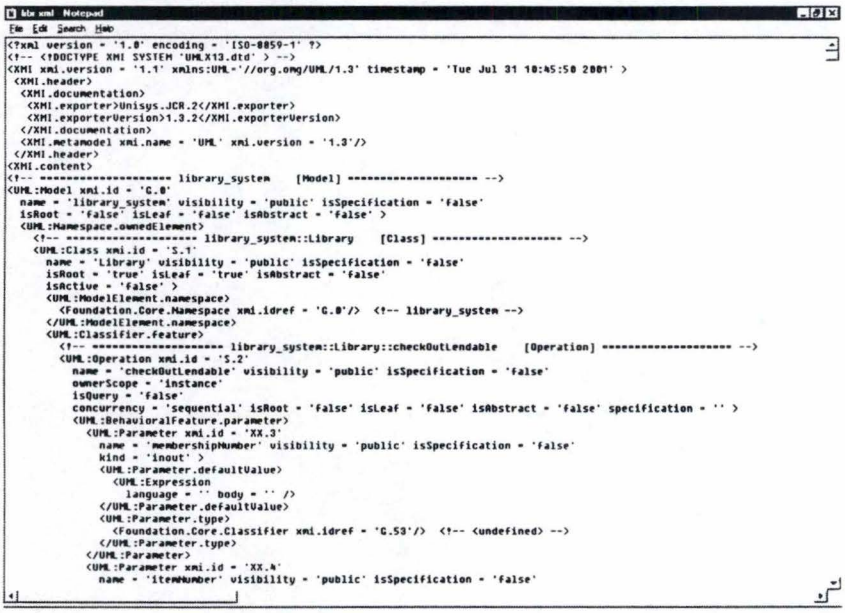
Un accord passé entre Rational et Unisys permet de disposer d'un add-in gérant le format XMI. Cet add-in peut être téléchargé à partir de :

http://www.rational.com/products/rose/forms/xmisupport/xmisupport.jsp

Une fois cet add-in installé, le menu « tools » est enrichi avec une fonction « Export to UML » et « Import UML model ».





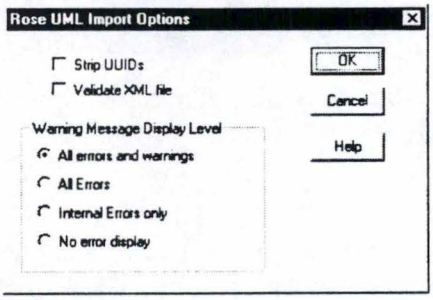


L'exportation prend en charge les méta données et non les diagrammes.

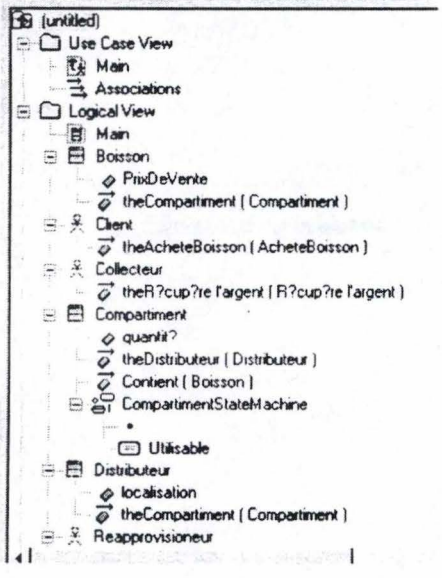
Le format PTL est natif.

2.6. Importation XMI / PTL

Comme expliqué ci-dessus, un add-in d'Unisys permet de remplir cette fonction.



Nous avons testé la fonction avec l'importation d'un modèle créé sous Argo/UML. On notera juste au passage un problème dans la gestion des jeux de caractères.



Comme expliqué ci-dessus, l'importation prend donc en charge les méta données et non les diagrammes.

Le format PTL est natif.

2.7. Travail de groupe

Le « Guide to Team Development » (disponible en format Acrobat sur le CD de documentation reprend la philosophie de Rose pour gérer des projets en équipe.

2.7.1. Accessibilité simultanée

Les tâches à mettre en œuvre pour permettre un accès simultané à un projet sont décrites dans le « Guide to Team Development »

Ce manuel décrit tant l'environnement humain (rôles dans l'équipe) que l'environnement technique à mettre en œuvre.

Le projet doit être découpé en packages (avec une interdépendance faible) destinés à être développés en parallèle. On assigne alors à ces packages le statut de « Controlled Units » Les packages concernés sont alors stockés

dans des fichiers spécifiques (et non un « .MDL » monolithique comme prévu par défaut)

L'accès simultané est réalisé par l'intermédiaire d'un disque partagé (reconnu sur le client comme un lecteur virtuel ou un « mounted filesystem »). Il est nécessaire d'établir des « Virtual Path Maps » pour permettre de rendre le client indépendant du chemin d'accès à la ressource (qui peut être différent de client en client).

2.7.2. Notions de sites et de réplication

Considéré comme non utile, cette fonctionnalité n'est possible qu'au travers d'un outil optionnel de gestion de versions (Clearcase Multi-site).

2.7.3. Notion d'utilisateurs

L'outil ne prévoit pas de notion d'utilisateurs. Cette fonctionnalité est laissée à la discrétion du système d'exploitation.

Pour obtenir une gestion d'utilisateurs intégrée au produit et indépendante de l'outil, il faut passer par un outil optionnel de contrôle de versions : Clearcase.

2.7.4. Notion de groupes d'utilisateurs

Se base sur le système d'exploitation, à moins d'utiliser l'outil optionnel de contrôle de version.

2.7.5. Possibilité de droits d'accès sur le projet

Aucune granularité n'est disponible au sein de l'outil. Les droits d'accès sont laissés à la discrétion du système d'exploitation. Tout au plus peut-on bloquer l'accès au controlled unit à partir de l'outil.

2.7.6. Gestion des rôles

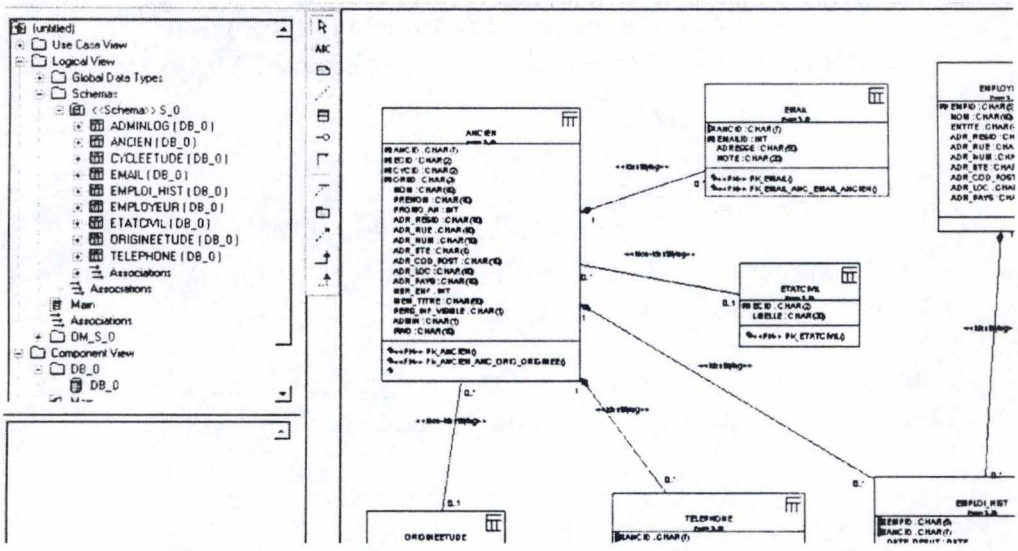
Aucune gestion de rôles au sein de l'outil.

3. GESTION DE LA PERSISTANCE

La gestion de la persistance est assurée par un module de l'outil appelé « Data Modeler »

3.1. Mapping objet vers physique

Rose dispose d'un outil appelé le « Data Modeler » Cet outil permet d'établir une correspondance entre les classes persistantes d'un diagramme de classes et des tables au sein d'un schéma incluant les caractéristiques spécifiques à une Base de Données.



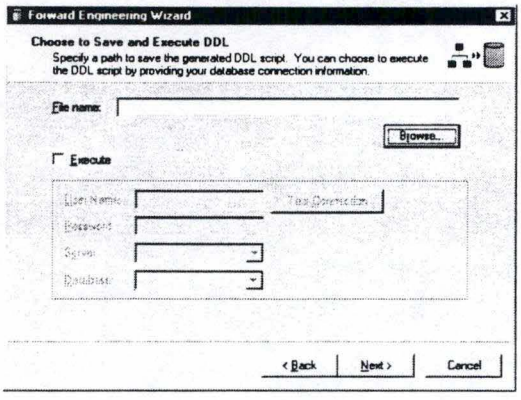
3.2. Support des procédures stockées

L'outil gère les procédures stockées mais celles-ci doivent impérativement être de type SQL (pas de Java ou de PL/SQL).

3.3. Création / modification de la base

3.3.1. Création/Modification via DDL

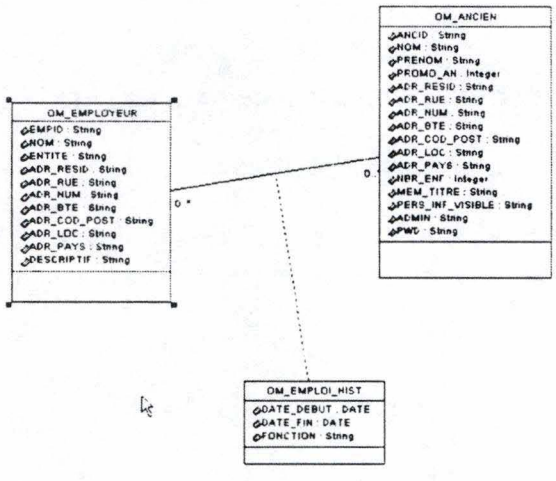
Il est possible de générer sur base de ce schéma un script DDL et, sur option, d'enchaîner son exécution.



3.3.2. Reprise d'un DDL existant

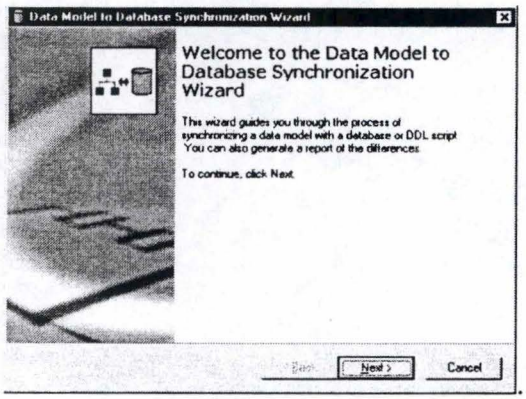
Il est possible (Tools / Data Modeler / Reverse Engineer) de recréer un schéma à partir d'un script DDL ou du catalogue DB2, Microsoft SQL server, Oracle ou Sybase.

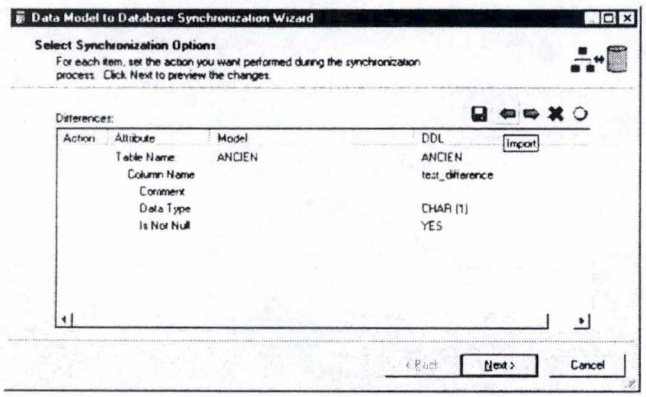
On peut ensuite transformer le schéma ainsi obtenu en diagramme de classes.



3.3.3. Round-trip avec DDL

L'outil de synchronisation permet de comparer le modèle avec le catalogue ou le DDL de la base et de rapporter les différences. L'utilisateur a alors la possibilité de décider ce qu'il fait avec les différences (ignorer, modifier la source externe, adapter le schéma…)





4. SUPPORT ET INTÉGRATION DES FRAMEWORKS, TRAVAIL PAR COMPOSANTS

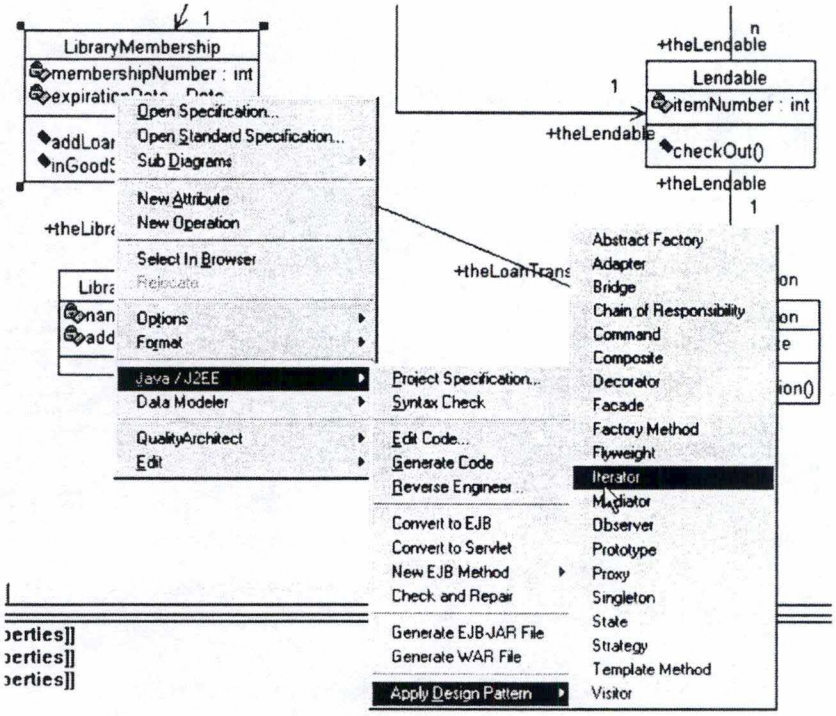
4.1. Disponibilité de modèles pour frameworks spécifiques

Dans le cadre de RUP (un produit optionnel), il existe des templates permettant de développer des solutions à déployer sur des environnements tels IBM Websphere ou BEA Weblogic.

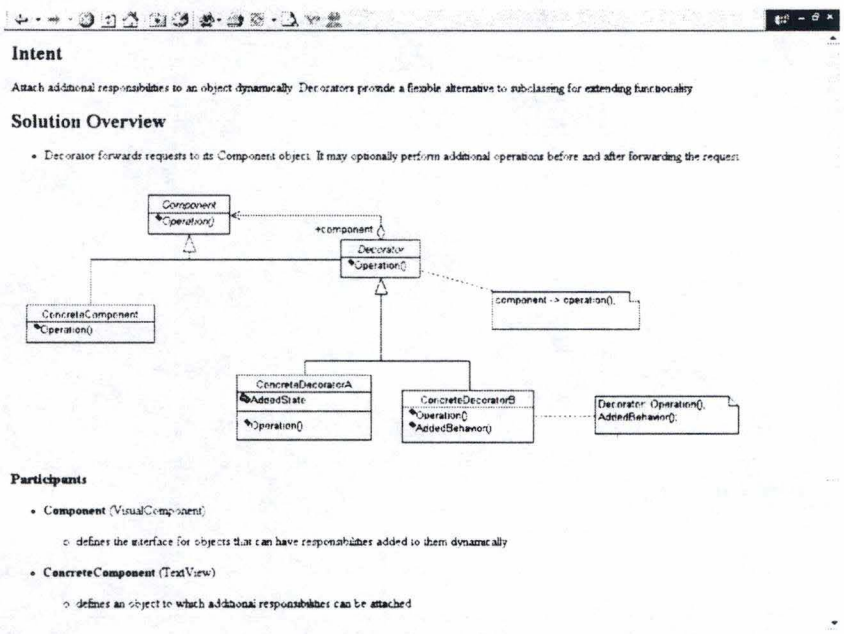
4.2. Disponibilité de patterns

Les patterns du « Gang of four »[2] sont livrés en standard avec l'outil et peuvent être appliqués aux classes Java.

[2] http://pigseye.kennesaw.edu/~dbraun/csis4650/A&D/GoF_Patterns/index.htm



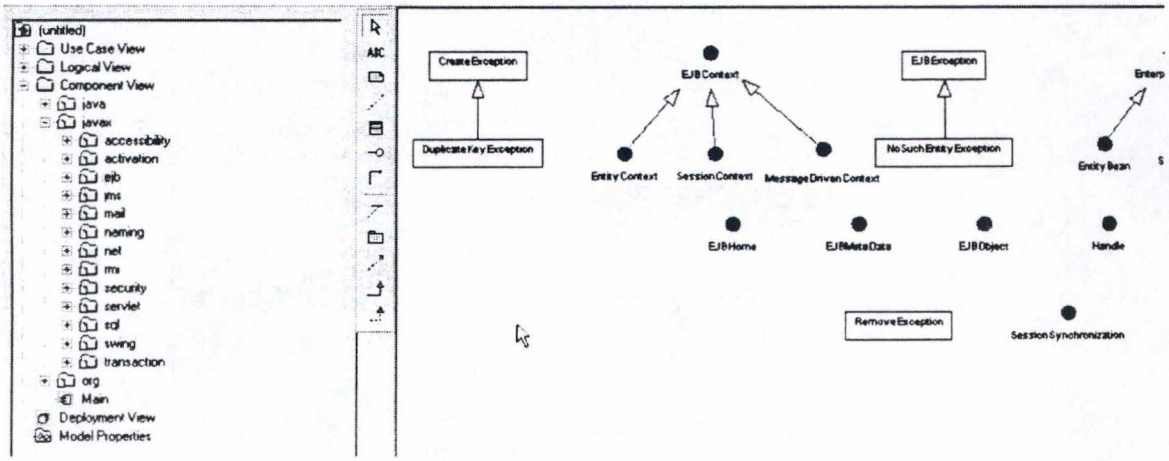
Les patterns sont décrits par des fichiers HTML présents dans le répertoire « Design Patterns » de l'outil.



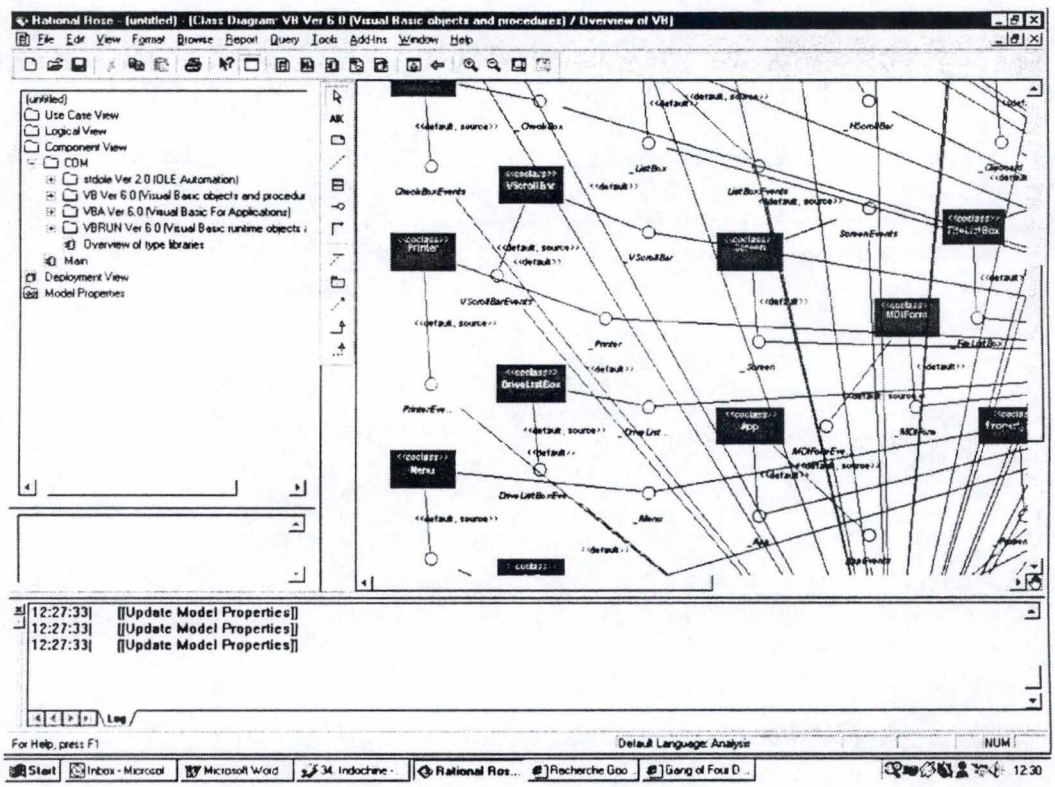
Le site http://www.rational.net/ fournit également une communauté qui met à disposition, entre autres choses, des patterns aux utilisateurs enregistrés (qui doivent disposer d'un contrat de support pour y avoir accès).

4.3. Support d'architecture

Comme nous l'avons évoqué plus haut, l'architecture J2EE est disponible comme template dans l'outil.



Pour les projets Visual Basic, il en va de même pour l'architecture COM.



En ce qui concerne Corba, tant le Reverse engineering que la génération des fichiers IDL est possible. Il n'existe pas de framework Corba (ce n'est pas un langage), mais des stéréotypes spécifiques (par exemple CORBAConstant, CORBANative, CORBATypedef, CORBAEnum, CORBAStruct, CORBAUnion, CORBAException).

4.4. Intégration avec environnements de développement

Le document « What's new in Rational Rose V2001A » nous indique, en ce qui concerne Java, l'intégration avec Forté, VisualCafé, Jbuilder et Visual Age. Nous n'avons pu investiguer celle-ci car nous ne possédions aucun des outils.

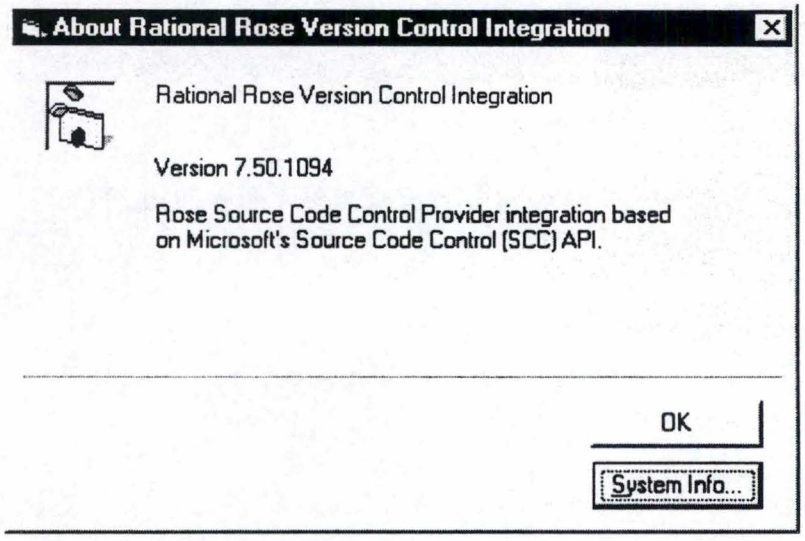
4.5. Création de composants

Les composants qui doivent être rendus « publics » sont isolés dans des packages spécifiques. Des controlled units doivent alors être créer pour permettre leur partage avec d'autres projets.

Aucun outil ne permet d'en publier la disponibilité.

5. GESTION DES VERSIONS

Tous les outils supportant l'API S.C.C. (Source Code Control) peuvent être utilisés, notamment Visual Source Safe, Rational Clearcase, Revision Control System (R.C.S.) et Source Code Control System (SCCS).



Rational Clearcase est toutefois un produit optionnel. Nous n'avons pu le tester et abordons donc ses fonctionnalités d'une manière sommaire.

http://www.rational.com/products/clearcase/prodinfo.jsp

5.1. Journalisation des modifications avec points de reprise

Le produit (optionnel) permet de prendre des « snapshots » des artefacts du projet à un moment donné.

5.2. Comparaison de versions

L'outil (optionnel) intègre la possibilité de comparer entre elles des versions des artefacts en ce compris les documents HTML, XML, Word.

5.3. Gestion des requêtes de changement

Un autre outil optionnel Clearquest (http://www.rational.com/products/clearquest/index.jsp) prend en charge la problématique de l'acquisition des requêtes de changement et de leur suivi.

5.4. Analyse de l'impact d'une modification

Des métriques concernant les modèles sont exploitées par les produits optionnels Clearcase et Clearquest. Elle permettent de tracer des graphes ex post.

6. GESTION DU CODE

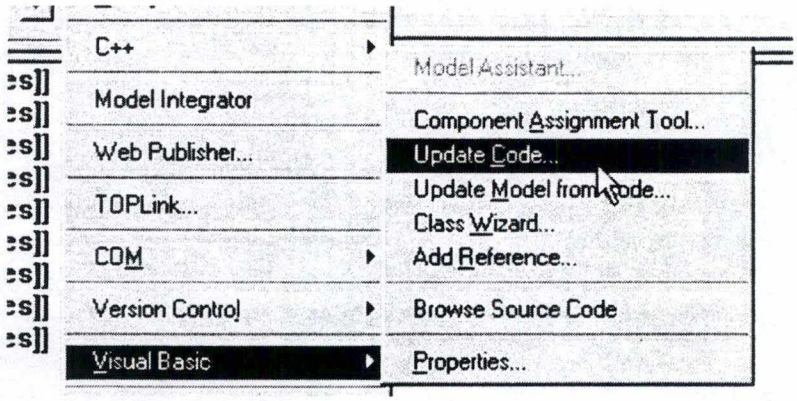
Nous n'avons pas testé la génération de code ou la reprise de code existant. Nous nous sommes donc basés sur la documentation.

6.1. Forward to Java

Le manuel « Using Rose J » consacre un chapitre complet (3) au sujet de la génération de code Java.

6.2. Forward to Visual Basic

Le chapitre 1 du manuel « Using Rose Visual Basic » nous apprend que le « Code Update Tool » nous permet de générer et mettre à jour les sources Visual Basic à partir des informations contenues dans les modèles, tout en préservant les modifications apportées par l'utilisateur depuis l'itération précédente.



6.3. Reverse from Java

Le chapitre 4 du manuel « Using Rose J » décrit la manière de repartir d'une source Java pour recréer des modèles.

6.4. Reverse from Visual Basic

Le chapitre 1 du manuel « Using Rose Visual Basic » nous apprend que le « model update tool » peut mettre à jour les modèles à partir du code Visual Basic.

6.5. Reverse from Powerbuilder

Cet outil est développé par une société tierce. Plus d'information est disponible à l'URL http://www.metex.com/Products/rpb.asp.

6.6. Round-Trip Java

A la fin du chapitre 3 du manuel « Using Rose J » consacré à la génération du code, il est mentionné que le code est préservé lors d'une réimportation des fichiers classes permettant ainsi une approche itérative.

L'outil maintient des commentaires (/** @roseuid */) dans les sources afin de faciliter le mapping entre les modèles et le code modifié.

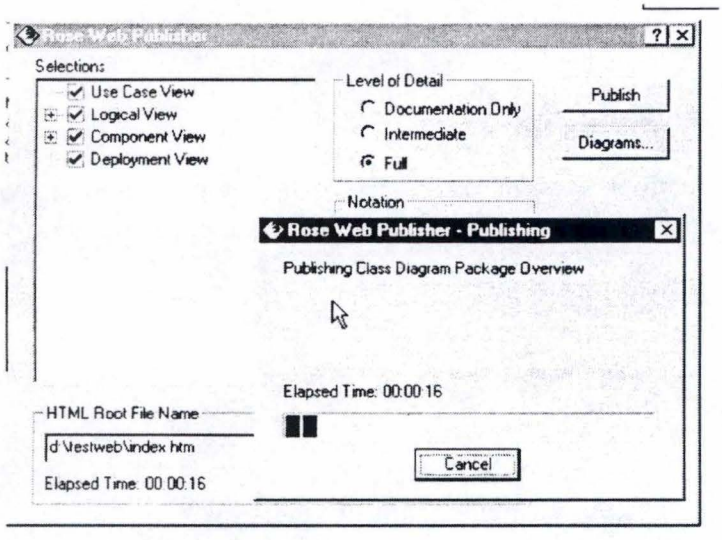
7. GESTION DE LA DOCUMENTATION (DU PROJET)

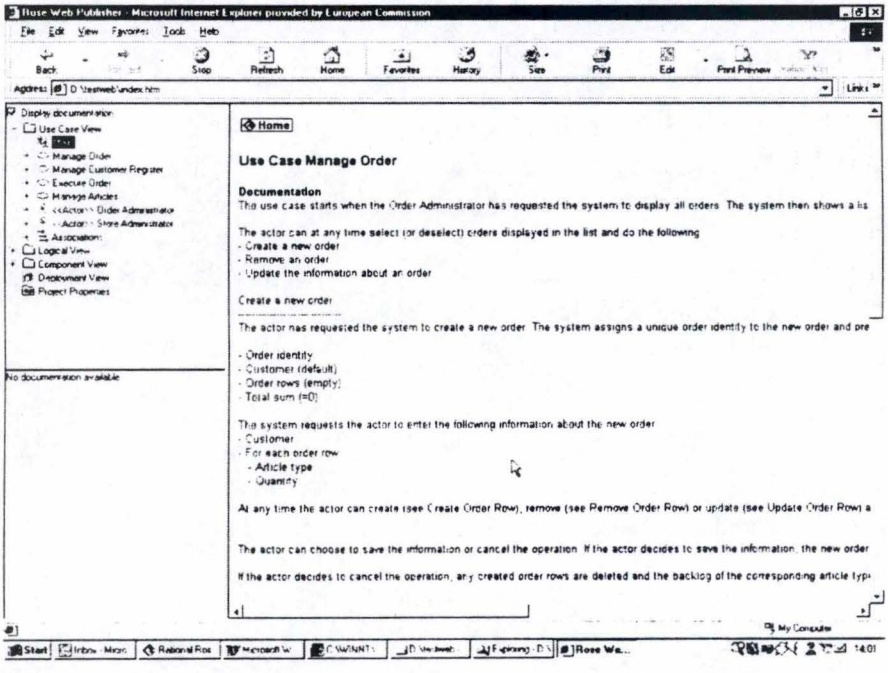
7.1. Génération de documentation sur base des artefacts du projet

L'utilisation du script « reportgen » permet de générer un mini rapport Word sur base du diagramme de classes, mais c'est fort limité. Nous considèrerons donc qu'il n'est pas possible de générer directement de la documentation en format Word. Cela doit se faire via l'outil optionnel SoDA. La génération de documentation en format HTML pur est possible (voir 7.2). Il n'est pas possible de générer du XML en lieu et place de l'HTML.

7.2. Publication de la documentation vers un site Web du projet

Sous le menu « tools », une fonction « Web Publisher » réalise cette publication. Le format de la documentation générée est fixe.



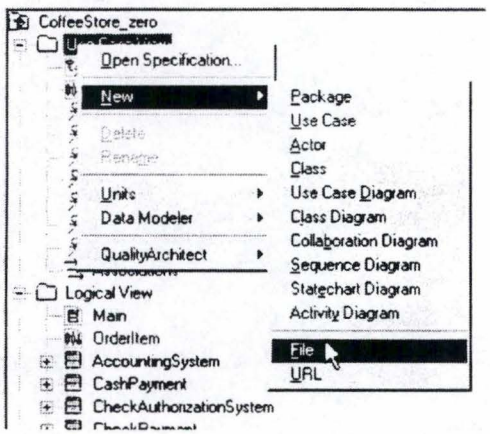


7.3. Adaptabilité du format de la documentation générée

La possibilité de configurer le contenu des rapports créés à partir du projet nécessite l'outil optionnel « SoDA » Il est possible de définir des rapports types avec cet outil.

7.4. Intégration de documents externes dans le projet

Il est possible d'intégrer dans l'outil des documents externes. Des raccourcis sont alors créés dans le browser du projet, au niveau de n'importe quel package vers les documents concernés.



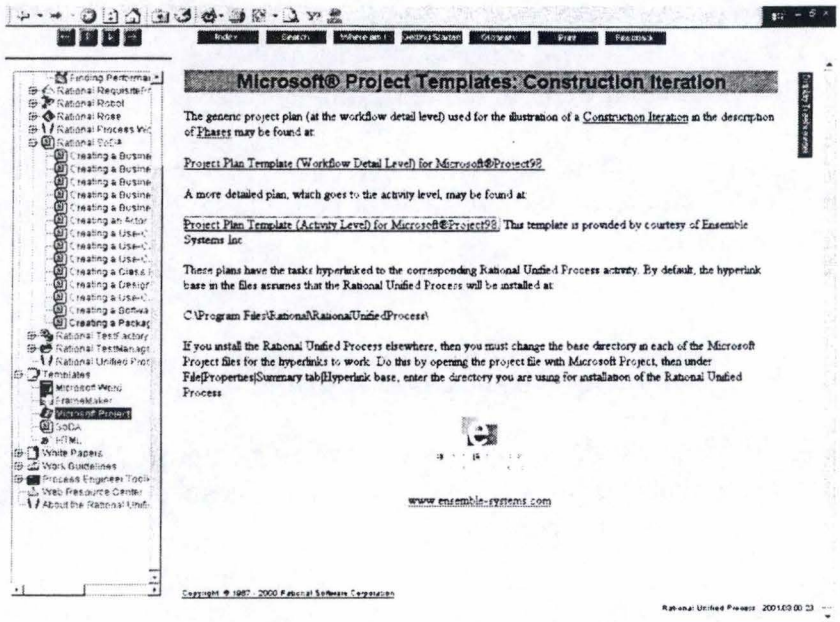
Le document apparaît alors dans le browser avec une icône adaptée à son contenu. Lorsqu'on double clique, l'application (par exemple office ou un browser) est lancée.

7.5. Multilinguisme dans la documentation du projet

Cette fonctionnalité n'est pas disponible.

7.6. Lien avec un gestionnaire de projets

Il n'y a pas d'intégration à proprement parler avec MS Project mais il existe, dans le cadre de RUP (un produit optionnel), des templates MS Project qui supportent la méthodologie.



Les produits optionnels Clearquest (gestion des demandes de changement) et Requisite Pro (Gestion des exigences) s'intègrent avec Microsoft Project 2000.

8. GESTION DES EXIGENCES

Cette fonctionnalité fait l'objet d'un produit spécifique et optionnel appelé « Requisite Pro » Plus d'information sur ce produit est disponible sous http://www.rational.com/products/reqpro/index.jsp

8.1. Collecte des exigences

Les exigences sont rassemblées dans une base de données.

8.2. Liens entre exigences et artefacts

Les liens sont à réaliser manuellement car il n'existe pas nécessairement une correspondance un à un entre les exigences et les artefacts (par exemple les use cases).

8.3. Tableaux de bord

L'outil permet de générer des « traceability tree » et « traceability matrix » afin de suivre l'évolution et le niveau d'implémentation des différents requirements.

9. GESTION DES TESTS

La gestion des tests est prise en charge par un produit optionnel appelé « Test Manager » (http://www.rational.com/products/testmanager/index.jsp).

9.1. Collecte des tests à réaliser

L'outil optionnel « Testmanager » permet de collecter les tests.

9.2. Aide à la définition des tests

L'outil optionnel « QualityArchitect » permet de générer des stubs pour les tests sur base des informations contenues dans l'outil Rose (sequence diagram) ou dans Requisite Pro. Il gère également un « data pool » qui permet d'exprimer les domaines acceptables en entrée et en sortie des modules testés. Ceci ne fonctionne que pour des composants EJB ou COM.

Des scripts peuvent être préparés pour une exécution dans Robot (http://www.rational.com/products/robot/index.jsp), un autre produit optionnel.

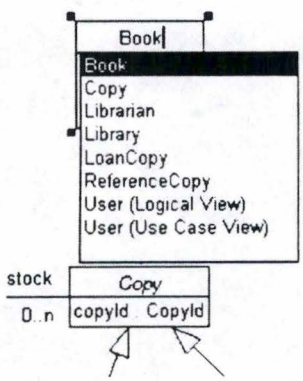
9.3. Tableaux de bord d'avancement / Succès des tests

L'outil « Testmanager » permet également de générer des tableaux de bord et d'en gérer le statut.

10. ERGONOMIE

10.1. Propositions contextuelles

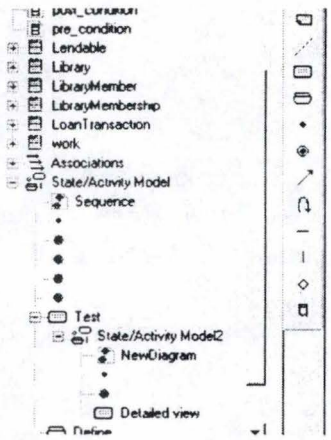
Le système propose effectivement des éléments adaptés au contexte courant.

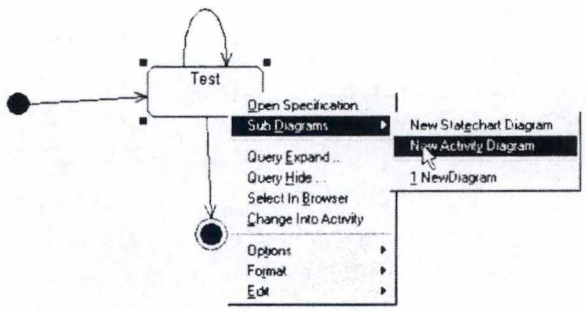


10.2. Facilité de navigation

Les capacités de navigation sont agréables à utiliser. La barre d'outil dispose d'un outil zoom in, zoom out, fit in window et unfit in window. La souris avec molette est supportée, ce qui améliore le confort d'utilisation.

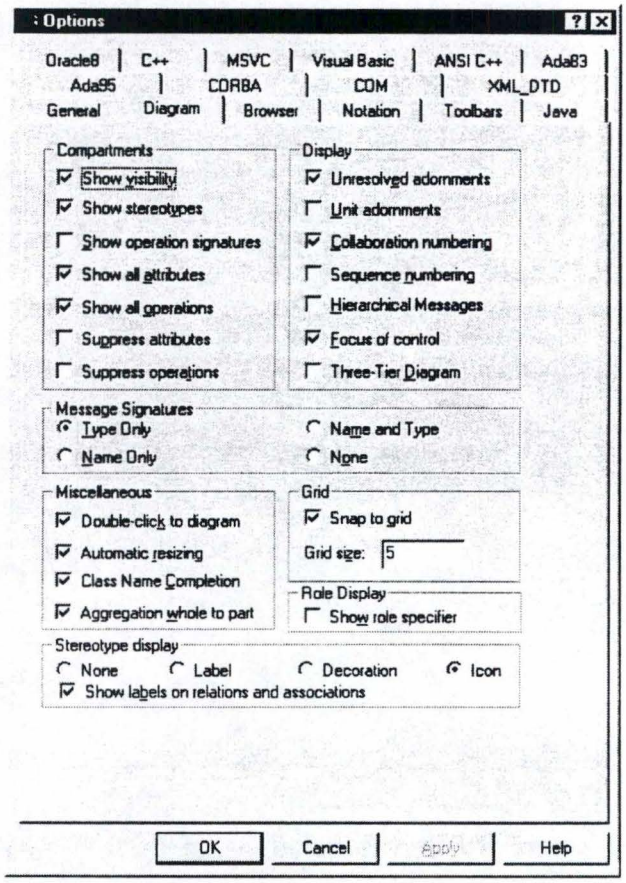
Le menu contextuel qui apparaît en cliquant avec le bouton droit de la souris permet de définir un sous diagramme pour les diagrammes d'activité et d'état, ce qui est appréciable pour améliorer la compréhension des diagrammes fort complexes.





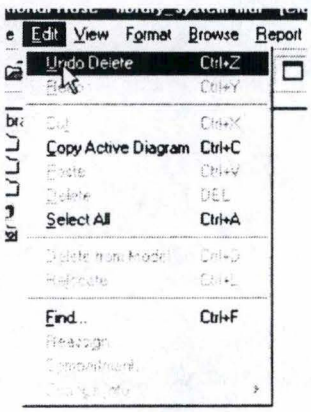
10.3. Adaptabilité de l'interface

L'outil dispose de nombreuses options de configuration, tant du point de vue d'utilisation de l'interface, affichage des diagrammes et options spécifiques à l'environnement cible.



10.4. Fonctions faire et défaire

Les fonctions faire et défaire sont présentes mais ne sont pas toujours fonctionnelles et ne supportent qu'un seul niveau.



10.5. Stabilité

La stabilité de l'outil nous a paru bonne.

10.6. Performance

La performance de l'outil est également bonne. Toutefois, le produit nous a paru très lent à charger (environ cinq minutes sur une station récente pour le chargement avec un template J2EE). Nous ne pénaliserons pas trop l'outil pour ce défaut car c'est quelque chose que l'on fait en principe une seule fois sur la journée.

11. PLATES-FORMES

11.1. Tourne sur toutes les plates-formes de l'utilisateur

D'après la brochure informative concernant le produit[3], l'outil tourne sous NT4, Windows 2000, sous Linux Redhat 6.2 et 7.

11.2. Fonctionnant dans un environnement hétérogène (clients)

L'échange de fichiers entre des clients Unix (Linux) et Windows ne devrait pas poser de problème car le format PTL est identique sur les deux plates-formes (in « Guide to Team Developpement »).

11.3. Fonctionnement avec un repository sur une autre plate-forme

L'outil fonctionnant sur base de fichiers partagés, la localisation de la « base de fichiers » a peu d'importance, dès que la plate-forme fournit un mécanisme de partage des fichiers entre les clients et le serveur accueillant ces fichiers (NFS, Samba...)

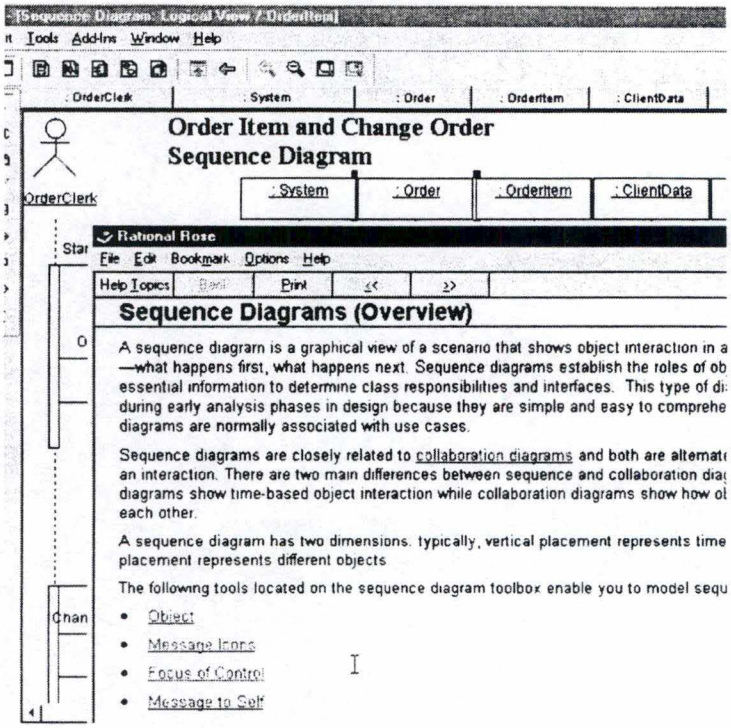
12. AIDE À L'UTILISATION DE L'OUTIL

12.1. Aide intégrée contextuelle

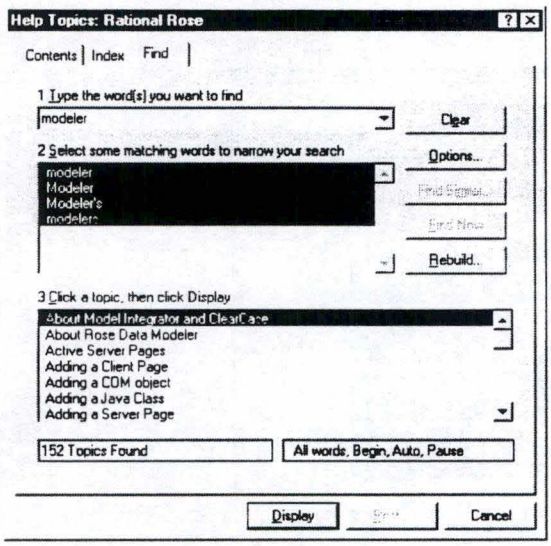
L'aide est bien une aide contextuelle telle qu'on la trouve classiquement dans Windows.

---

[3] « Brochure » sous http://www.rational.com/products/rose



Ce système intègre un outil de recherche :



12.2. Documentation papier

L'outil est fourni avec un CDROM de documentation où l'on trouve la documentation en format Acrobat. Cette formule permet donc d'imprimer ce que l'on estime nécessaire. Le format est proche d'un manuel traditionnel.

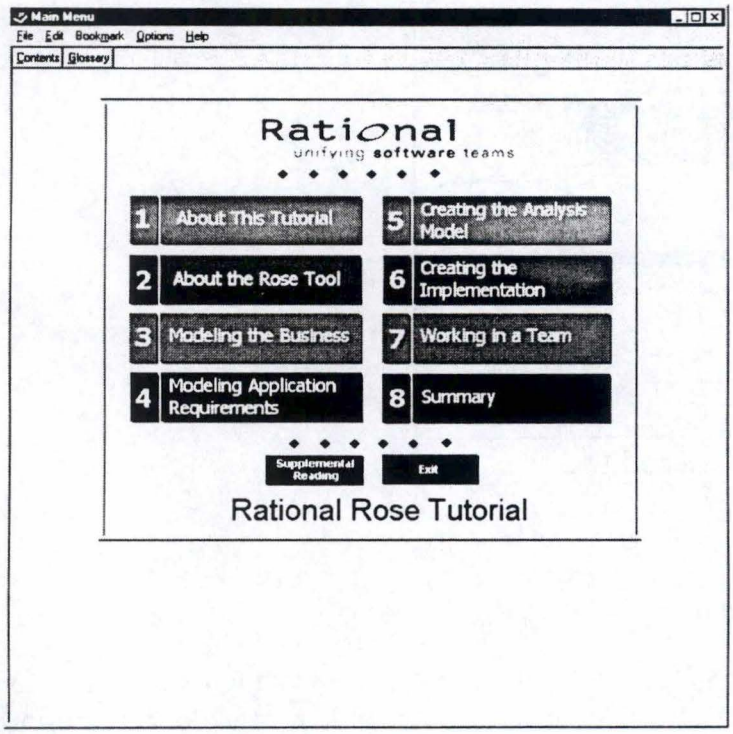
Un kit de documentation peut être acquis en sus. Le kit mis à notre disposition (version 2000.02.10) contenait les ouvrages suivants :

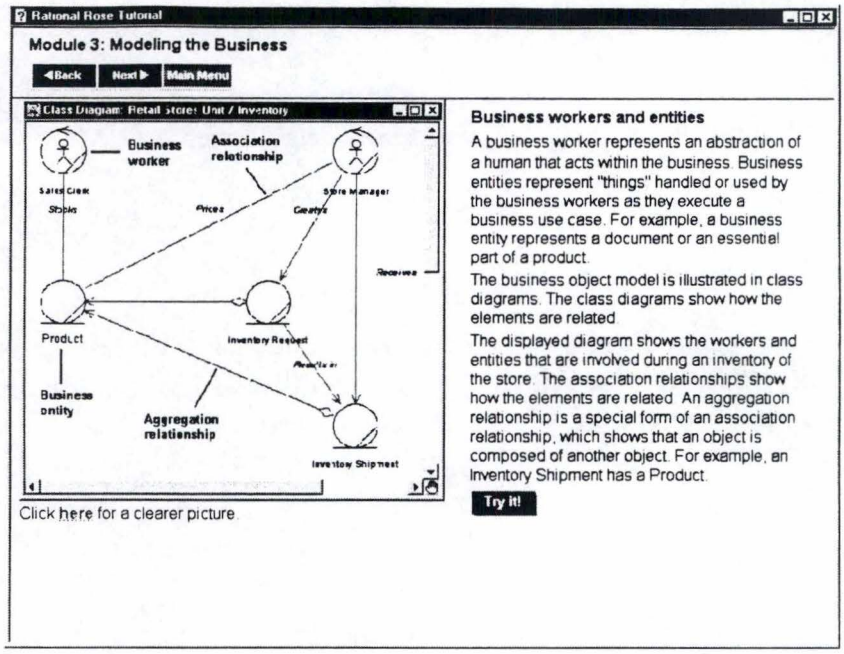
- Using Rose
- Rose Extensibility User's Guide

- Rose Extensibility Reference
- Using Rose Oracle8
- Using Rosa CORBA
- Using Rose C++
- Using Rose Visual C++
- Using Rose Visual Basic
- Forward and Reverse Engineering with ADA 83
- Forward and Reverse Engineering with ADA 95

12.3. Tutorial

Le CD contenant le produit contient un exemple de modélisation expliqué. La manière de construire les modèles y est détaillée et les corrigés sont disponibles en format Rose.





Le set de documentation contient également un livre « tutorial » :

- Visual Modeling with Rational Rose 2000 and UML par Terry Quatrany chez Addison-Wesley

12.4. Formation

L'éditeur organise des formations sur base d'un catalogue assez complet. On peut consulter les formations disponibles à l'adresse http://www.rational.com/worldwide/benelux/courses/fr_courses.jsp

13. EDITEUR

13.1. Quote-part du marché

Nous n'avons pas trouvé de chiffres concernant uniquement le marché des outils UML. Toutefois, d'après une enquête indépendante concernant le marché des outils CASE, le chiffre d'affaire de Rational en l'an 2000 dans ce segment atteint près de 30% du segment de marché. Ils surpassent de loin les suivants (13,5 chez Computer Associates, 8,9 chez Oracle…)

13.2. Produit existe depuis

Le produit est apparu en 1992. Au début, il tournait uniquement sous Unix. Il utilisait la notation Booch. En 1996, il a décidé de s'aligner sur UML.

13.3. Coût d'acquisition / utilisation

L'outil est assez cher à l'acquisition. La licence node locked est limitée à une machine tandis que la licence floating peut être utilisée sur n'importe quelle machine d'un pool connecté à un serveur de licences.

13.4. Coût de mise à jour

Les mises à jour sont incluses dans le cadre du contrat de support.

13.5. Support

Le support est facturé pour un prix annuel de 20% du coût d'acquisition. Cela reste dans la norme.

14. Méthodologie

14.1. Existence d'une méthodologie d'utilisation de l'outil

Rational a développé RUP mais a su laisser son outil suffisamment indépendant de la méthodologie pour rester ouvert et permettre l'utilisation d'une autre méthodologie au cas où ce serait souhaité.

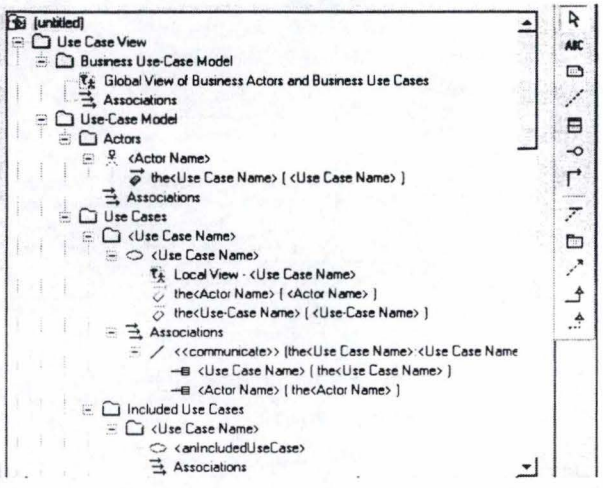
14.2. Adaptabilité

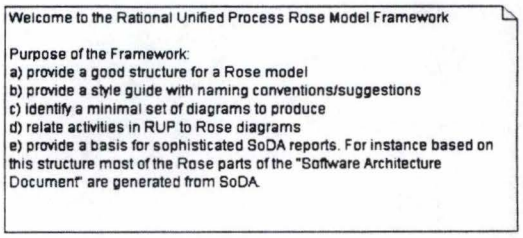
L'approche de Rational est de fournir la méthodologie comme un template. Il ne devrait donc pas être impossible de définir un nouveau template pour mettre à disposition dans l'outil une autre méthodologie (p.ex. 2TUP – Two Tracks Unified Process de VALtech).

14.3. Méthodologie de l'éditeur

Rational a développé, sous la houlette de Philippe Krutchen une méthodologie nommée RUP[4] pour Rational Unified Process.

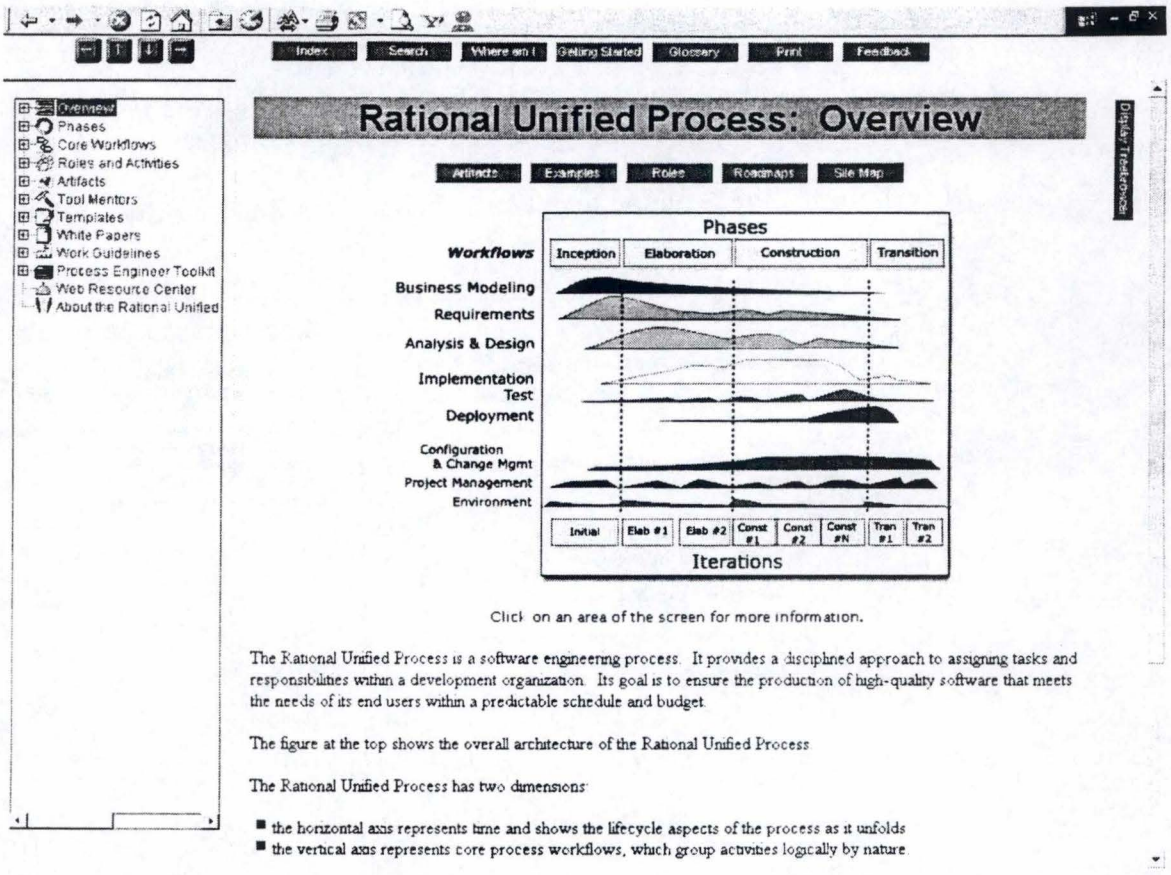
Celle-ci est livrée comme template standard dans Rational Rose Entrerprise Edition.





Pour profiter pleinement du R.U.P., il faut toutefois l'acquérir comme un produit optionnel séparé. Il est alors livré sous format d'un site web documentant tout le processus, avec templates et mentors.

[4] http://www.rational.com/products/rup/index.jsp



The Rational Unified Process is a software engineering process. It provides a disciplined approach to assigning tasks and responsibilities within a development organization. Its goal is to ensure the production of high-quality software that meets the needs of its end users within a predictable schedule and budget.

The figure at the top shows the overall architecture of the Rational Unified Process.

The Rational Unified Process has two dimensions:

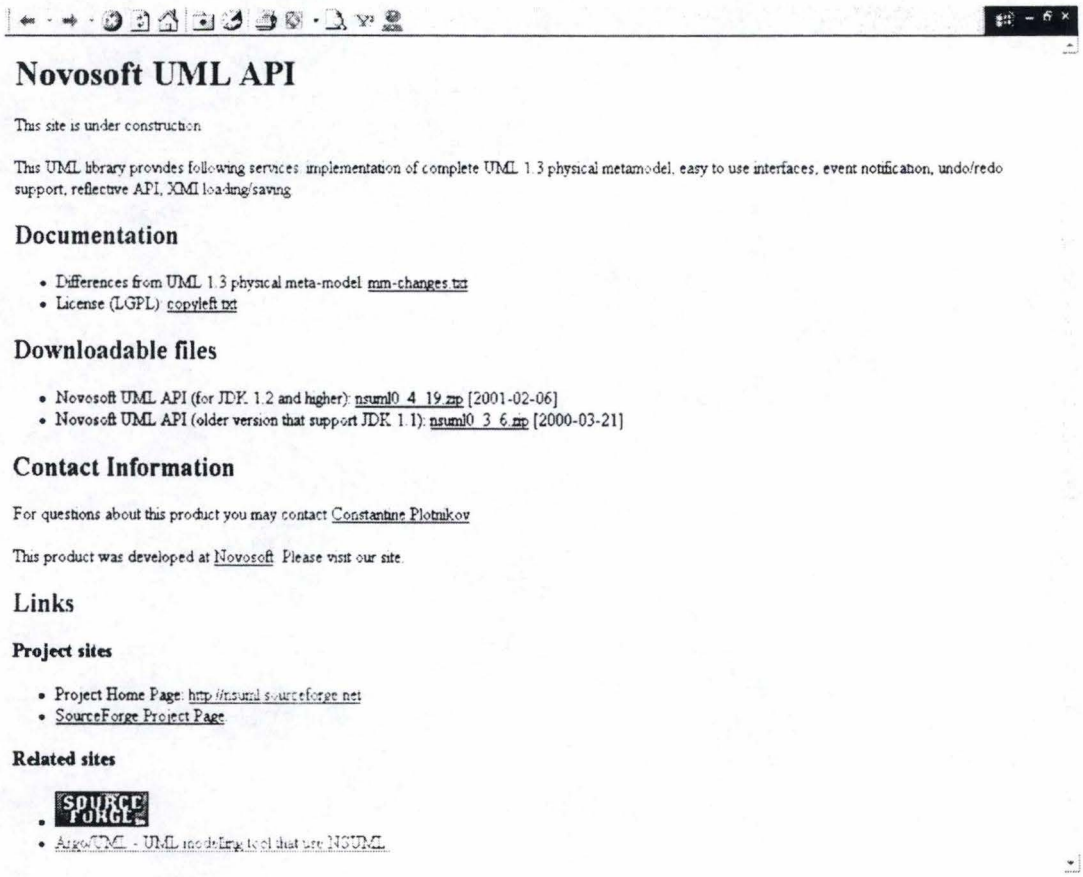
- the horizontal axis represents time and shows the lifecycle aspects of the process as it unfolds
- the vertical axis represents core process workflows, which group activities logically by nature.

Annexe 2
Illustrations de l'analyse d'Argo/UML

1. LANGAGE

1.1. Support de la version courante du langage

Argo/UML utilise une librairie Novosoft pour l'implémentation du métamodèle d'UML. Le site de l'éditeur d'Argo (Tigris) affirme que la dépendance d'Argo est telle que l'outil peut s'adapter en quelques jours à une nouvelle spécification du métamodèle (ref. http://argouml.tigris.org/features.html)



Novosoft UML API

This site is under construction

This UML library provides following services: implementation of complete UML 1.3 physical metamodel, easy to use interfaces, event notification, undo/redo support, reflective API, XMI loading/saving

Documentation

- Differences from UML 1.3 physical meta-model: mm-changes.txt
- License (LGPL): copyleft.txt

Downloadable files

- Novosoft UML API (for JDK 1.2 and higher): nsuml0_4_19.zip [2001-02-06]
- Novosoft UML API (older version that support JDK 1.1): nsuml0_3_6.zip [2000-03-21]

Contact Information

For questions about this product you may contact Constantine Plotnikov

This product was developed at Novosoft. Please visit our site.

Links

Project sites

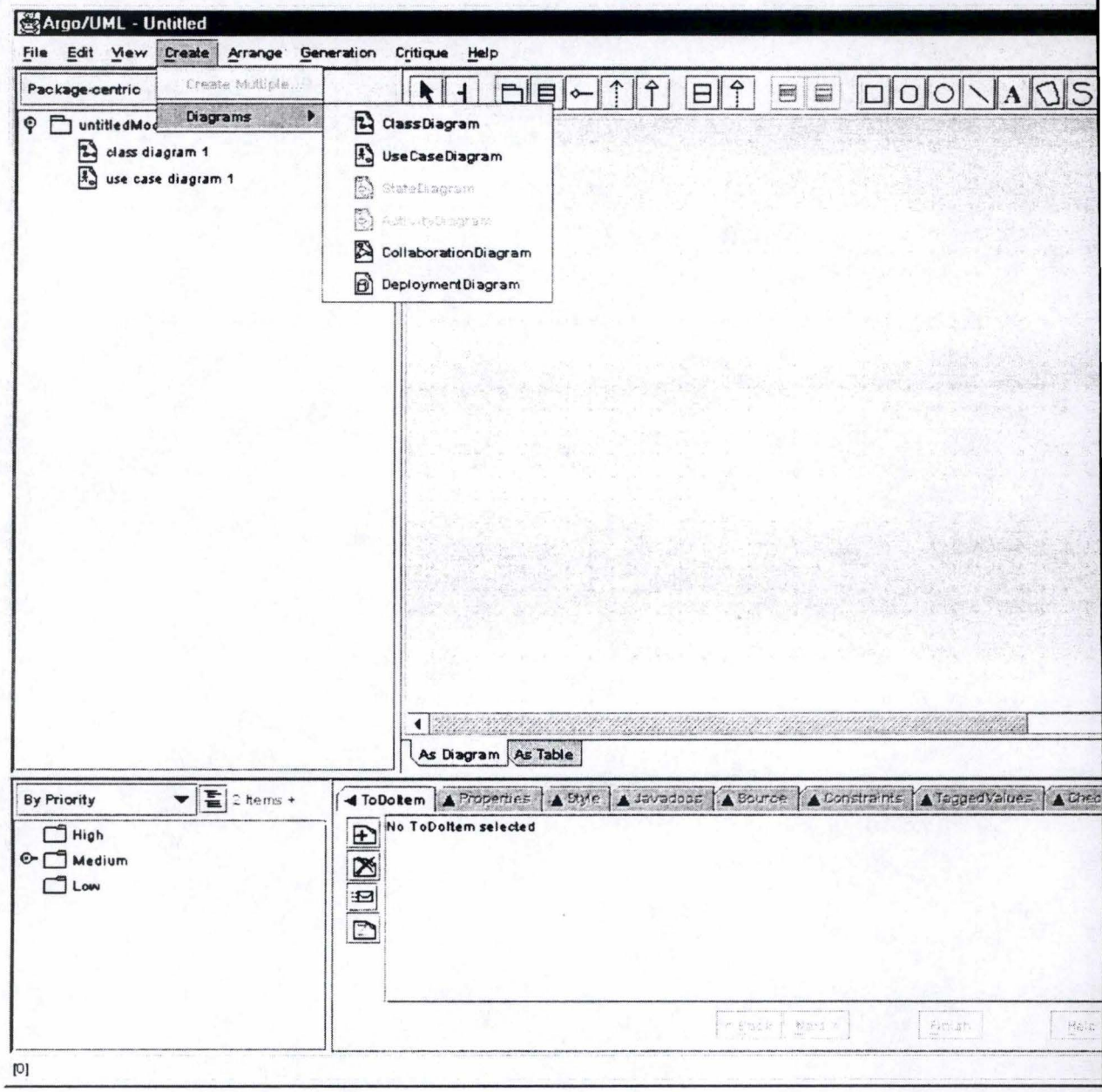
- Project Home Page: http://nsuml.sourceforge.net
- SourceForge Project Page

Related sites

- SOURCEFORGE
- Argo/UML - UML modeling tool that uses NSUML

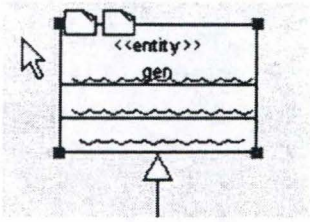
1.2. Support de tous les diagrammes

Argo/UML ne supporte pas tous les diagrammes. Le Diagramme de séquence ne sera supporté que dans la version 0.9 (en cours de développement). Le diagramme objet est absent mais peut être simulé en créant un diagramme de collaboration.

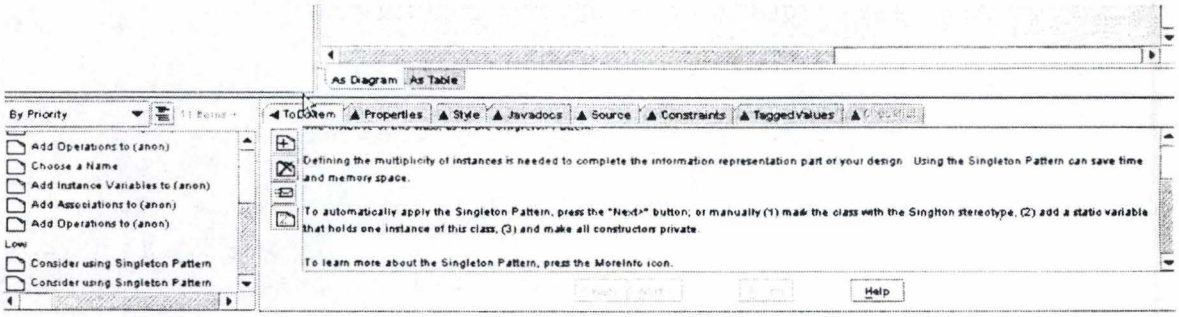


1.3. Extensibilité du langage

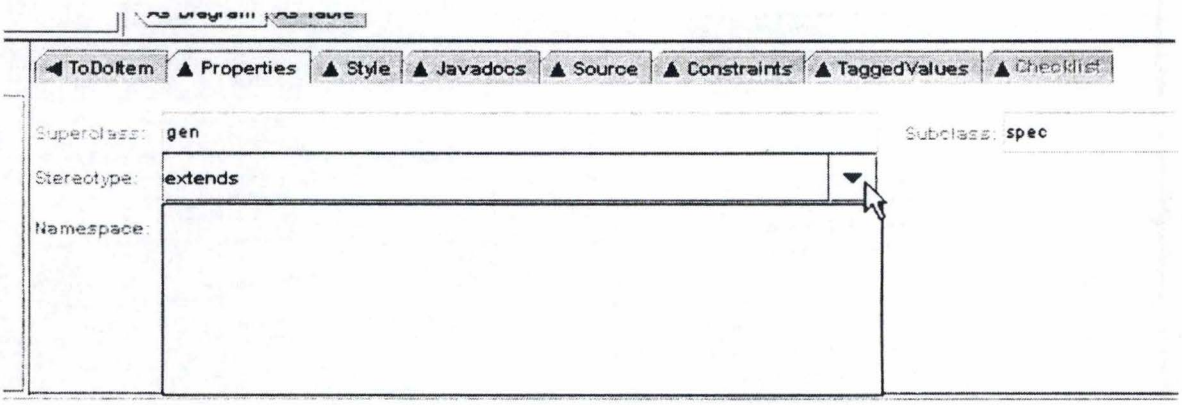
Les stéréotypes sont supportés par l'outil.



L'outil en suggère parfois l'utilisation.



Leur utilisation est toutefois peu ergonomique : la liste descendante qui permet de sélectionner les stéréotypes est vide. Il faut y taper le stéréotype qui ne sera sauvegardé que s'il est valide…



Les stéréotypes acceptés sont « Boundary », « Case Worker », « Control », « Entity », « Implementation Class », « Internal Worker », « Machine », « Metaclass », « Organization », « Person », « Powertype », « Process », « Thread », « Type », « Utility », « Worker »

1.4. Possibilité d'adaptation à de nouveaux types de projets

Cette souplesse était souhaitable. Malheureusement, l'outil a été conçu pour générer du Java pur et nous n'avons trouvé aucun module additionnel permettant de modéliser d'autres environnements (un site web par exemple, J2EE, EJB).

1.5. Extensibilité par scripting

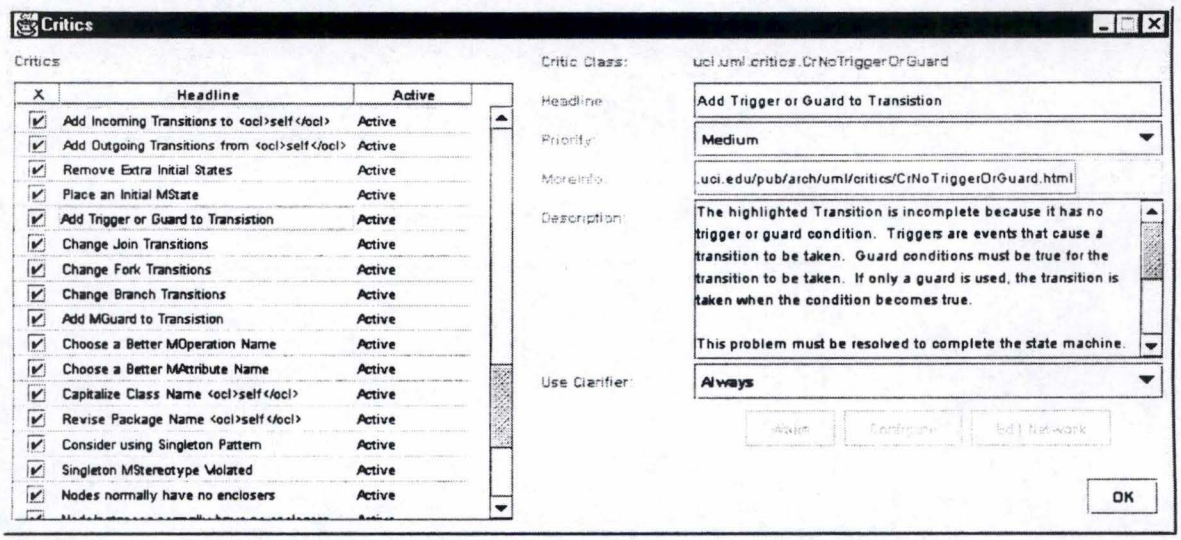
Pas utile. Il n'y a pas de langage script propre à l'outil. Les sources de l'outil (Java) sont disponibles (la philosophie open source veut qu'on puisse le modifier pour l'adapter aux besoins personnels). Cependant, cette approche n'offre pas la souplesse d'un langage script et comporte des risques d'incompatibilité avec de nouvelles versions du produit.

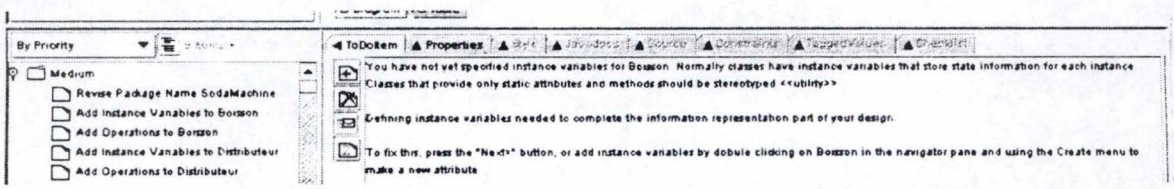
1.6. Support d'autres notations

Pas utile et absent. L'outil est vraiment orienté vers UML.

1.7. Outil de vérification de la cohérence

L'outil dispose d'une série de règles (critiques) ; ces critiques sont regroupées en fonction du contexte dans le panneau des choses à faire (« to do ») C'est ce qu'ils appellent du « cognitive support ».





2. MÉTA DONNÉES

2.1. Documentation du schéma des méta données.

Cette documentation est disponible parmi les sources du projet : http://argouml.tigris.org/source/browse/argouml/src/uci/docs/ , tant pour le format XML (en format DTD) que pour la version SQL (en DDL). Remarquons toutefois que ces schémas ne sont pas commentés.

2.2. Facilité d'accès

Il n'y a pas de langage de scripting à proprement dit et donc, la facilité est toute relative ; mais cela peut être fait en JDBC standard ou avec un outil d'accès à des informations XMI.

2.3. Extensibilité du métamodèle.

Jugée comme non utile, l'ajout d'attributs ou de tables dans la base de données ne devrait pas poser de problèmes. Cela nous semble moins aisé avec les fichiers XMI (mode de fonctionnement standard)

2.4. Sauvegarde/ Restauration du métamodèle.

Jugée comme non utile, rien n'est prévu à ce titre. On peut toutefois émettre l'hypothèse qu'il serait possible de développer une solution lorsque l'outil est configuré pour stocker les méta données dans un SGDB.

2.5. Exportation vers XMI / PTL

Le format standard de sauvegarde est basé sur XMI et PGML. Il n'y a pas de possibilité d'exportation vers PTL (Rose)

2.6. Importation XMI / PTL

XMI est le format standard de sauvegarde et donc l'import devrait être possible. En ce qui concerne les fichiers provenant de Rose, le site renvoie vers un outil payant de la société Meta Integration qui permet de récupérer les méta données (pas les diagrammes)

2.7. Travail de groupe

Si l'installation de l'outil avec une base de données permet d'y sauvegarder les méta données et les diagrammes « use cases », il faut utiliser des

fonctions charger et sauver pour accéder au projet. On se trouve donc dans un outil de type check-out / check-in. L'outil n'est pas, à son stade actuel, fait pour un accès de groupe.

2.7.1. Accessibilité simultanée

Jugée non utile et non prévue.

2.7.2. Notions de sites et de réplication

Jugée non utile et non prévue.

2.7.3. Notion d'utilisateurs

Jugée non utile et non prévue.

2.7.4. Notion de groupes d'utilisateurs

Jugée non utile et non prévue.

2.7.5. Possibilité de droits d'accès sur le projet

Jugée non utile et non prévue.

2.7.6. Gestion des rôles

Jugée non utile et non prévue.

3. GESTION DE LA PERSISTANCE

Aucune facilité n'est présente dans le logiciel pour la création et la modification d'une Base de Données destinée à assurer la persistance des objets manipulés au sein du projet, ni en générant du DDL, ni en permettant d'intégrer un schéma Entité / Relation. C'est, nous semble t'il, une grosse lacune.

3.1. Mapping objet vers physique

Etait souhaitable mais aucune aide n'est présente de l'outil.

3.2. Support des procédures stockées

Jugée non utile et absent.

3.3. Création / modification de la base

3.3.1. Création/Modification via DDL

Etait souhaitable mais est absent de l'outil.

3.3.2. Reprise d'un DDL existant

Jugée non utile et absent.

3.3.3. Round-trip avec DDL

Jugée non utile et absent.

4. SUPPORT ET INTÉGRATION DES FRAMEWORKS, TRAVAIL PAR COMPOSANTS

L'outil ne dispose d'aucune méthode pour isoler des composants et les publier à l'extérieur. Par ailleurs, les groupes de discussions sont plus focalisés sur les développements à apporter à l'outil et l'aide dans son utilisation qu'à l'échange de composants ou de patterns.

4.1. Disponibilité de modèles pour frameworks spécifiques

Jugée non utile et absente.

4.2. Disponibilité de patterns

Jugée utile mais absente

4.3. Support d'architecture

Jugée utile mais absente

4.4. Intégration avec environnements de développement

Jugée non utile et absente ; Gentleware a d'ailleurs étendu l'outil (qui s'appelle alors Poséidon) pour supporter Netbeans et Forté.

4.5. Création de composants

Jugée utile mais absente. La liste de tâches à réaliser (http://argouml.tigris.org/jojar.html) contient d'ailleurs un support des JavaBeans.

5. GESTION DES VERSIONS

L'outil ne comprend aucune gestion intégrée des versions des modèles. Gentleware indique cette fonctionnalité comme une des évolutions probables de son outil Poséidon, basé sur Argo/UML.

5.1. Journalisation des modifications avec points de reprise

Jugée non utile et absent.

5.2. Comparaison de versions

Jugée non utile et absent.

5.3. Gestion des requêtes de changement

Jugée non utile et absent. Les listes « to do » pourraient être utilisées pour apporter une fonctionnalité de base.

5.4. Analyse de l'impact d'une modification

Jugée non utile et absent.

6. GESTION DU CODE

6.1. Forward to Java

L'outil génère sans problème particulier du code Java (et pour l'instant, uniquement Java). Il permet de générer non seulement la structure des classes mais également du code à partir des contraintes O.C.L.

6.2. Reverse from Java

Jugée non utile. Argo/UML ne possède pas de fonction de reprise de code Java. Cette fonctionnalité est présente dans Poséidon de Gentleware (un outil basé sur le code Argo/UML disponible en licence O.P.L.).

6.3. Round-Trip Java

Jugé utile, le round-trip n'est pas possible dans Argo/UML. Il ne l'est d'ailleurs pas non plus dans Poséidon, qui permet pourtant le reverse engineering en Java. Le Round-Trip n'est pas simplement la combinaison du forward et du reverse ; il faut pouvoir gérer la (re)synchronisation.

7. GESTION DE LA DOCUMENTATION (DU PROJET)

Malheureusement, l'outil ne prévoit rien qui permette de gérer de la documentation en dehors de lui-même. Il n'est pas possible de générer un document ou un site HTML en partant des modèles d'un projet.

L'outil gère bien la norme JavaDoc mais celle-ci concerne les sources Java et non l'ensemble des Artefacts du projet.

7.1. Génération de documentation sur base des artefacts du projet

Cette exigence (indispensable) ne peut être satisfaite. La génération d'un site HTML à partir d'un modèle figure bien dans la liste de tâches (http://argouml.tigris.org/jojar.html) mais est absent de la version actuelle. L'export de diagrammes en formats divers est une maigre consolation : il appartient à l'utilisateur d'intégrer ces graphiques un à un dans un document. Il lui appartient d'en maintenir la cohérence lors de l'évolution des diagrammes au sein de l'outil.

7.2. Publication de la documentation vers un site Web du projet

Jugé non utile et absent (voir ci-dessus)

7.3. Adaptabilité du format de la documentation générée

Jugé non utile et absent puisqu'il n'y a pas de génération de documentation.

7.4. Intégration de documents externes dans le projet

Jugé non utile et absent.

7.5. Multilinguisme dans la documentation du projet

Jugé non utile et non prévu dans l'outil.

7.6. Lien avec un gestionnaire de projets

Jugé non utile et absent.

8. Gestion des exigences

Rien n'est prévu en tant que tel dans l'outil pour collecter les exigences et en vérifier le niveau d'implémentation. Le mécanisme des listes « to do » permet de simuler une forme rudimentaire de suivi de ces exigences.

8.1. Collecte des exigences

Jugé non utile. Absent mais peut être simulé avec les listes « to do ».

8.2. Liens entre exigences et artefacts

Jugé non utile et absent.

8.3. Tableaux de bord

Jugé non utile et absent.

9. Gestion des tests

De même que pour les exigences, l'outil ne prévoit rien de spécifique pour la gestion des tests. Le mécanisme des listes « to do » permet également de simuler une

forme rudimentaire de suivi des tests, mais il est laissé au bon vouloir de son utilisateur.

9.1. Collecte des tests à réaliser

Jugé non utile. Absent mais peut être simulé avec les listes « to do ».

9.2. Aide à la définition des tests

Jugé non utile et absent.

9.3. Tableaux de bord d'avancement / Succès des tests

Jugé non utile et absent.

10. Ergonomie

10.1. Propositions contextuelles

Le panneau des propriétés pour l'objet sélectionné est adapté au type d'objet sur lequel on travaille. Toutefois, dans le diagramme lui-même, les mécanismes contextuels mériteraient d'être développés.

10.2. Facilité de navigation

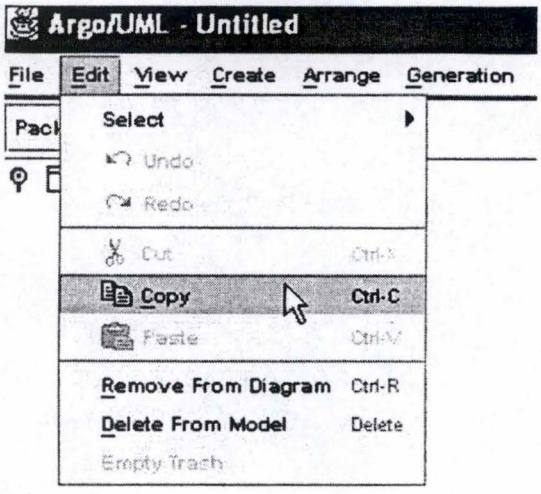
Cette fonctionnalité souhaitable ne nous semble pas être suffisamment développée dans l'outil. Il faut utiliser les scroll bar pour se déplacer dans le diagramme. L'utilisation d'une souris équipée d'une roulette est sans effet et il n'y a pas de fonction zoom.

10.3. Adaptabilité de l'interface

Quelques petites choses sont configurables mais les modifications semblent ne pas être conservées automatiquement d'une exécution à l'autre. C'est peu agréable.

10.4. Fonctions faire et défaire

Les options faire et défaire, pourtant souhaitables sont présentes dans le menu mais sont non fonctionnelles (tout le temps grisées). Elles sont d'ailleurs listées dans la liste des tâches à réaliser.



10.5. Stabilité

La stabilité de l'outil nous a paru bonne.

10.6. Performance

La performance de l'outil est également bonne. Les autres applications (en dehors de l'outil) souffrent parfois de la charge imposée sur le système.

11. PLATES-FORMES

11.1. Tourne sur toutes les plates-formes de l'utilisateur

L'outil est écrit en Java et peut donc tourner sur toute machine disposant d'un « Java Runtime Environment » 1.2, donc sur un grand nombre de plates-formes (dont Windows, Linux, Solaris...)

11.2. Fonctionnant dans un environnement hétérogène (clients)

Cette fonctionnalité jugée non nécessaire ne pose pas de problèmes, car le fait que l'outil soit écrit en Java entraîne une grande portabilité.

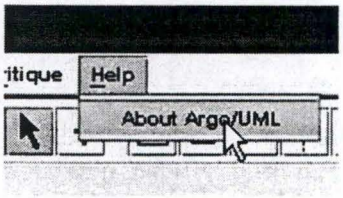
11.3. Fonctionnement avec un repository sur une autre plate-forme

Jugé non utile. Lorsqu'on stocke les méta données dans un DBMS, la seule contrainte est que ce dernier supporte JDBC. Cela ne devrait donc pas poser de problème puisque JDBC prend en charge cette dimension.

12. AIDE À L'UTILISATION DE L'OUTIL

12.1. Aide intégrée contextuelle

Cette fonctionnalité souhaitée fait cruellement défaut. L'aide se limite à une information « à propos d'Argo/UML » qui laisse sur sa faim.

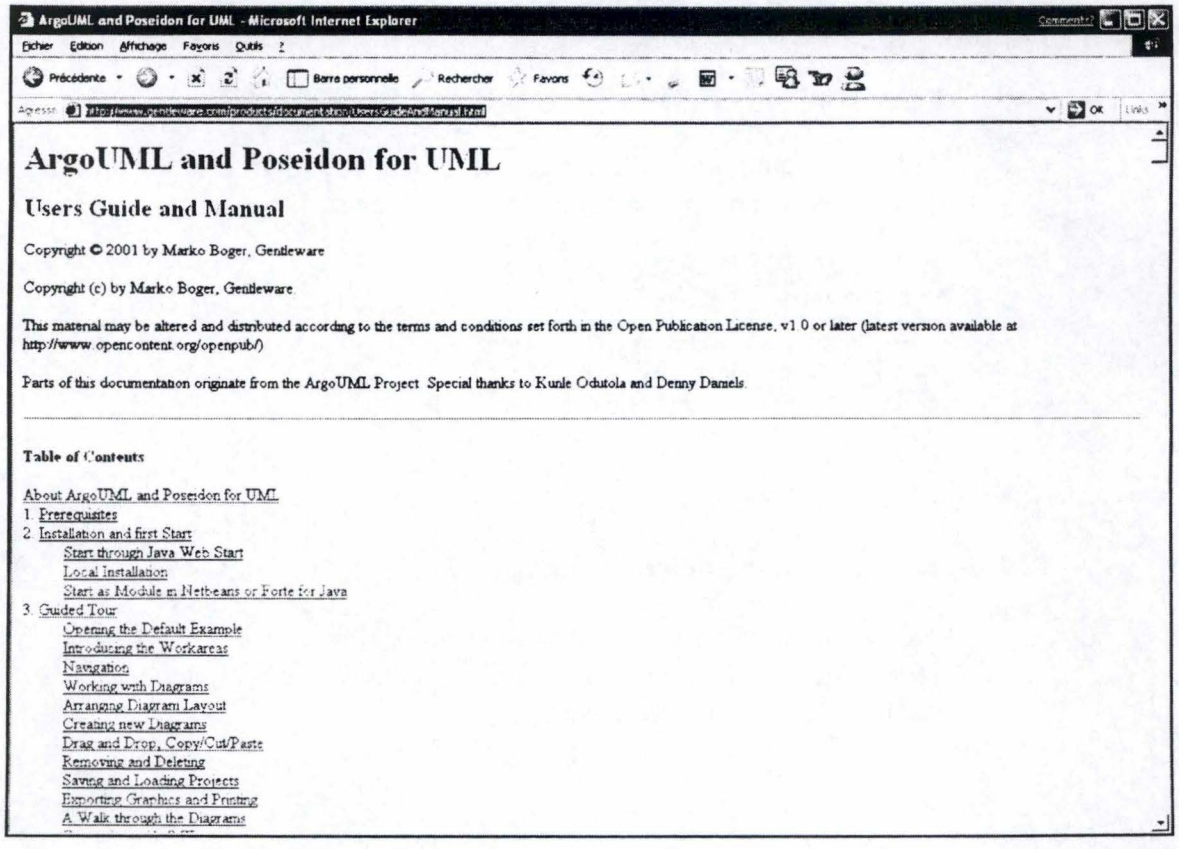


12.2. Documentation papier

Une documentation papier aurait été utile. Il n'y en a pas, fût-ce en format électronique imprimable. Il s'agit à nouveau de quelque chose qu'on retrouve dans la liste des tâches…

12.3. Tutorial

Le tutorial, jugé comme indispensable, au vu de notre public cible est un peu sommaire (http://argouml.tigris.org/tours/index.html). Heureusement, Gentleware a réalisé un tutorial plus conséquent (http://www.gentleware.com/products/documentation/UsersGuideAndManual.html) qui traite à la fois d'Argo/UML et de Poséidon (qui est basé sur UML). Malheureusement, le fait que ce document est commun aux deux outils risque de troubler l'utilisateur… qui préférera finalement télécharger Poséidon. Nous supposons que cela n'est pas innocent.



12.4. Formation

Jugée comme utile, Tigris n'offre pas de service formation. Toutefois, Gentleware en organise bien volontiers. Elle déclare d'ailleurs que formation et support sont les deux mamelles qui lui permettent de développer ces projets en logiciel libre.

13. EDITEUR

13.1. Quote-part du marché

Jugée non utile, cette information est difficile à définir, vu le caractère non commercial de l'éditeur. Ils affichent 86000 téléchargement comme référence.

13.2. Produit existe depuis

L'outil a été créé en 1998 et son développement semble encore actif.

13.3. Coût d'acquisition / utilisation

L'outil est gratuit, ce qui rencontre pleinement nos exigences.

13.4. Coût de mise à jour

Il en va de même que pour le coût d'acquisition.

13.5. Support

Le support gratuit est dispensé par les groupes de discussions. Il faut toutefois être patient pour obtenir une réponse. Un support professionnel (payant) peut être obtenu chez Gentleware.

14. MÉTHODOLOGIE

La liste des fonctionnalités de l'outil (http://argouml.tigris.org/features.html#processmodel) nous apprend qu'il y a une méthodologie Argo (qui se traduit par les actions qui peuplent la liste « to do » et qui vous guident dans le travail). Elle nous signale également qu'il est possible d'ajouter des actions à cette liste. Il serait souhaitable de pouvoir modifier les actions existantes, éventuellement d'en supprimer. Il n'est pas clair si cela est possible sans modifier les sources de l'outil.

14.1. Existence d'une méthodologie d'utilisation de l'outil

Considérée comme utile, on peut dire qu'une méthodologie existe, même si elle est peu ou mal documentée.

14.2. Adaptabilité

Une méthodologie peut être adaptée via le mécanisme des listes « to do ».

14.3. Méthodologie de l'éditeur

Comme expliqué ci avant, il existe un embryon de méthodologie dans l'outil. Il manque cependant d'un fondement théorique et de documentation qui en explique la philosophie indépendamment de l'utilisation de l'outil.