
Rétro-ingénierie dirigée par les métamodèles

Concepts, Méthodes et Outils

Jean-Marie Favre* — **Jonathan Musset****

**Université Joseph-Fourier
Grenoble, France
<http://www-adele.imag.fr/~jmfavre>*

***Obeo
95 route de gachet
44307 Nantes, France
<http://www.obeo.fr/>*

RÉSUMÉ. L'Ingénierie Dirigée par les Modèles est un thème en pleine expansion aussi bien dans le monde académique que dans le monde industriel. Bien que l'on puisse imaginer un futur basé sur l'utilisation systématique de modèles, il n'en reste pas moins que les pratiques industrielles sont centrées sur le code. Le succès de l'IDM sera non seulement lié à la prise en compte des nouveaux développements mais aussi et surtout à la prise en compte des logiciels existants voire patrimoniaux. Cet article montre que l'IDM est en fait extrêmement bien adapté à cette problématique. La notion de métamodèle, trop souvent négligée dans le domaine de la rétro-ingénierie, se révèle en fait être un point clé. La rétro-ingénierie dirigée par les métamodèles est une piste extrêmement prometteuse. Cette thématique est abordée dans cet article selon trois axes. (1) D'un point de vue conceptuel, on montre que les concepts fondamentaux de l'IDM et de la rétro-ingénierie se révèlent en fait être les mêmes. (2) D'un point de vue méthodologique, cet article montre que la notion de métamodèles est au coeur des processus de ré ingénierie globale. Le processus CacOphoNy est brièvement décrit à titre d'exemple. (3) D'un point de vue outils, on montre que les technologies de l'IDM s'appliquent à la rétro-ingénierie et à l'évolution des logiciels existants. A titre d'illustration, un exemple de cartographie dirigée par les métamodèles est présenté en utilisant l'environnement Obeo Agile Reverse basé sur Eclipse et EMF.

MOTS-CLÉS : Ingénierie Dirigée par les Modèles, IDM, Rétro-ingénierie, Evolution, Maintenance, Métamodèle, Transformation, Analyseur, Cartographie, Urbanisation, Logiciels Patrimoniaux.

1. Introduction

Les discours futuristes prévoyant la "disparition du code" et la production automatique de logiciels à partir de modèles abstraits ne doivent pas faire oublier l'état de la pratique en génie logiciel : plus d'un demi-siècle après les débuts de l'informatique, l'industrie du logiciel est toujours résolument centrée sur le code... Bien que l'*Ingénierie Dirigée par les Modèles* (IDM, MDE en anglais) soit une approche fort prometteuse [1][2][3][4], négliger l'existant et ignorer les problématiques liées à l'évolution des logiciels est sans doute le meilleur moyen de faire avorter la "révolution" que certains prédisent. D'un point de vue pragmatique nous pensons que l'IDM devrait plutôt être vue comme une évolution plus qu'une révolution. Alors que la très grande majorité des travaux existant se concentrent presque exclusivement sur la *génération de code* à partir de modèles, nous pensons indispensable d'intégrer dans les démarches IDM des fonctionnalités de *rétro-ingénierie* [5][6] permettant d'extraire des modèles à partir de code existant. Ingénierie et rétro-ingénierie doivent être vues comme deux facettes indissociables de tout processus d'évolution. Le développement n'est pas le point dur du génie logiciel, le problème est l'évolution [8][9][10].

Pour bon nombre d'informaticiens l'Ingénierie Dirigée par les Modèles a une connotation "technologie de pointe" voire futuriste, alors que Cobol a une connotation passéiste. En dépit des qualités certaines des langages mûrs et matures, ces mythes sont particulièrement répandus dans le monde académique [11]. L'association des termes Cobol et métamodèle est souvent perçue comme incongrue, notamment par les néophytes. Il n'en est pourtant rien. Bien au contraire.

D'un point de vue très pragmatique, c'est justement lorsque les langages et les technologies sont difficiles à comprendre que la plus-value de les formaliser via des métamodèles est importante [14].

Alors qu'un nombre significatif de travaux de recherche visent à définir des métamodèles pour des langages ou des technologies dont l'avenir est hypothétique, cet article s'intéresse à l'application des principes de l'IDM aux logiciels existants ; et ce tel qu'on les trouve dans l'industrie, quel que soit le niveau de maturité ou d'obsolescence des langages utilisés.

Pour fixer les idées indiquons que les applications logicielles qui sont à la base de la problématique traitée dans cet article peuvent typiquement dépasser le million de lignes de code, sont écrites dans des langages provenant de générations différentes, sont basés sur des technologies parfois ad hoc ou propriétaires. La durée de vie de tels logiciels se compte typiquement en décennies, voire en fraction de siècles [12], tout comme d'ailleurs la durée de vie de langages éprouvés comme Cobol, Fortran ou le langage C. Rappelons que selon diverses estimations environ 80% du volume de code actuellement en exploitation sur notre planète serait écrits dans des langages non structurés. Et c'est bien sur cette même planète que les approches de type MDE sont censées se développer ... [3].

Selon nous le futur de l'IDM, et notamment son adoption à une large échelle, résident tout autant dans la prise en compte des *logiciels patrimoniaux* que dans le développement "from-scratch" de logiciels, fussent ils basés sur les dernières technologies à la mode. A ce propos il est ici essentiel de rappeler que le terme *patrimoine* n'a pas une connotation négative dans le langage courant. Après tout, comme le fait remarquer Parnas dans son fameux article

"Software Aging" [8], ce sont uniquement les logiciels qui ont du succès qui atteignent une maturité et un âge avancé. Beaucoup de logiciels ne sont au contraire pas suffisamment utiles pour que l'on décide de les maintenir et de les faire évoluer.

Seul les logiciels utiles et les langages utiles traversent les décennies.

Par ailleurs il a été montré que le terme patrimonial, traditionnellement associé à des logiciels écrits en Cobol, s'appliquait en fait à toutes les générations de paradigmes et de technologies y compris ceux considérés comme "modernes" [13]. La *rétro-ingénierie des logiciels à objets* est par exemple devenu un thème important au cours des dernières années [10]. Remarquons aussi que les méthodes de développement agiles intègrent au sein même de tout processus de développement des activités de refactorisation (refactoring en anglais), Les techniques de restructuration [5], que l'on croyait jusqu'alors réservées aux logiciels anciens, intègrent ainsi le paysage des paradigmes dits modernes (p.e. l'objet), voire des paradigmes émergents. Il devrait être ainsi naturel de prévoir dès à présent les problématiques de rétro-ingénierie de logiciels à composants, de logiciels à services, de logiciels à aspects, etc. En fait, l'apparition d'un nouveau paradigme ou de nouveaux langages donne invariablement naissance à une problématique de rétro-ingénierie adaptée à ce paradigme.

Par exemple, dès la fin du millénaire précédent (notons le changement d'échelle de temps préconisé dans [12]), nous avons mis à jour l'importance de la *rétro-ingénierie des logiciels à composants*, et ce dans le contexte de la société Dassault Systèmes, le leader mondial du CAD/CAM et l'un des pionniers avec Microsoft du paradigme à composants. Nous avons développé dans ce contexte des outils de rétro-ingénierie pouvant servir aussi bien dans une problématique de développement que d'évolution [14], et ce bien avant que la vague des composants n'intègre la longue liste des modes informatiques [13].

Notons, et nous verrons que cela est particulièrement important pour bien comprendre la problématique à laquelle on s'intéresse dans cet article, que dans le cas cité précédemment la technologie utilisée était propriétaire, car développée en interne, par et pour Dassault Systèmes. Nous avons alors dû dans ce contexte extraire un *métamodèle dédié*, ou langage dédié (*domain specific language*, DSL en anglais). Ces concepts seront décrits plus précisément dans cet article. Pour l'instant notons simplement que si l'on se réfère à des technologies plus répandues, les besoins en terme de rétro-ingénierie ne diminuent pas. Il est par exemple fort probable que les logiciels basés sur des technologies "classiques" telles que les EJBs soient considérés dans quelques années comme des applications patrimoniales.

Tout environnement IDM devrait intégrer à la fois des techniques de génération de code à partir de modèle mais aussi des techniques de rétro-ingénierie pour extraire des modèles à partir du code.

La littérature concernant les approches génératives et l'IDM est particulièrement bien fournie [3]. D'autre part la communauté travaillant sur le thème de la rétro-ingénierie est déjà ancienne [5][6]. Par contre l'intersection entre les communautés rétro-ingénierie et IDM est relativement dépourvue, en tout cas au regard des enjeux industriels et scientifiques correspondants.

Cet article est un manifeste pour la *rétro-ingénierie dirigée par les métamodèles*, en tant que problématique industrielle majeure, mais aussi en temps qu'axe de recherche. Cet article établit un bref panorama de la rétro-ingénierie dirigée par les métamodèles en considérant successivement les concepts, les méthodes et les outils. En ce qui concerne les méthodes et les outils, nous n'avons pas cherché à établir un état de l'art exhaustif. Nous avons choisi au contraire d'illustrer chacun des différents aspects abordés à partir des techniques que nous avons développées au cours des dernières années. Bien évidemment des références vers d'autres travaux sont fournis tout au long de l'article.

Le reste de cet article est structuré comme suit. La section 2 ("concepts") montre que les bases de la rétro-ingénierie et de l'IDM sont en fait les mêmes. La section 3 ("méthodes") montre que les métamodèles peuvent jouer un rôle central dans les processus de rétro-ingénierie. La méthode *CacOphoNy* est brièvement présentée à titre d'illustration. La section 4 ("outils") décrit un scénario simple dans laquelle on cherche à obtenir une cartographie d'une application patrimoniale écrite en C et en Cobol. L'environnement *Acceleo d'Obeo* est présenté à titre d'exemple. Finalement la section 7 conclut et présente les nombreuses perspectives qu'ouvre ce travail.

2. CONCEPTS : Rétro-ingénierie et IDM - des bases conceptuelles identiques

Pour beaucoup d'informaticiens l'IDM et plus ou moins synonyme de génération de code. Il s'agit d'une vision bien réductrice car l'IDM a un potentiel suffisamment large pour couvrir tout le spectre du cycle de vie [2], et ceci inclut la rétro-ingénierie. En fait, comme nous allons le voir l'IDM et la rétro-ingénierie sont basés sur les mêmes concepts fondamentaux.

2.1. Définitions et concepts fondamentaux de l'IDM

Bien que les notions de *modèles*, de *métamodèles* et surtout de transformations restent à définir de manière précise, il existe toutefois un relatif consensus sur le fait que ces notions forment les piliers de l'IDM [1][2]. Rappelons ci-dessous deux définitions. Le lecteur intéressé par un ensemble plus vaste de définitions pourra se référer à la section glossaire de planetmde.org [3] ainsi qu'à nos travaux sur la "mégamodélisation" (chapitre 2 de [2], [19]).

Modèle	<i>A model is a simplification of a system built with an intended goal in mind. The model should be able to answer questions in place of the actual system. [18]</i>
Métamodèle	<i>A metamodel is a model that defines the language for expressions a model. (MOF Standard, Version 1.4)</i>

2.2. Définitions et concepts fondamentaux de la rétro-ingénierie

Après des premières années caractérisées par une terminologie foisonnante [6], les bases conceptuelles de la rétro-ingénierie et le vocable associé ont été établis en 1990 par Chikofsky et Cross dans l'article "Reverse Engineering and Design Recovery: a Taxonomy" [11] (voir table 1). Cette taxonomie fait toujours référence et c'est donc sur ces définitions qu'est basé cet article. Comme le montre la Figure 1, extraite de [5] la taxonomie est basée sur une classification des transformations qu'il est possible d'appliquer au logiciel. Le concept de *transformation* est ainsi au cœur de la rétro-ingénierie. La légende originelle de la Figure 1 indique : "la rétro-ingénierie et les processus associés sont des transformations inter ou intra

niveaux d'abstraction". Quelques années plus tard, Arnold raffine cette taxonomie en fournissant "une procédure décisionnelle pour classifier les *transformations du logiciel*" [6].

Ingénierie directe <i>Forward engineering</i>	<i>Forward engineering is the traditional process of moving from high-level abstractions and logical, implementation-independent designs to the physical implementation of a system</i>
Rétro-ingénierie <i>Reverse engineering</i>	<i>Reverse engineering is the process of analyzing a subject system with two goals in mind: (1) to identify the system's components and their interrelationships; and, (2) to create representations of the system in another form or at a higher level of abstraction</i>
Redocumentation	<i>Redocumentation is the creation or revision of alternate views semantically coherent with the examined system</i>
Rétro-conception <i>Design recovery</i>	<i>Design recovery is a subset of reverse engineering in which domain knowledge, external information, and deduction or fuzzy reasoning are added to the observations of the subject system to identify meaningful higher level abstractions beyond those obtained directly by examining the system itself</i>
Restructuration <i>Restructuring</i>	<i>Restructuring is the transformation from one representation to another at the same relative abstraction level, while preserving the subject system's external behaviour</i>
Re-ingénierie <i>Reengineering</i>	<i>Reengineering is the examination of a subject system to reconstitute it in a new form and the subsequent implementation of the new form</i>

Table 1. Taxonomie de Chikofsky and Cross.

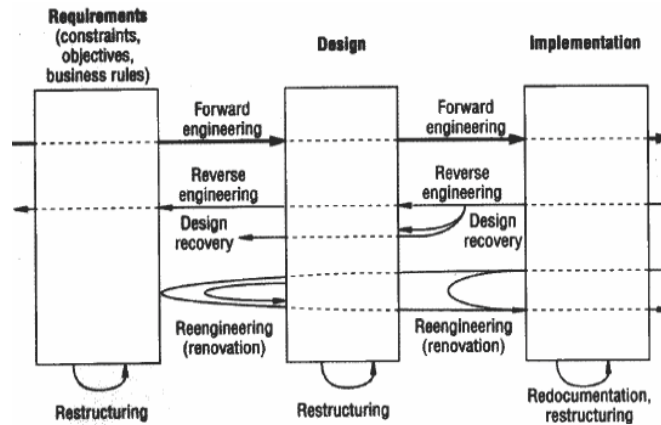


Figure 1. Classification des transformations. Figure extraite de [5]

La notion de transformation est donc clairement au cœur de la rétro-ingénierie, mais qu'en est-il de la notion de *modèle* ? Ce mot n'apparaît pas dans l'article originel [5]. Ce concept est pourtant bien présent, notamment dans la définition du terme rétro-ingénierie puisqu'il s'agit "d'analyser un système" et de créer des "représentations de ce système". On retrouve ici la relation de représentation (ReprésentationDe, μ) qui caractérise la notion de modèle [2][19]. En fait, l'article fondateur de Chikofsky et Cross pourrait être réécrit sans en changer le sens en remplaçant les mots "représentation", "vue" et "abstraction" par "modèle".

IDM et rétro-ingénierie partagent les mêmes fondements conceptuels. Ces deux démarches sont donc vouées à être intégrées.

Le lecteur attentif aura sans doute remarqué que parmi les *trois* piliers de l'IDM, on retrouve explicitement dans le domaine de la rétro-ingénierie que deux notions *modèle* et *transformation*. La notion de métamodèle a tout simplement été oubliée dans la description des fondations de la rétro-ingénierie. Quinze ans après, ce papier vise à combler ce manque

en rendant aux métamodèles la place qu'ils méritent. Mais tout d'abord continuons avec la description des fondations techniques de la rétro-ingénierie.

2.3. Architecture des outils de rétro-ingénierie

Au cours des années 90, de très nombreux outils de rétro-ingénierie ont été proposés. Arnold a fait remarquer que tous ces outils partageaient globalement l'architecture présentée dans la Figure 2. A gauche on trouve les artefacts logiciels à partir desquels sont extraites les informations. Dans la terminologie de l'IDM il s'agit du "système étudié" (cf la section 2.2). Les vues produites à droite peuvent être présentées aux ingénieurs sous de multiples formes. Ce sont des "modèles" du logiciel. La base d'information que l'on trouve au centre peut également être considérée comme un modèle du logiciel.

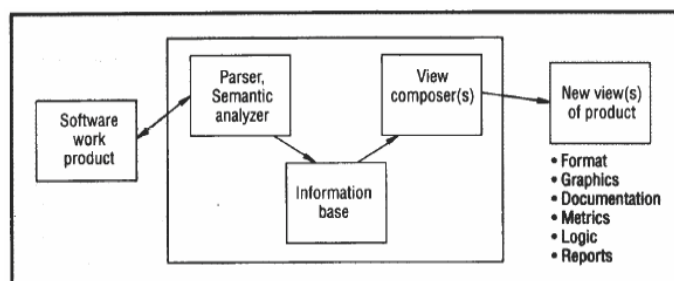


Figure 2. Architecture des outils de rétro-ingénierie [6]

Au point de vue des techniques classiques de rétro-ingénierie on peut bien évidemment citer les techniques d'analyse du logiciel (partie gauche du diagramme), mais aussi les techniques de *visualisation* (partie droite) qui sont alors particulièrement adaptées et l'on souhaite typiquement obtenir des "cartographies" qui sont des modèles visuels des applications logicielles existantes. Dans le monde de la rétro-ingénierie, on parle souvent d'environnement d'*exploration des logiciels* (e.g. [20]), car il s'agit de "naviguer" d'un modèle à l'autre et ce de manière interactive.

L'un des défis auquel les environnements de rétro-ingénierie doivent faire face consiste à prendre en compte la multiplicité des sources d'informations disponibles. Cet aspect n'a hélas pas été clairement identifié dans les travaux fondateurs en rétro-ingénierie [5][6]. Par exemple remarquons qu'une et une seule boîte est présente à gauche de la Figure 2. Bien que cette boîte soit à juste titre appelée "produit logiciel", elle a été pendant très longtemps assimilée à tort au code source, qui n'est en fait qu'un aspect particulier du logiciel.

2.4. Défi n° 1 : rendre flexible les outils de rétro-ingénierie

La première génération d'outils de rétro-ingénierie correspondait à des outils construits de manière monolithique et "câblée" centrée sur une fonctionnalité précise. Ces outils étaient typiquement liés à un langage de programmation unique, proposaient un seul type d'analyse et un seul type de vue. C'est le cas par exemple des outils permettant à partir de programmes C d'extraire un graphe d'appels entre fonctions et d'afficher cela sous forme graphique. Ces solutions ad hoc, câblées et monofonctionnelles ne sont plus suffisantes dans le contexte

actuel. On assiste à une multiplication (1) des sources de données en entrée, mais aussi (2) des types de vues à produire en sortie. Chacun de ces deux points est étudié ci-dessous.

La problématique liée à la *multiplicité des sources de données en entrée* dépasse largement l'aspect multi-langages au sens classique, même si celui-ci a été catalogué comme l'un des points difficiles mais essentiel à résoudre en pratique [21]. Désormais il s'agit en effet de prendre en compte une très large gamme d'artefacts logiciels qui inclut par exemple code source, bytecode, binaires, bibliothèques, traces, fichiers de configurations, rapports de bug, etc.

Les environnements de rétro-ingénierie doivent pouvoir extraire des informations provenant de sources d'informations très variées.

Prendre en compte la *multiplicité des vues à produire en sortie* est également un défi pour les outils de rétro-ingénierie. Cet aspect a été identifié plus clairement dans les travaux fondateurs comme le montre la partie droite de la Figure 2. Par contre au cours des 15 années qui ont suivi, la compréhension de ce qu'est un logiciel a grandement évolué. Aujourd'hui le standard IEEE 1471 sur l'architecture des logiciels [22] définit clairement que les différents *intervenants* (stakeholder) d'un projet logiciel devraient pouvoir considérer le logiciel selon différents *points de vues* (view points) en fonction de leur métier et de leurs compétences. Il s'agit là clairement de l'application du principe de *séparations des préoccupations*, qui est à la base aussi de l'IDM. En fait les bases conceptuelles du standard IEEE 1471, de l'IDM et de la rétro-ingénierie sont très proches. Dans le standard IEEE 1471 le terme *vue* correspond à la notion de *modèle*, alors que le terme *point de vue* correspond à la notion de métamodèle [23].

Les environnements de rétro-ingénierie doivent pouvoir générer un ensemble de vues variées et adaptées à chaque type d'intervenant.

2.5. Défi n° 2 : rendre agiles les processus de rétro-ingénierie

La plupart des outils classiques de rétro-ingénierie disponibles étant "câblés", ils n'offrent que très peu de souplesse. Sachant qu'ils ne couvrent généralement que certaines fonctionnalités, l'une des difficultés récurrentes des projets de ré ingénierie consiste à trouver une manière de couvrir tous les besoins du projet en intégrant autant que faire se peut les outils dont on dispose. Ainsi l'interopérabilité des outils de rétro-ingénierie est devenue un thème de recherche important. L'une des approches les plus courantes consiste à faire communiquer les différents outils via l'import et l'export de données sous la forme de fichiers écrits dans des standards adaptés à la représentation de graphes, tel GXL. Cette approche offre bien peu de souplesse. Non seulement il est nécessaire d'écrire des "wrappers" permettant de faire la correspondance entre les structures internes manipulées par les outils et les formats externes, mais en plus ces solutions sont basées sur l'échange permanent de gros volumes de données entre outils (typiquement via des fichiers XML). Une telle solution ne permet pas non plus l'intégration fine des outils dans les environnements de programmation existants. L'ajout de nouvelles fonctionnalités est également difficile car il est nécessaire de posséder de nombreuses expertises techniques : analyse syntaxique, transformations vers des formats d'échanges, interface homme-machine, visualisation du logiciel, etc. Toutes ces difficultés résultent dans des processus de rétro-ingénierie lourds et peu flexibles car entièrement contraints par les outils existants. Développer de nouveaux outils pour prendre en

compte des technologies obsolètes ou propriétaires a un coût souvent considéré comme rédhibitoire. En fait, de part la nature exploratoire des processus de rétro-ingénierie, l'un des défis qu'il est nécessaire de surmonter est de minimiser ces coûts de développement de sorte que l'on puisse adapter les outils "à la demande" au cours du déroulement de chaque projet de rétro-ingénierie.

L'un des défis à surmonter est de pouvoir passer de processus de rétro-ingénierie rigides et figés à des processus de *rétro-ingénierie agile*, c'est-à-dire dans laquelle la flexibilité est essentielle et où les outils sont développés à la demande au fur et à mesure du processus.

2.6. Synthèse : vers la rétro-ingénierie dirigée par les métamodèles

En résumé, les nouveaux défis auxquels sont confrontés les projets de rétro-ingénierie sont d'une part (1) de prendre en compte la multiplicité des sources de données et des vues à produire ; et d'autre part (2) de pouvoir adapter processus et outils "à la demande" et ce afin de prendre en compte la nature exploratoire de la rétro-ingénierie.

Les concepts identifiés dans les travaux fondateurs de la discipline ne se sont hélas pas avérés suffisants pour surmonter ces défis : globalement les outils existants sont trop rigides. Et c'est justement là que l'Ingénierie Dirigée par les Modèles intervient. Si l'on compare les concepts de l'IDM (section 2.1, modèle, métamodèle, transformation), et les concepts de la rétro-ingénierie (section 2.2, modèle et transformation), on note que le grand absent est le concept de *métamodèle*. L'article de Chikofsky et de Cross n'y fait d'ailleurs à aucun moment référence. Pourtant il est simple, a posteriori, de voir que ce concept à la base de l'IDM, permet d'apporter la flexibilité nécessaire. Par exemple, si l'on considère l'architecture des outils de rétro-ingénierie (Figure 2), on constate qu'à chaque fois qu'un modèle est présenté, le métamodèle correspondant devrait être explicité, ce qui permettrait d'utiliser des outils génériques de transformations de modèles. C'est là la leçon de l'Ingénierie Dirigée par les Modèles, qu'il s'agit d'appliquer dans le contexte de la rétro-ingénierie.

Nous pensons que le concept de métamodèle est le pilier manquant de la rétro-ingénierie, et qu'il est possible de revisiter toutes les techniques de rétro-ingénierie proposées au cours des dernières décennies à la lumière de ce concept. Nous parlerons alors de *rétro-ingénierie dirigée par les métamodèles*.

3. METHODES : Exemple —Le processus *CacOphoNy*

Dans la section précédente l'intersection entre IDM et rétro-ingénierie a été considérée au niveau *conceptuel*. Cette section considère l'aspect *méthode* et montre que la notion de métamodèle peut réellement être au cœur des processus de rétro-ingénierie. Pour illustrer cet aspect nous avons choisi de présenter brièvement la démarche *CacOphoNy* qui résume notre expérience acquise au cours des dernières années dans des contextes tels que notre collaboration avec la société Dassault Système. Le lecteur se rapportera par exemple à [14] ou [24] pour avoir plus de détails sur l'utilisation de métamodèles dans le cadre de cette

collaboration industrielle. L'objectif ici est simplement de montrer un exemple de processus de "rétro-ingénierie dirigée par les métamodèles".

Supposons que l'on doive lancer un nouveau projet de rétro-ingénierie dans une grande entreprise. Contrairement à une approche classique, la caractéristique principale de notre solution consiste à rendre explicites les métamodèles utilisés implicitement au sein de l'entreprise considérée, autrement dit à modéliser le savoir faire de l'entreprise en terme de génie logiciel. Cette démarche s'inscrit donc réellement dans le cadre de l'IDM. Notons qu'indépendamment de cette solution, le problème à traiter est répertorié sous le nom de "View Set Scenario" dans le catalogue défini par le Software Engineering Institute (SEI) [22]. Le processus *CacOphoNy* a été défini pour résoudre ce problème [21]. Entièrement basé sur la notion de métamodèle, il propose une méthode globale pour définir un environnement de rétro-ingénierie adapté aux besoins spécifiques des entreprises.

La difficulté que cherche à résoudre le processus *CacOphoNy* consiste à construire un environnement de rétro-ingénierie adapté à une situation spécifique.

On ne saurait trop insister sur l'énoncé ci-dessus et sur le fait que le point dur auquel on s'intéresse est la *construction* d'un environnement, plutôt que sur son *utilisation*. Autrement dit en terme du standard IEEE 1471, on cherche à définir des *points de vues* lors de la construction de l'environnement (alors que l'exécution de cet environnement permettra d'extraire des *vues* à partir d'un logiciel particulier). En termes d'IDM, on travaille donc naturellement au niveau *métamodèle* (M2) et l'on cherche à instrumenter ces métamodèles (ce qui permettra lors de l'exécution de l'environnement d'extraire des *modèles* d'un logiciel particulier).

Soulignons également que la démarche telle qu'elle est présentée ici est taillée pour une grande entreprise. Les concepts identifiés ici proviennent entre autre de notre expérience dans un contexte où plus de 1200 ingénieurs travaillaient en parallèle sur un logiciel de plusieurs millions de lignes de code, formés de plus de 70000 classes C++, de 8000 composants, etc. Bien évidemment le processus proposé ici peut être simplifié pour le cas de petites entreprises, mais l'approche ne change pas même si les différents concepts sont moins évidents à discerner dans le cas simple de petites structures. Qui peut le plus, peut le moins.

Par ailleurs, le processus est décrit ci-dessous de manière séquentielle pour bien montrer les différentes activités, mais bien évidemment l'idée est de suivre un processus agile dans lequel chaque étape s'enchevêtre. En particulier la construction de l'environnement et son utilisation se fait en pratique de manière complètement entremêlée ce qui assure le caractère exploratoire de la rétro-ingénierie : on adapte l'outil au fur et à mesure de son utilisation, voire au cours même de son utilisation pour les techniques interactives les plus avancées [27]. Quoi qu'il en soit pour bien différencier les concepts, le processus est décomposé en 4 phases successives découpées en sous phases. Seule l'esquisse du processus est donné ci-dessous. Rappelons que notre objectif ici est uniquement de montrer l'omniprésence de la notion de métamodèle dans toutes les phases de la rétro-ingénierie. Le lecteur se reportera à [21] pour une description de *CacOphoNy*.

3.1. Analyse du domaine (du génie logiciel).

La première étape consiste à définir quelles sont les entités et relations utilisées. Il s'agit ainsi d'établir une cartographie des concepts de génie logiciel tels qu'ils sont utilisés dans l'entreprise. Il est essentiel de comprendre que l'on ne fait pas référence à une cartographie d'une application (niveau M1), au contraire que l'on se place ici (comme par la suite) au niveau métamodèle, c'est-à-dire au niveau M2. Le résultat attendu est un ensemble de métamodèles précis établis à partir des artefacts concrets de l'entreprise. La figure 3 décompose l'analyse du domaine en trois sous étapes.

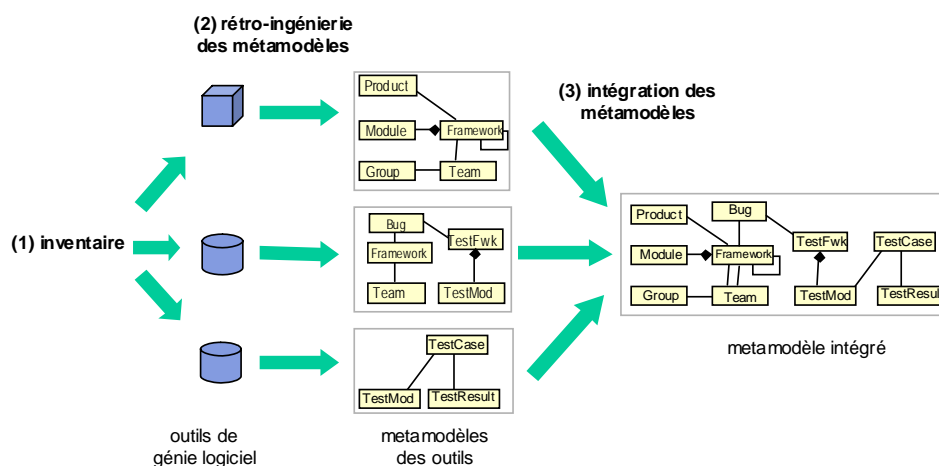


Figure 3. Analyse du domaine

A1) Inventaire des outils de génie logiciel. Il s'agit d'identifier les outils et les artefacts de génie logiciel utilisés afin d'établir une cartographie technologique de l'entreprise comme celle de la figure 3. Cela revient à identifier quelles sont les sources d'informations disponibles. On peut trouver par exemple une base de données de rapports d'incidents, des sources en Cobol et en C, des bases de tests, etc.

A2) Rétro-ingénierie des métamodèles. Il s'agit de reconstituer le métamodèle de chaque outil ou source d'information inventoriée ci-dessus. Dans l'espace technique Gammarware, Lämmel et ses collègues ont identifié cette problématique sous le terme *rétro-ingénierie des grammaires* [25]. Si l'on dispose d'une base de rapports de bugs telle que celle de Bugzilla, à partir du schéma relationnel de cette base, on pourra retrouver un métamodèle conceptuel. De même si l'on dispose d'un compilateur, d'un ensemble de fichiers XML, d'une DTD on pourra retrouver un métamodèle. Tout comme l'ingénierie des langages, la *rétro-ingénierie des langages* et la *rétro-ingénierie des métamodèles* sont des thèmes de recherche émergents.

A3) Intégration des métamodèles. Il n'est alors pas rare que les outils utilisés donnent lieu à des chevauchements entre métamodèles. La dernière étape dans l'analyse du domaine consiste à intégrer les métamodèles conceptuels des différents outils. L'intégration et la composition de métamodèles sont des thèmes de recherche importants de l'IDM qui soulèvent autant des problèmes conceptuels que techniques.

3.2. Analyse des (méta) besoins et spécifications externes.

Trop souvent la notion de métamodèle est déconnectée des besoins. *CacOphoNy* propose de relier ceux-ci au métamodèle global en utilisant la technique traditionnelle des cas d'utilisation, appliquée au niveau méta (M2) et non pas au niveau traditionnel (M1).

(B1) *Identification des (méta)acteurs*. Il s'agit des utilisateurs des langages de génie logiciel, et non pas les utilisateurs finaux de l'application développée.

(B2) *Identification des (méta) cas d'utilisation*. Il s'agit d'identifier les tâches réalisées par les différents intervenants de l'entreprise.

(B3) *spécification de l'environnement de ré-ingénierie*. Il s'agit de spécifier l'environnement à construire, et en particulier de déterminer quel sous-ensemble du métamodèle global est approprié pour réaliser chaque cas d'utilisation.

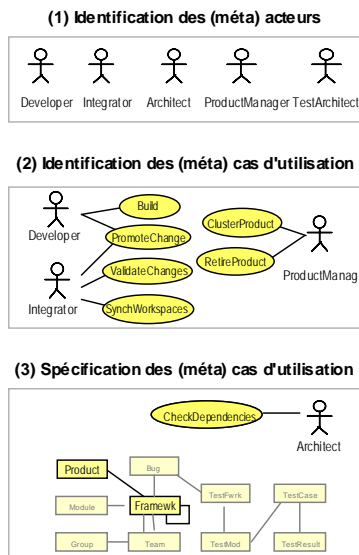


Figure 4. Meta-acteurs/Meta-besoins

3.3. Intégration et implémentation.

L'étape suivante consiste à implanter l'environnement spécifié en reproduisant globalement l'architecture présentée dans la Figure 2 (page 56). Il s'agit alors d'intégrer les briques existantes et non pas de les re-développer.

(C1) *Extraction de modèles* (partie gauche de la Figure 2). Il s'agit d'exploiter la traçabilité établie dans l'étape (A) entre la liste des outils fournis lors de l'inventaire et le métamodèle global. Le but est de trouver quel est l'ensemble des outils à instrumenter pour obtenir les modèles dont on a besoin. Des "wrappers" et/ou des transformations doivent être réalisés pour concrétiser les différentes étapes établies en (A).

(C2) *Présentation des modèles*. La concrétisation de la partie droite de la Figure 2 peut également se baser sur la réutilisation d'outils et techniques existants, mais aussi sur la paramétrisation d'outils génériques par exemple de visualisation. Des exemples seront donnés dans la section 4.

3.4. Exécution et tests.

L'objectif ultime est d'utiliser l'environnement ainsi produit pour supporter le processus d'évolution des applications logicielles développées par l'entreprise. En pratique ceci donne lieu à de nombreux retours et demandes de modification. Comme nous l'avons dit, les étapes présentées ci-dessus s'enchevêtrent en pratique, donnant lieu à des processus dans lesquels l'outil de rétro-ingénierie à construire est développé au fur et à mesure de son utilisation pour répondre aux besoins souvent découverts de manière incrémentale.

3.5. Synthèse

Comme le lecteur a pu le constater les métamodèles jouent un rôle central tout au long du processus. Les étapes (A), (B) et (C) se déroulent uniquement au niveau méta (M2) et différentes problématiques ont été identifiées : rétro-ingénierie des métamodèles, transformations de métamodèles, intégration de métamodèles, etc. La terminologie "rétro-ingénierie dirigée par les métamodèles" est donc tout à fait justifiée.

4. OUTILS : Exemple — Cartographie avec l'environnement Obeo

Après avoir étudié l'intégration IDM et rétro-ingénierie au niveau conceptuel (section 2), puis au niveau méthode (section 3), cette section se concentre sur l'aspect outil. Nous allons voir que l'utilisation de métamodèles permet d'obtenir des outils de rétro-ingénierie d'une flexibilité inégalée jusqu'alors. Nous illustrerons notre propos à partir d'un cas d'utilisation simple. Supposons que l'on cherche à établir une cartographie d'une application logicielle basée sur des technologies hétérogènes Cobol et C qui, de plus, mélange interface graphique, logique métier et accès aux données. Nous allons voir que la notion de métamodèle et les techniques d'IDM correspondantes permettent de résoudre ce problème de manière agile.

4.1. Vision globale

La Figure 5 ci-dessous présente une vision globale du système. Comme on peut le voir, l'architecture utilisée correspond bien à l'architecture classique des outils de rétro-ingénierie (Figure 1, page 55), si ce n'est que l'on peut noter l'omniprésence de métamodèles à chaque étape. On suppose ici que l'application à analyser est formée de programmes Cobol et C à gauche. Le modèle de cartographie en haut à droite montre le résultat à obtenir. On ne décrit par la suite que le chemin décrit en gras, mais bien évidemment d'autres vues peuvent être produites comme le montre la Figure 5. Notons qu'entre les deux extrêmes à droite et à gauche, les transformations utilisent un outil générique de transformation de modèles, ici Obeo Transformer. Tous les métamodèles sont détaillés dans les sections suivantes en parcourant la figure 5 de gauche à droite.

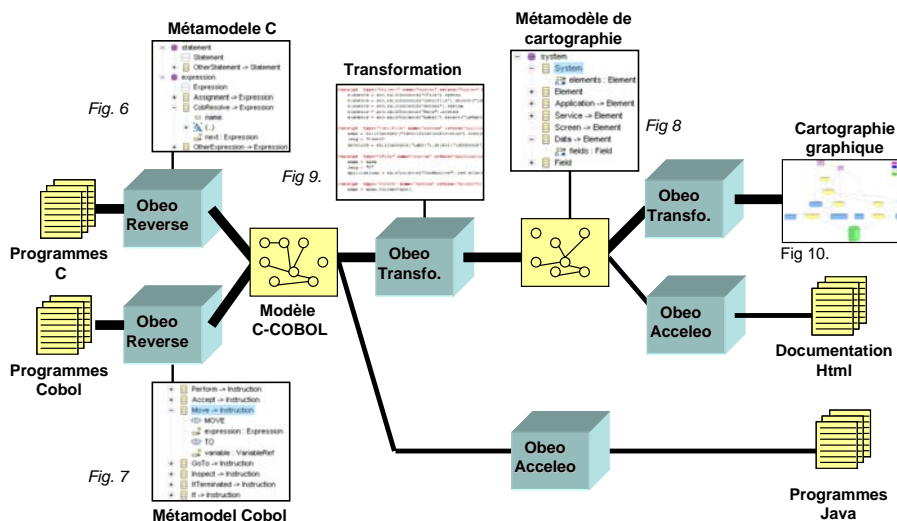


Figure 5. Une chaîne complète de rétro-ingénierie

4.2. Extraction des modèles : analyse et métamodèles de code

La mise au point d'une telle chaîne de rétro-ingénierie commence par la création d'un métamodèle de code qui est décoré avec des informations syntaxiques. La création de ces métamodèles de code grâce à Obeo Reverse permet de se concentrer sur le langage à analyser, car il fusionne les étapes d'analyse syntaxique et de métamodélisation. Un modèle, résultat de l'analyse de technologies hétérogènes, est obtenu à partir : (1) d'un analyseur de surface pour le langage C afin de ne récupérer que les informations macroscopiques de détection des appels aux programmes cobol, autrement dit les appels à une méthode système appelée `cob_resolve` et, (2) d'un analyseur fin pour Cobol détectant précisément tous les aspects du langage nécessaire au cas d'étude considéré.

La figure 6 ci-dessous à gauche montre un extrait du métamodèle de C. L'analyseur de surface C est simple car les informations que l'on veut retirer des fichiers C sont simples. On distingue 2 types d'expression : les expressions « CobResolve » et les autres. Ainsi, le modèle résultant de l'analyse de fichiers C sera facile à exploiter. Le code C ne contient pas de logique métier, ni d'interface graphique. Toute cette connaissance se situe dans le langage Cobol, ce qui explique que l'analyse du code Cobol doit être plus fine. On va chercher à exploiter chaque instruction des sources Cobol. L'extrait du métamodèle pour Cobol présenté dans la figure 7 à droite montre un certain nombre d'instructions du langage et la manière de décorer les éléments du métamodèle avec des informations syntaxiques. On retrouve notamment la structure d'une instruction MOVE, permettant d'affecter une expression à une variable : `MOVE <expression> TO <variable>`. Les décorateurs syntaxiques MOVE et TO sont présents en bleu dans le métamodèle et délimitent les références vers d'autres éléments eux aussi décorés. Cette démarche rappelle l'utilisation de sous règles dans les grammaires classiques.

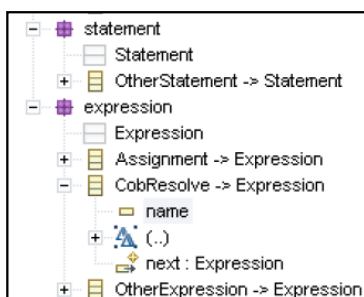


Figure 6. Métamodèle de surface pour le C

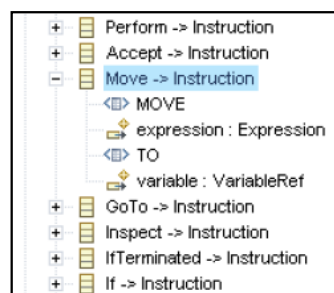


Figure 7. Métamodèle de COBOL

4.2. Transformations et métamodèle de cartographie

Les modèles produits par les analyseurs sont proches des technologies sources. Il reste donc à en extraire les informations pertinentes (diagrammes d'architecture, modèles de haut niveau, règles métiers, ...). Cela passe par la définition d'un métamodèle de cartographie. Le métamodèle ci-contre permet de décrire les concepts à représenter, et les dépendances qu'il y a entre eux. Il précise qu'un système est représenté par des éléments d'architecture

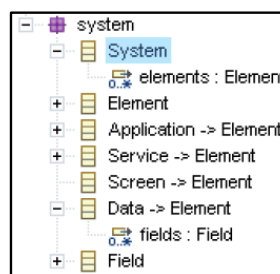


Figure 8. Métamodèle de cartographie

5. Conclusion

En fait il n'existe que peu de différences conceptuelles entre la rétro-ingénierie et la modélisation. Dans un cas on part d'un logiciel existant formé d'artefacts concrets, dans l'autre cas on part de besoins exprimés informellement. Historiquement, le génie logiciel s'est tout d'abord concentré sur la modélisation, ce qui est bien naturel en l'absence d'existant. Mais dans un monde où le logiciel est omni-présent et où la part de l'existant est de plus en plus importante, la tendance devrait peu à peu s'inverser pour arriver à une situation où modélisation et rétro-ingénierie ne seront plus opposées, mais vu au contraire comme deux techniques complémentaires au sein d'un phénomène d'évolution continu.

6. Remerciements

Nous tenons à remercier Stéphane LACRAMPE, Etienne JULIOT, Cédric BRUN de la société Obeo pour leur participation à ce travail.

7. Bibliographie

- [1] J. Bézivin, On the Unification Power of Models, Software and System Modeling Journal, 4(2):171--188, 2005
- [2] J.M. Favre, J. Estublier, M. Blay, éditeurs, *L'Ingénierie Dirigée par les Modèles : au-delà du MDA*, Hermès - Lavoisier, ISBN 2-7462-1213-7, 240 pages, 2006
- [3] PlanetMDE, the community web portal for Model Driven Engineering, <http://planetmde.org>
- [4] IEEE Computer, Numéro spécial dédié à l'IDM, Février 2006
- [5] E. Chikofsky, J. Cross, *Reverse Engineering and Design Recovery: A Taxonomy*. IEEE Software, Jan. 1990
- [6] R. Arnold, *Software Reengineering*, IEEE, ISBN 0-8186-3272-0, 1993
- [7] R. Arnold, *On Software Restructuring*, IEEE, ISBN 0-8186-0680-0, 1986
- [8] D.L. Parnas. *Software Aging*. ICSE 1994
- [9] T. Mens and al. *Challenges in Software Evolution*. IWPSE 2005
- [10] S. Demeyer, S. Ducasse, O. Nierstrasz, *Object-Oriented Reengineering Patterns*, Morgan Kaufmann/Elsevier, ISBN 1-55860-639-4, 2003
- [11] R. Lammël, *Stop Bashing Cobol!*, Cobol Research Laboratory at the Free University of Amsterdam, <http://www.cs.vu.nl/~ralf/Cobol>
- [12] J.M. Favre. *Languages Evolve Too! Changing the Software Time Scales*. IWPSE 2005
- [13] L. Racoon. *Fifty Years of Progress in Software Engineering*. Software Engineering Notes, Vol. 22, n. 51, Jan. 1997
- [14] J.M. Favre and al. *Reverse Engineering a Large Component-based Software Product*. CSMR 2001
- [15] R. Marvie, L. Duchien, M. Blay, *Les Plateformes d'exécution et l'IDM*, Chapitre 4, [2]
- [16] J.M. Favre, J. Bézivin, I. Bull, "Evolution, rétro-ingénierie et l'IDM : du code aux modèles", Chapitre 8, [2]
- [17] Obeo, *Acceleo, une solution 100% open source pour l'IDM*, <http://www.acceleo.org>

- [18] J. Bézivin, O. Gerbé, *Towards a Precise Definition of the OMG/MDA Framework*, ASE 2001
- [19] J.M. Favre, *Towards a Basic Theory to Model Model Driven Engineering*, WiSME 2004, papier disponible à partir de <http://www-adele.imag.fr/~jmfavre>
- [20] J.M. Favre, *GSEE : A Generic Software Engineering Environment*, IWPC 2001
- [21] L. O'Brien, C. Stoermer, C. Verhoef, *Software Architecture Reconstruction: Practice Needs and Current Approaches*, SEI Technical Report CMU/SEI-2002-TR-024, 2002
- [22] *IEEE Recommended Practice for Architectural Description of Software-Intensive Systems*, IEEE Std 1471-2000
- [23] J.M. Favre, *CacOphoNy: Metamodel-Driven Architecture Reconstruction*, WCRE 2004
- [24] J. Estublier, J.M. Favre, R. Sanlaville, *An Industrial Experience with Dassault Systèmes' Component Model*, Book chapter in *Building Reliable Component-Based Systems*, I. Crnkovic, M. Larsson editors, Archtech House publishers, ISBN 1-58053-327-2, 2002, pp. 375-386
- [25] P. Klint, R. Lämmel, C. Verhoef. *Toward an engineering discipline for grammarware*, ACM Transactions on Software Engineering and Methodology (TOSEM), Vol. 14, N. 3, Juillet 2005
- [26] T.R. Dean and al., *Agile Parsing in TXL*, Journal of Automated Software Engineering 10,4 (October 2003), pp. 311-336.
- [27] J.M. Favre, *A New Approach to Software Exploration : Back-packing with G^{SEE}*, CSMR 2002
- [28] L. Moonen, *Generating Robust Parsers using Island Grammars*, WCRE 2001
- [29] J. Bézivin, N. Ploquin. *Tooling the MDA framework: a new software maintenance and evolution scheme proposal*. Journal of Object-Oriented Programming, JOOP, 2001
- [30] J. Bézivin, H. Brunelière, F. Jouault, I. Kurtev. *Model Engineering Support for Tool Interoperability*, WiSME 2005, <http://planetmde.org/wisme-2005>, Oct. 2005
- [31] I. Kurtev, J. Bezivin, and M. Aksit, *Technological spaces: an initial appraisal*, In CoopIS, DOA'2002 Federated Conferences, Industrial track, Irvine, 2002
- [32] F. DeRemer, H. Kron. *Programming-in-the-Large vs. Programming-in-the-Small*. IEEE Transactions on Software Engineering, Vol. 2, N. 2, Fev. 1976, pp. 80-86.
- [33] I. Bull, J.M. Favre, *Visualization in the Context of Model Driven Engineering*, International Workshop on Model Driven Development of Advanced User Interfaces, MDDAUI @ MoDELS 2005, Montego Bay, Jamaica, Octobre 2005
- [34] J.M. Favre, *Maintenance et Ré-ingénierie globale des logiciels*, Thèse de doctorat, Université de Grenoble, 1996
- [35] A. Mendelzon, J. Sametinger. *Reverse Engineering by Querying and Visualization*. in Software - Concepts and Tools, Vol. 16, N. 4, 1995, pp. 170-182
- [36] R. Kazman, S. J. Carrière, *View Extraction and View Fusion in Architectural Understanding*, International Conference on Software Reuse, ICSR 1998
- [37] S. Ducasse, M. Lanza, S. Tichelaar, *Moose: an Extensible Language-Independent Environment for Reengineering Object-Oriented Systems*, 2nd International Workshop on Constructing Software Engineering Tools, CoSET 2002
- [38] M. Lanza, *CodeCrawler: Lessons Learned in Building a Software Visualization Tool*, 2003