Correspondance structurelle entre produits et procédés : un pattern classique, analysé avec des méta-modèles explicites.

Jean Bézivin[(1)], Jean-Paul Bouchet[(2)] & Erwan Breton[(1,2)]

[(1)] LRSG, Université de Nantes♣
Faculté des Sciences et Techniques
2, rue de la Houssinière
BP 92208
44322 Nantes cedex 3
Jean.Bezivin@sciences.univ-nantes.fr

[(2)] Soft-Maint
4, rue du Château de l'Eraudière
BP 588
44074 Nantes cedex 3
jpbouchet@sodifrance.fr
ebreton@sodifrance.fr

Résumé :

Le pattern P&P (pour produit et procédé ou product and process) traite des relations structurelles entre les espaces de produits et les espaces de procédés. Ces relations ont souvent été mises en évidence par le passé, par exemple dans la méthode SADT ou dans les travaux de N. Wirth. Elles n'ont toutefois jamais été réellement formalisées. Cela est principalement dû au manque d'outils permettant de mettre en relation des modèles de produits et des modèles de procédés. Aujourd'hui, avec les nouveautés apparues dans le domaine de la méta-modélisation telles que l'architecture quatre couches, le MOF (Meta-Object Facility) et les méta-modèles explicites, il est possible d'établir des liens formels entre les entités dynamiques d'un modèle d'activité et les entités statiques d'un modèle de données. C'est dans ce cadre que se place le pattern P&P dont le but est de fournir un ensemble de mécanismes facilitant la co-conception des modèles de produits et des modèles de procédés. Ce travail en est actuellement à sa phase initiale, c'est-à-dire l'analyse d'exemples où l'articulation produit/procédé joue un rôle majeur, et la détermination de correspondances structurelles qu'il serait possible d'expliciter en termes de relations entre les méta-modèles. Il sera suivi d'une formalisation des correspondances structurelles et appliqué à la définition d'une stratégie globale de migration de systèmes d'informatique d'entreprise. Seul le travail initial d'investigation est présenté dans cet article.

Mots-clefs :

Patterns ; Méta-modèles ; Meta-Object Facility.

♣ Ce travail a été partiellement supporté par le projet RNRT Pilote.

1 Introduction.

L'existence d'un lien entre les données et les algorithmes est connue depuis longtemps. C'est ce lien, cette correspondance structurelle, que nous allons tenter d'expliciter et de généraliser en une relation entre produits et procédés. Plus précisément nous nous attacherons surtout à étudier cette relation au niveau des méta-modèles. Actuellement, la plupart de ces méta-modèles donnent une vue asymétrique de la réalité, selon qu'ils penchent du côté produit, comme UML (Unified Modeling Language), ou du côté procédé, comme les systèmes de workflow. Dans ce papier nous discutons donc de la relation entre méta-modèle de produit et méta-modèle de procédé. Nous commençons par présenter l'état de l'art dans le domaine de la modélisation, en particulier en ce qui concerne la représentation explicite des méta-modèles, ce qui nous offre la possibilité nouvelle de baser notre étude sur des exemples plus nombreux et plus précis. Nous décrivons ensuite le pattern P&P [1]. Nous en donnons enfin quelques exemples d'applications réalisées avant l'émergence des méta-modèles explicites. Et enfin nous présentons quelques applications plus récentes du pattern P&P avant de conclure sur l'intérêt d'utiliser des méta-modèles explicites et sur les retombées pratiques que l'on peut en attendre.

2 Les techniques industrielles de méta-modélisation.

La nature du processus de développement logiciel a énormément évolué au cours de la dernière décade. Ces changements en annoncent probablement d'autres, encore plus importants. Il apparaît de plus en plus clairement aujourd'hui que le passage, dans les années 80, de la technologie procédurale à la technologie des objets ne nous a pas mené à un état de stabilité. Bien au contraire, le remplacement des procédures par les objets a surtout permis d'ouvrir de nouvelles voies de développement, tout en accélérant les évolutions technologiques. Quelques aspects importants de cette transition méritent d'être notés.

Le pouvoir unificateur des objets, fréquemment invoqué comme l'un des atouts essentiels de cette technologie peut apparaître aujourd'hui comme une qualité surfaite, voire même une publicité un peu mensongère. On sait maintenant qu'il n'y a pas une, mais plusieurs sémantiques pour les objets et qu'elles sont loin d'être toutes similaires. Le concept de classe ou la relation d'héritage recouvrent par exemple souvent des réalités bien éloignées dans leurs différents contextes d'utilisation. S'il pouvait être légitime de mettre l'accent sur les similarités de ces sémantiques dans les années 80, il est aujourd'hui nécessaire de prendre conscience de leur diversité et de se donner les moyens de la maîtriser.

Le problème du fossé sémantique entre les différents modèles à objets s'est d'abord posé en termes d'interopérabilité de code exécutable. L'OMG a du réaliser des investissements importants avant de mettre au point la solution basée sur CORBA et

IDL. D'autres ont esquivé le problème en se définissant un modèle pivot de référence de nature propriétaire (Microsoft avec COM, Sun avec Java). Sans vouloir porter un jugement de valeur sur ces trois cultures technologiques (Microsoft/COM, Sun/Java, OMG/Corba), il est cependant assez clair que la solution la plus générale sur le plan conceptuel est celle de l'OMG.

Mais le problème s'est encore aggravé quand on est passé des objets purement exécutables aux objets de développement. L'arrivée des composants, objets à double vie, l'une au développement et l'autre à l'exécution, a augmenté d'un ordre de magnitude la complexité du problème. Les modèles de composants comme les EJB ou le nouveau modèle Corba des composants vont devenir incontournables en développement standard de logiciel. Les propriétés de ces composants sont plus nombreuses, plus complexes et annoncent d'autres extensions.

Cette section résume quelques aspects importants de la transition vers les techniques de modélisation basées sur les méta-modèles explicites.

2.1 L'architecture à quatre niveaux

Le paysage de la modélisation logicielle a radicalement évolué au cours de ces dernières années. Ces changements majeurs sont principalement à porter au crédit de l'OMG (Object Management Group), et de ses trois recommandations : UML, MOF [15] (Meta Object Facility) et XMI (XML Metadata Interchange). Cette émergence n'est pas spontanée mais fait suite aux travaux menés par des communautés telles que IRDS ou encore CDIF [2] durant la dernière décennie. On assiste donc aujourd'hui à un consensus autour d'une architecture à quatre niveaux :

- le niveau M3 constitué par le méta-méta-modèle,
- le niveau M2 constitué par les méta-modèles,
- le niveau M1 constitué par les modèles,
- et enfin le niveau M0, qui se place au niveau des instances terminales.

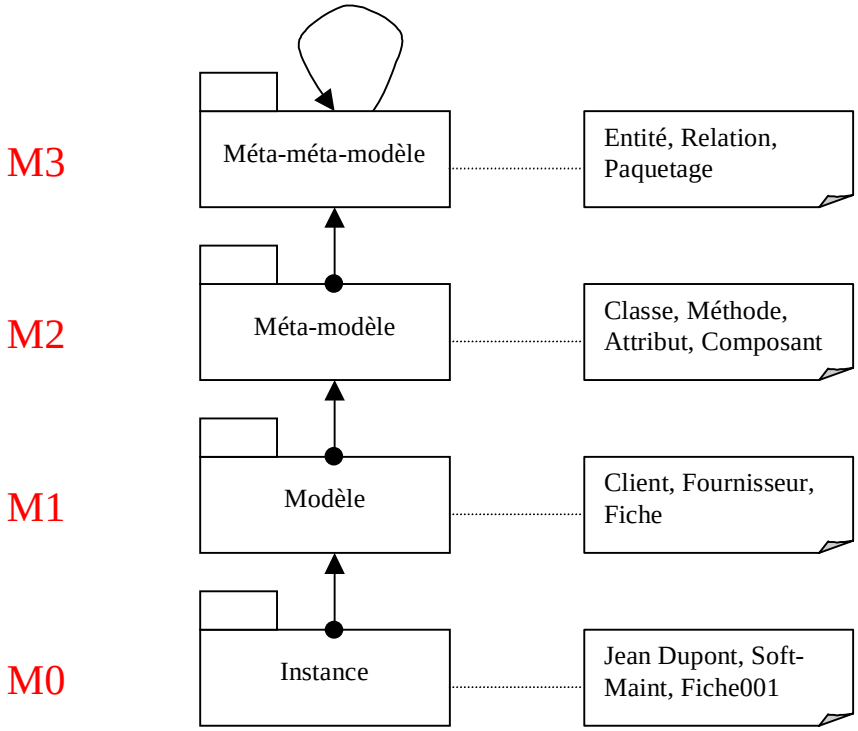

Figure 1 : L'architecture quatre couches

Le méta-méta-modèle va donc contenir des méta-entités. Le méta-modèle va contenir des entités et des relations. Le modèle, lui, va contenir des classes, des méthodes, etc. Enfin, l'instance, elle, va s'exprimer en terme de client, fournisseur, numéro d'identification, etc. Chaque couche va donc se définir en fonction des entités définies à la couche supérieure, à part le méta-méta-modèle qui se définit lui-même. On peut situer le modèle au niveau de l'application, le méta-modèle au niveau du domaine (systèmes logiciels pour UML, Workflow pour jFlow, …). Le méta-méta-modèle peut donc être considéré comme une sorte de bus de connaissance universel. En ce sens on peut comparer le MOF à CORBA, le MOF permettant l'interopérabilité au niveau des connaissances, CORBA situant l'interopérabilité au niveau du code. Toutes les entités définies dans les méta-modèles étant exprimées en fonction des méta-entités définies dans le MOF, il va être possible de les comparer et de les mettre en correspondance.

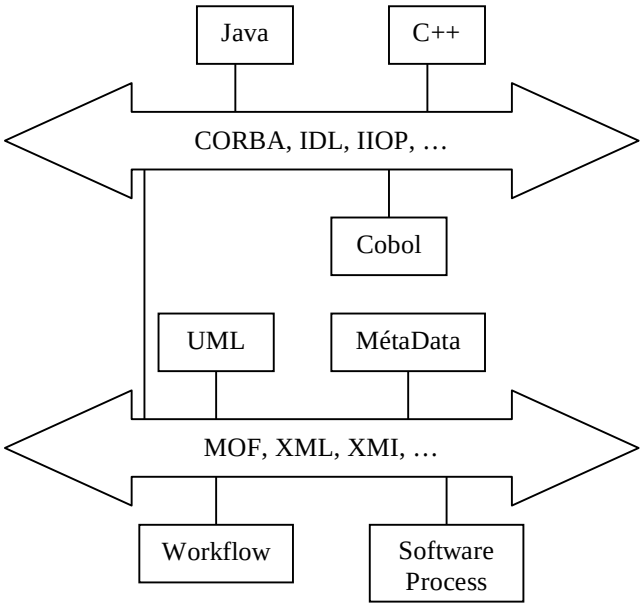

Figure 2 : Les deux bus d'interopérabilité de l'OMG

2.2 Un mouvement d'unification qui se généralise

La proposition de l'OMG consiste donc principalement en trois propositions. Le MOF est le méta-méta-modèle standard unique. XMI est le format qui va permettre l'échange de modèles et de méta-modèles. Il est basé sur XML. Enfin UML est l'un des premiers méta-modèles basé sur le MOF adopté par l'OMG. D'ailleurs le MOF a été extrait du noyau d'UML et reste en correspondance avec celui-ci. On peut noter que Microsoft suit la même dynamique avec des propositions concurrentes : OIM (Open Information Model), RTIM (Repository Type Information Model) et XIF (XML Interchange Format). OIM est le modèle d'architecture à quatre couches. RTIM est le méta-méta-modèle, correspondant au MOF de l'OMG. Et enfin XIF est le format d'échange, alter-ego de XMI. On assiste donc à une véritable standardisation dans l'architecture de définition des méta-modèles. Les industriels des différents domaines ne sont pas en reste puisqu'ils se regroupent de plus en plus, que ce soit en consortium ou en groupe de travail, afin de mettre au point des méta-modèles minimaux par domaines. UML va donc être rejoint par un certain nombre d'autres méta-modèles concernant des domaines tels que le workflow [19], le data warehouse, ou encore le software process [16]. Le MDC (Meta-Data Coalition) construit un méta-modèle pour les règles métier (business rules). On va donc avoir une architecture organisée autour d'un méta-méta-modèle unique, et comprenant au niveau M2 un certain nombre de méta-modèles. A l'intérieur de cette couche, on va donc pouvoir distinguer deux groupes, les méta-modèles de produits définissant des données statiques (classe, attribut, relation,…) telles qu'UML ou encore Merise ou MCX, et les méta-modèles de procédés définissant des données dynamiques (activité,

tâche, …) tels que les méta-modèles de worklow ou ceux de software process. Notre but est donc d'étudier les relations existantes entre ces deux groupes.

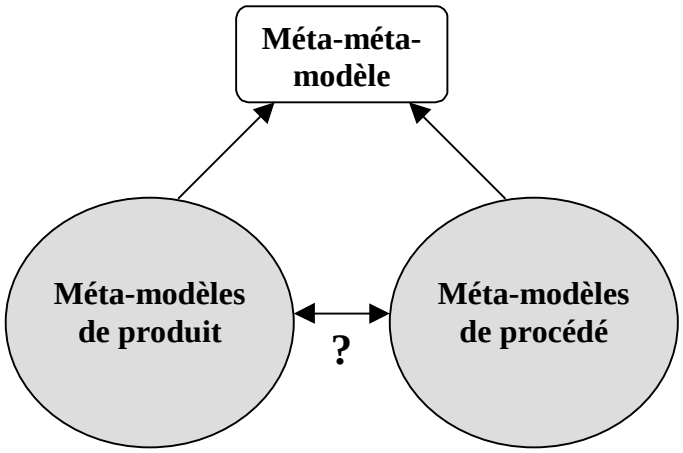

Figure 3 : Pattern P&P et méta-modèles explicites

3 Le pattern P&P.

3.1 Les fondements

La relation entre produits et procédés est un thème récurrent en informatique. De nombreux travaux ont suggéré l'existence d'une forte correspondance structurelle entre les modèles de produit et ceux de procédés, et plusieurs auteurs ont apporté des arguments empiriques en faveur de cette relation. Ainsi, alors que la plupart des gens travaillaient encore sur la programmation structurée, N. Wirth [20] notait déjà l'importance de cette relation. Cette relation prend aujourd'hui de plus en plus d'importance. La qualité des logiciels est devenu un enjeu majeur et le contrôle de cette qualité s'est déplacé du produit final au procédé de production. Les conditions dictées par le marché aux entreprises étant en perpétuelle évolution, celles-ci doivent donc adapter continuellement leurs produits et leurs procédés pour y répondre. Il en va de même avec les changements importants qui se sont produits au niveau technologique (programmation objet, par composant, Internet, …). Pour toutes ces raisons, et la liste est loin d'être exhaustive, les entreprises vont devoir se concentrer autant sur ce qu'elles manipulent et produisent (les informations et la technologie en support) que sur la façon dont elles le produisent ou le consomment (les procédés). Les interactions entre ces deux espaces (statique et dynamique) doivent être clairement définies, afin de pouvoir cerner immédiatement l'impact que peut avoir une modification dans l'une ou l'autre des dimensions. C'est à ce niveau que le pattern P&P va+ trouver son utilité.

3.2 Les promesses

Le pattern P&P traite donc de l'explicitation des relations entre produit et procédé. Selon nous, dès la phase de conception le produit et son procédé de fabrication vont évoluer de façon plus ou moins parallèle. Ainsi, selon la nature du produit on va plus ou moins insister sur certaines phases du procédé. On va donc assister à une co-conception et à une co-évolution des modèles produits et procédés, qui vont s'enrichir mutuellement au cours du temps. La découverte d'une nouvelle contrainte pour le produit pourra amener des modifications pour le procédé. L'introduction de nouvelles méthodes de développement pourront introduire des modifications sur le produit final. Mais le pattern P&P ne s'arrête pas au stade de la conception. Tout au long de son cycle de vie différents procédés vont s'appliquer sur le produit, que ce soit la conception, l'implémentation, le test, la mise en place, l'utilisation ou la maintenance. Chacun de ces procédés va considérer le produit sous un certain point de vue. Certaines de ces phases vont directement s'attaquer au produit, que ce soit en création lors des phases de conception, d'implémentation, ou en modification lors de la maintenance. D'autres auront un regard plus extérieur, plus utilisateur. C'est le cas des phases de test et d'utilisation.

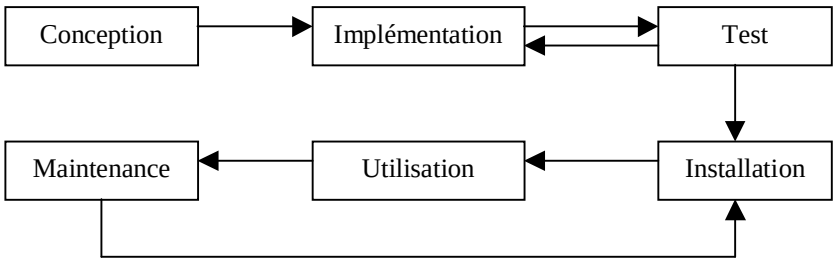

Figure 4 : Cycle de vie simplifié d'un produit

Toutes ces phases vont être en étroite relation avec le produit. Nous nous proposons donc d'étudier les répercussions que pourrait entraîner une modification d'un espace dans l'autre. Pour cela nous comptons établir un certain nombre de règles de cohérence destinées à assurer l'adéquation entre un produit et ses procédés, ainsi que des patterns permettant un report fluide des modifications d'une dimension vers l'autre. Ce travail de recherche et d'analyse porte sur deux axes principaux. D'une part, afin d'étayer nos travaux d'exemples concrets, nous cherchons des applications où nous pensons déceler l'importance de cette articulation produit/procédé. Quelques-uns de ces exemples seront présentés dans les chapitres suivants. D'autre part nous étudions les apports que pourraient constituer les nouvelles techniques de méta-modélisation (voir le chapitre précédent).

4 Quelques exemples d'applications.

À des fins illustratives, nous donnons dans cette section, une courte présentation de plusieurs situations typiques où le pattern P&P peut se manifester.

4.1 Le formalisme SADT/IDEF0

La méthode SADT (Structured Analysis and Design Technique) a été proposée par D.T. Ross à la société Softech en 1976-77. Cette méthode préconisait de décomposer le problème selon deux dimensions : la dimension activités et la dimension données. Les diagrammes d'activité étaient représentés par des actigrammes et les diagrammes de données par des datagrammes. Loin d'être indépendants, ces deux aspects devaient en permanence être confrontés afin de rester cohérents. En effet, la méthode originale recommandait de suivre l'une ou l'autre des voies et d'alterner l'analyse en changeant de voie et en vérifiant la similarité des démarches.

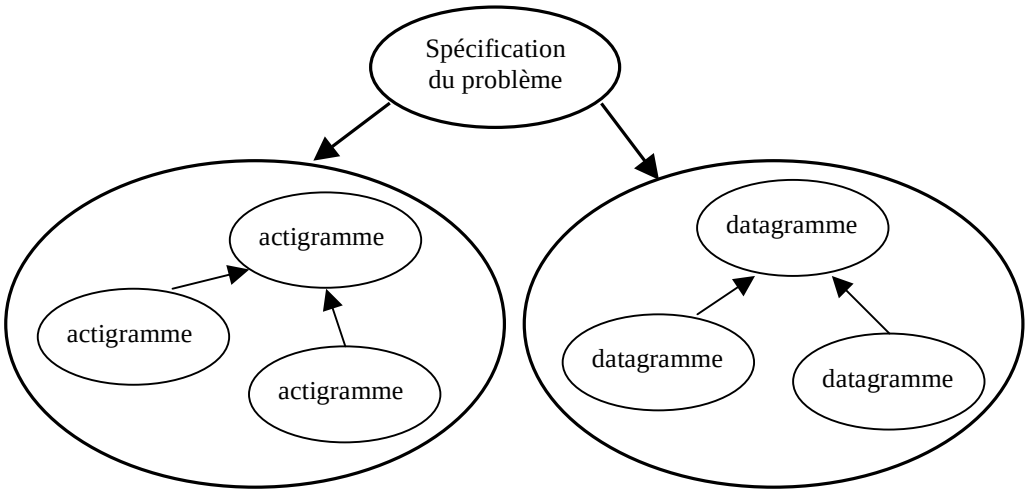

Figure 5 : La décomposition du problème entre actigrammes et datagrammes

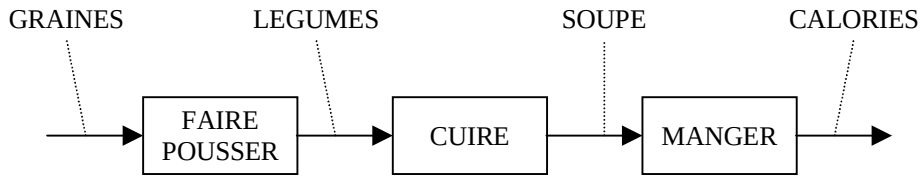

Figure 6 : Un actigramme [11]

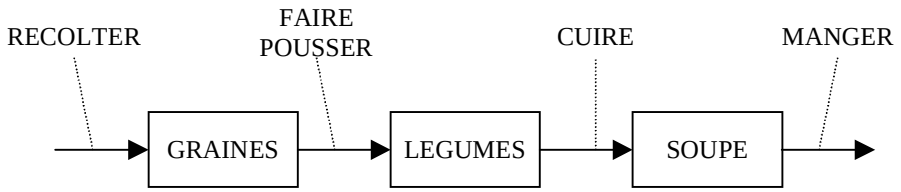

Figure 7 : Un datagramme [11]

Dans [11], M. Lissandre notait: "*Lors de la modélisation SADT, les aspects "données" et "activités" sont toujours examinés ensemble. Cependant, à un instant donné, l'accent est porté prioritairement sur l'un des deux aspects. SADT demande en fait la création de deux décompositions : la décomposition des activités et la décomposition des données. La première … met l'accent sur les fonctions, qui sont représentées par des boîtes, et montre les objets ou les données qu'elles manipulent, représentées par des flèches. La seconde met l'accent sur les données qui sont, cette fois, représentées par des boîtes, tout en montrant, sous forme de flèches, les activités qui les utilisent ou les créent. … L'étape finale du processus de modélisation consiste alors à vérifier la cohérence du modèle d'actigrammes et du modèle de datagrammes, c'est à dire de s'assurer que tous les éléments (boîtes et flèches) figurant dans une décomposition se retrouvent de manière cohérente (mais sous l'autre représentation), dans la décomposition duale. … Plus compliqué qu'il ne paraît au premier abord, dans la mesure où il n'y a pas de correspondance biunivoque entre les boîtes d'un côté et les flèches de l'autre, ce processus consiste essentiellement à établir ces correspondances (appelées lien activités/données ou lien données/activités selon le cas) et à effectuer un contrôle croisé des deux liens dans le but de rechercher d'éventuelles erreurs ou omissions dans les deux décompositions.*"

4.2 Les méthodes JSP/JSD

La méthode JSD est une méthode de conception de systèmes qui possède de nombreux aspects originaux. Elle a suivi la méthode JSP de programmation structurée. La méthode JSP propose une façon de modéliser le flot d'entrée et le flot

de sortie de données d'un programme en utilisant des arbres de structure (séquences de données, répétition, hiérarchie, alternatives, etc.). A partir de là, M. Jackson montre comment on peut se servir de ces informations pour dériver la structure de contrôle du programme (opérations et processus). Considérons par exemple un fichier qui débute par un enregistrement de type T1, suivi d'un nombre quelconque d'enregistrements de type T2 et se terminant par un seul enregistrement de type T3. Dans [8], cette analyse s'exprime sur les données par un schéma identique à celui décrit en Figure 8. On déduit de ceci la structure de contrôle abstraite suivante:

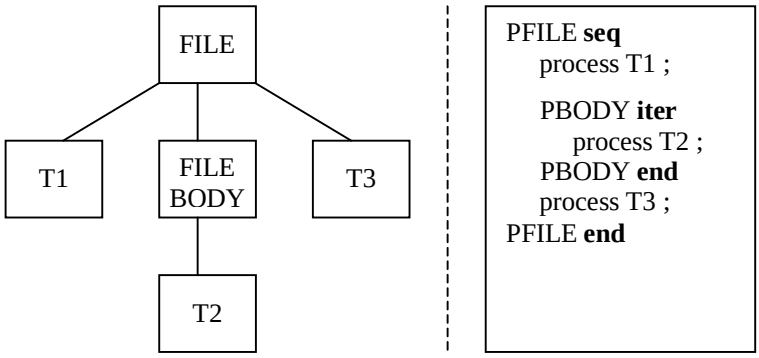

Figure 8 : Un arbre de Jackson et son code associé

De nombreuses idées des méthodes JSP/JSD ont également été reprises dans la méthode SSADM [18]. Dans les années 80, des travaux similaires avaient également été réalisés en France sur la méthodologie de programmation (école de Nancy) .

4.3 La construction algorithmes+données de N. Wirth

N. Wirth a remarqué la similarité de structure entre l'organisation des données et l'organisation du contrôle. Cette similarité de structure apparaît plus clairement lorsque certaines notations sont utilisées, comme le langage Pascal par exemple :

- à un vecteur correspondra une boucle *pour*,
- à un tableau à deux dimensions correspondra une boucle *pour* imbriquée,
- à un enregistrement avec variante correspondra une instruction *case*,
- etc.

```
var unTableau: array [0..7] of boolean;
```

```
for i := 0 to 7 do
        unTableau[i] := i % 2 = 0;
```

```
type TPersonne = record
          nom : string;
          suivant : ^TPersonne;
        end;
var uneListe : ^TPersonne;
```

```
while (not isNull(uneListe)) do begin
        uneListe^.nom := " ";
        uneListe := uneListe^.suivant;
end;
```

```
type TPersonne = record
          case sexe : char of
          ' m ' : (degageOM : boolean);
          ' f ' : (nomJF : string);
        end;
var unePersonne : TPersonne;
```

```
case unePersonne.sexe of
        ' m ' : degageOM := true;
        ' f ' : nomJF := " ";
end;
```

Figure 9 : Quelques exemples de correspondances donnée/algorithme

N. Wirth suggère d'utiliser cette correspondance pour la construction de programmes constitués de données et d'algorithme. Dans [20] par exemple, alors que l'on s'intéressait surtout à la Programmation Structurée, il remarquait que : *"… structuring considerations of program and data are often closely related. Hence, it is only natural to subject also the specification of data to a process of stepwise refinement. Moreover, this process is naturally carried out simultaneously with the refinement of the program".*

4.4 La méthode du "Chief Programmer Team" de Harlan Mills

H. D. Mills a proposé chez IBM, dans les années 70, une méthode connue sous le nom de "Chief Programmer Team" (CPT). Dans le contexte de la prise de conscience des problèmes de génie logiciel, la proposition de Mills constituait une réponse de type "organisation" alors qu'à la même époque E.W. Dijkstra proposait une réponse de type "langage" avec la programmation structurée.

Le modèle de Mills est similaire à celui du chirurgien entouré de son équipe qui préparent son travail, l'assistent pendant l'intervention et s'occupent de toutes les tâches annexes. En contexte CPT, le chef-programmeur est assisté au minimum d'un programmeur confirmé et d'un assistant documentaliste. D'autres intervenants peuvent compléter cette équipe minimale d'intervention.

La méthode CPT a été utilisée et évaluée initialement pour la construction du système d'information du "New York Times". Elle a ensuite été adaptée et généralisée, principalement dans l'environnement IBM. Elle n'a d'intérêt aujourd'hui qu'à titre historique. Nous la mentionnons ici car elle a permis de constater que "*la structure d'un système logiciel est partiellement déterminée par la structure de l'organisation qui a contribué à produire ce système logiciel*". Nous avons affaire

ici encore à l'affirmation d'une relation entre la structure du produit et celle du processus, même s'il s'agit d'une forme particulière du processus, à savoir l'organisation du réseau des acteurs de production.

4.5 L'atelier Semantor de Soft-Maint

L'une des originalités de l'atelier de rétro-ingénierie *Semantor* de Soft-Maint [3] est d'extraire la totalité de la "sémantique" d'un programme Cobol à l'aide d'un méta-modèle composite. Ce méta-modèle contient les éléments de structuration de données aussi bien que les éléments de structuration de contrôle. Le modèle dérivé est donc composé des deux types d'entités. En observant le méta-modèle d'extraction, on pourrait séparer la partie contrôle de la partie données. Il s'agit donc ici d'une démarche analytique correspondant aux démarches synthétiques de Wirth ou de Jackson.

4.6 La définition du formalisme UML et de la méthode associée

Initialement l'OMG s'était fixé l'objectif de définir une « méthode » unifiée d'analyse et de conception par objets. Une méthode est l'association d'une notation de définition de produits et d'un ensemble de processus génériques (démarche). Il est rapidement apparu que cet objectif était beaucoup trop ambitieux et qu'il avait peu de chances de pouvoir être mené à terme. Pour limiter le risque, l'OMG a donc, dans un premier temps, défini une notation de description d'artefacts logiciels. Une fois cet objectif atteint, il a été possible de mettre en chantier un nouveau travail de définition d'un processus générique associé à UML. C'est un exemple intéressant de "stratégie de zig-zag" qui consiste à progresser alternativement sur l'une puis sur l'autre branche d'un arbre de décomposition produit/procédé. Aujourd'hui qu'UML a été approuvé, l'OMG a lancé un nouveau RFP concernant le Software Process Engineering (SPE). Le but n'est pas de fournir une méthode standard, mais plutôt un framework que chacun pourra adapter à sa guise. Une des contraintes est de fournir un SPE particulièrement adapté à UML, d'être en mesure d'en exploiter toutes les capacités. Les connections entre les deux méta-modèles devront être particulièrement étudiées.

4.7 Des cas d'utilisation au modèle statique

Depuis leur introduction par Jacobson les cas d'utilisation sont très populaires pour la phase de collecte des besoins du système. Ils permettent de mettre à jour les processus les plus importants du système à concevoir et constituent une base de dialogue avec les utilisateurs. Pour pouvoir être exploité par les analystes, chaque cas d'utilisation va donner lieu à un certain nombre de scénarii. Ces scénarii ont pour but de recenser les différents cas qui peuvent se présenter lorsqu'un acteur externe utilise le système dans un but particulier. Ils vont décrire les interactions entre le système et les acteurs externes. En recoupant ces scénarii on va donc obtenir la structure générale du processus d'utilisation. On va pouvoir en extraire la structure de contrôle du processus, le rôle de chacun des acteurs, et les données échangées. On

a donc une vue dynamique du système. De cette vue dynamique on va dégager la vue statique en étudiant le flot de données. On passe donc d'une vue orientée processus à une vue orientée produit. Cela se fait par l'analyse des scénarii. Dans chaque scénario on va extraire les informations qui passent de mains en mains. Ce sont ces informations qui constitueront les fondements du diagramme de classe.

4.8 Intégration d'UML avec les Event-driven Process Chains (EPC)

Les EPC sont utilisées par de nombreuses sociétés pour modéliser leurs processus. Ils sont également utilisées par des outils de type ERP tels qu'ARIS. UML étant devenu le nouveau standard de modélisation des systèmes logiciels des essais ont été réalisés pour intégrer les EPC avec UML [12, 13]. Les EPC permettent de capturer le flot d'activités et les évènements qu'ils génèrent. On a ainsi un diagramme d'ordonnancement du processus. Ce diagramme peut être enrichi avec des informations telles que l'unité organisationnelle chargée de la réalisation d'une activité, ou les données manipulées par une activité. UML ne couvrant pas le même spectre que les EPC, il a fallu trouver les points d'interactions entre ces deux modèles. P. Loos et T. Allweyer ont donc établi quelques règles de correspondances entre les EPC et UML. Une fonction EPC peut être reliée à un package UML si ce package est responsable de son exécution, ou à une classe s'il est possible de descendre à ce niveau de granularité. Elle peut également être reliée à une méthode de classe si cette méthode est l'implémentation de la fonction EPC. Le diagramme d'état peut également être relié aux EPC. En effet, avec les EPC, il est possible de modéliser le changement d'état d'une entité après une fonction. Il n'y a alors plus qu'à relier ces états à ceux du diagramme d'état UML de l'entité en question. Le diagramme UML des cas d'utilisation peut reprendre certaines notions présentes dans le diagramme EPC, de même que le diagramme UML d'activité. Toutefois, même si l'on peut déduire directement ces deux diagrammes à partir des EPC via des mécanismes de traduction, de nombreuses informations seront perdues. Les autres diagrammes UML ont été jugés trop proches de l'implémentation pour pouvoir être utilisés dans ce contexte. On voit qu'on a donc deux principaux types de relations qui se dégagent entre UML et les EPC. La première consiste à établir des références entre des concepts de chacun des modèles, par exemple entre les classes et les fonctions. La deuxième consiste à établir des règles de traduction d'un modèle vers l'autre, c'est le cas du diagramme d'activité. Cette traduction s'accompagne quand même d'une perte d'informations lorsqu'on cherche à faire rentrer des EPC dans un diagramme d'activité UML.

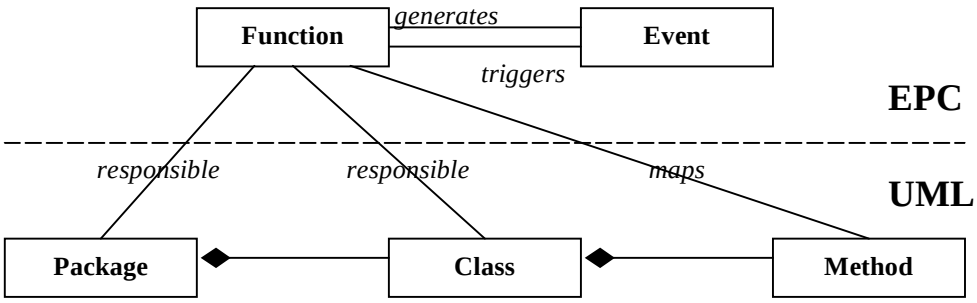

Figure 10 : Exemples de correspondance entre EPC et UML

4.9 Gestion de production

Le projet CRISTAL [4, 10] est un projet réalisé au sein du CERN dans le cadre de la production d'un détecteur utilisé pour la recherche de particules élémentaires. Ce détecteur est composé de plusieurs millions de composants dont la construction et l'assemblage sont répartis entre plusieurs sites disséminés à travers le monde. Le nombre de pièces à réaliser et le nombre de sites les réalisant imposent que les pièces, autant que les procédés qui s'y appliquent, soient clairement identifiés et identifiables. Pour cela un modèle a été mis au point qui mélange les aspects produits et procédés. Ce modèle est basé sur les travaux en cours dans le domaine de la méta-modélisation, et plus particulièrement sur les propositions de l'OMG, c'est-à-dire l'architecture à quatre niveaux et le MOF comme méta-méta-modèle. Ainsi, les concepts concernant la gestion de produit (PDM) et ceux concernant la gestion des procédés (Workflow) ne sont plus séparés en deux modèles différents, mais cohabitent au sein d'un même modèle.

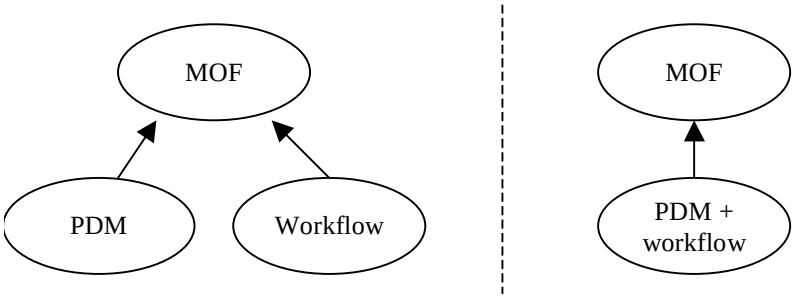

Figure 11 : Une architecture traditionnelle / l'architecture du projet CRISTAL

Comme les produits et les procédés sont dans un même référentiel, on peut établir des liens entre eux. On peut représenter la façon dont un procédé s'applique à un produit et les contraintes qui s'y attachent. On peut également déterminer les

différents procédés s'appliquant à un produit au cours de son cycle de vie. Ainsi, pour chaque produit on va définir les phases de conception, d'assemblage, de test et de maintenance. Ce modèle unique intégrant produit et procédé va donc servir de base d'informations universelle rassemblant toutes les données nécessaires à la réalisation du composant final. Cette intégration va donc apporter une plus grande facilité dans la consultation, mais également dans la conception et l'évolution concurrentes des produits et des procédés.

4.10 Autres exemples

Dans ce travail , nous continuons à rechercher des exemples de co-définition de la branche procédé et de la branche produit lors de la recherche de solution à un problème. L'un des domaines qui n'a pas été évoqué est celui de l'étude de l'utilisation de la méthode Merise. Ce n'est probablement pas le seul. Les éléments déjà recensés permettent cependant de conclure déjà à l'existence évidente d'un pattern. Une étude plus poussée va permettra de montrer comment la reconnaissance de ce pattern abstrait permet de proposer, dans certains champs d'application, des solutions issues de champs d'application très différents. Il y a en particulier deux disciplines où la distinction procédé/produit a été remarquée et signalée : la gestion de production et l'ingénierie des connaissances.

En ingénierie des connaissances, on peut par exemple citer J.L Ermine [5]: "*Beaucoup d'approches établissent une dichotomie entre les connaissances statiques (parfois nommées connaissances déclaratives) et les connaissances dynamiques (savoir-faire, procédures, …). La psychologie cognitive ne reconnaît pas cette dichotomie qu'elle juge non pertinente. Elle affirme même souvent que la connaissance statique est asservie à la connaissance dynamique selon l'adage "les objets sont structurés suivant l'action", base de la théorie de l'apprentissage par l'action. Ainsi, les méthodes de modélisation des connaissances statiques et dynamiques peuvent varier selon l'optique retenue, mis il y a toujours une interaction fondamentale entre ces deux types de savoir*". On peut mesurer, à partir de cette citation, la distance qui sépare les préoccupations des différentes disciplines mais également le schéma général qu'elles peuvent partager. De façon pratique, il est clair que l'on a beaucoup à gagner en comparant et en rapprochant des pratiques aussi différentes que la production de logiciel avec l'ingénierie de la connaissances en entreprise. Une méthode de génie cognitif comme MKSM (Methodology for Knowledge System Management) utilisée au CEA s'appuie par exemple fortement sur la séparation, et l'exploitation séparée, des connaissances statiques et dynamiques.

5 Conclusion.

Ce que nous avons d'abord voulu montrer dans cet article, c'est que la dualité P&P correspond à une situation récurrente en informatique. Elle est clairement apparente dans de très nombreux domaines. Elle a été maintes fois remarquée, mais elle a très

rarement donné lieu à une utilisation pratique. Elle n'a semble-t-il jamais été intégrée, à notre connaissance, dans une démarche méthodique systématique. Le chemin restant à parcourir avant cette intégration est sans doute encore très important.

Ce constat est fait dans un contexte nouveau que l'on peut caractériser sommairement par les deux éléments suivants :

- Le passage de la technologie procédurale à la technologie des objets a eu pour effet de créer un nouveau contexte :
  - où le nombre de modèles différents s'est considérablement accru,
  - où la complexité des modèles a augmenté de façon importante,
  - où chaque modèle est défini par un méta-modèle précis,
  - où les modèles de produits et les modèles de procédés forment une architecture régulière de production/consommation.
- La notion de patron, étudiée depuis une dizaine d'années, est maintenant relativement bien maîtrisée pour l'expression informelle de solutions génériques à des problèmes récurrents

Le nouveau contexte permet donc de réexaminer la relation produit/processus avec un espoir de pouvoir en tirer meilleur parti que par le passé. Il devrait ainsi être possible de se constituer une collection structurée de patrons s'appliquant aux méta-modèles de produits et de procédés. Les quelques exemples qui suivent constituent des illustrations de la démarche.

exemple n° 1

Une idée intéressante consiste à utiliser les observations de N. Wirth sur les correspondances entre structures de contrôle et structures de données dans les langages de programmation et d'essayer de les projeter dans le monde de la modélisation. En effet la situation qui prévaut en matière de modélisation d'organisation consiste à séparer complètement le contrôle (qui s'exprime par des workflow) et la modélisation de domaine (qui s'exprime souvent par des spécifications d'objets métier en UML). Ces deux mondes sont séparés. En observant la situation dans les langages de programmation, on peut se faire une idée de la façon d'intégrer les deux types de modèles, ou du moins de les mettre en correspondance. Il s'agit là d'un problème actuel et difficile. En effet, une spécification de workflow qui ne précise pas précisément les types de flux entre tâches est évidemment incomplète.

exemple n° 2

Considérons un fragment de modèle de procédé pour un repas :

```
SEQ
   SEQ
      choisir(Maître_d'hotel, Menu)
```

```
        choisir(Sommelier, Vin)
    PAR
        SEQ
            servir(Garçon, Vin)
            boire(Convive, Vin)
        SEQ
            servir(Garçon, Entrée)
            manger(Convive, Entrée)
            servir(Garçon, Plat_de_résistance)
            manger(Convive, Plat_de_résistance)
            servir(Garçon, Dessert)
            manger(Convive, Dessert)
```

Au vu de ce modèle de procédé (qui n'est qu'un exemple), on peut aisément déduire la structure du modèle de produit consommé, le repas. Le repas est ici composé d'un vin et d'un menu, ce dernier étant lui-même composé d'une entrée, d'un plat de résistance et d'un dessert. Toutefois ces simples liens de composition au niveau du modèle de produit ne sont pas assez riches pour en déduire la structure du procédé. Ils ne permettent pas de savoir qu'une relation d'ordonnancement existe entre les composants d'un repas. Il y a donc des informations concernant le produit, implicites au niveau du modèle de produit, qui ne sont explicitées que dans le modèle de procédé. C'est l'observation du modèle de procédé qui permettra de rendre compte de cette organisation qui va au-delà d'une simple contenance.

exemple n° 3

Le pattern du zig-zag a été évoqué à deux reprises, dans deux contextes différents (SADT et le formalisme UML). La définition d'un modèle générique de procédé logiciel par le groupe SPE (Software Process Engineering) de l'OMG est de ce point de vue significatif. Il consiste à repasser sur l'axe procédé alors que l'on venait de passer sur la branche produit avec UML. Ce dernier passage était lui-même le résultat d'un changement d'axe, car une méthode comme OMT se trouvait sur l'axe procédé. On pourrait remonter encore plus loin dans les démarches d'analyse et de conception procédurale (méthodes de la cascade et autres) et trouver trace de cette technique systématique de changement d'axe. Il est d'ailleurs tout à fait raisonnable de penser que la définition de procédés de développement de logiciel basés sur UML puisse amener à nouveau à étendre le formalisme et donc à changer d'axe à nouveau. Le pattern du zig-zag, partie plus spécifique du pattern P&P, répond bien à la définition générale d'un pattern puisque son utilisation est établie dans plusieurs contextes différents.

Au-delà de ces quelques exemples, il est raisonnable de penser que nous pouvons donner un éventail plus large de patterns. Il est surtout important de pouvoir donner des descriptions plus rigoureuses de ces correspondances.

Le consensus autour d'UML et l'arrivée à maturité de la technologie des objets ont permis de passer des techniques de production de logiciel basées sur le code vers des techniques basées sur les modèles. Cette transition est d'autant plus significative que l'OMG ne centre plus ses activités uniquement sur CORBA (Common Object Request Broker Architecture), mais aussi sur le MOF (Meta-Object Facility). Ceci permet de définir l'interopérabilité non plus seulement au niveau du code, mais aussi entre modèles. C'est cette nouvelle interopérabilité qui donne toute sa puissance au pattern P&P, autorisant l'exploration alternative de l'espace de définition produit et de l'espace de définition procédés et assurant le report aisé des modifications effectuées dans l'un des modèles vers l'autre.

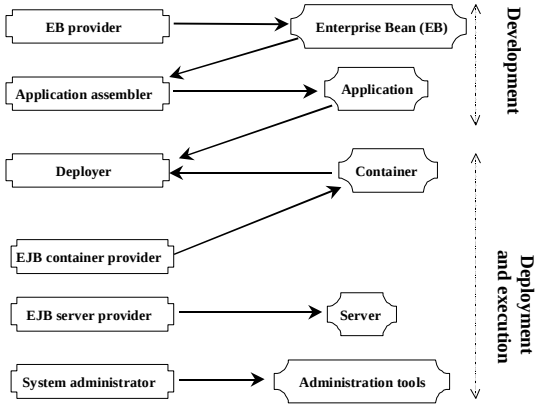

Figure 12 : Les rôles EJB d'après Bull Software [6]

Les applications potentielles du pattern P&P sont nombreuses et importantes. Il est même possible qu'il devienne incontournable. Si l'on considère la complexité rapidement croissante des modèles de composants comme le nouveau CCM de l'OMG (Corba Component Model) adopté en août 1999 à l'OMG, il est évidemment prévisible qu'elle va correspondre à des modèles encore plus complexes de procédés de création, de déploiement, de maintenance, etc. L'exemple de la Figure 13 extrait d'une présentation récente de Bull Software [6] montre le parallèle entre une ébauche de méta-modèles de produits et de procédés pour les EJB (Enterprise JavaBeans). Ce diagramme est simplifié, mais il traduit bien la complexité de la nouvelle situation que les structures de développement de logiciel vont devoir affronter. Dans cette situation nouvelle, le recours à l'utilisation de méta-modèles explicites et précis sera difficile à éviter.

Dans la colonne de gauche de la Figure 13 on trouve les éléments de procédés et dans la colonne de droite, les éléments de produits. Les flèches ont des significations diverses, mais elles montrent toutes des relations de production et de consommation entre les deux espaces.

De nombreux travaux portent actuellement sur la réalisation de méta-modèles explicites des EJB (par exemple travaux pour définir des modèles MOF des EJB). Un

tel méta-modèle peut être vu comme une extension du méta-modèles des Java Beans, qui lui même peut être vu comme une extension du méta-modèle du langage Java. Dans la pratique, le méta-modèle du langage Java va pouvoir prendre en compte des catégories syntaxiques et sémantiques. Nous appelons catégories syntaxiques des concept comme ceux de *Classe*, d'*Interface*, de *Méthode*, etc. Nous appelons catégories sémantiques des concepts comme *Exception*, *Evénement*, *SourceEvènement*, *Adapteur*, *Thread*, *Vue*, *Contrôleur*, etc. Il est bien clair que la définition hiérarchique de ces méta-modèles explicites des EJB devra être suivie d'une définition de méta-modèles explicites de procédés de développement de déploiement de ces EJB. L'espace des possibilités étant plus vaste pour les procédés que pour les produits, il faudra probablement prendre en compte la généricité des procédés.

L'objectif final de cette étude est de proposer des recommandations générales de développement coordonné produits/procédés, dans le domaine de la migration de systèmes d'information. Le travail amorcé ici sur le pattern P&P est encore très préliminaire et nécessite d'être complété.

6 Références bibliographiques.

[1] Bézivin, J., Bouchet, J.P. & Breton, E. **Revisiting the P&P Pattern with Explicit Meta-Models** INCOSE/LA Spring Conference, Pasadena, (April 1999).

[2] Bézivin, J., Ernst, J. & Pidcock, W. **Model Engineering with CDIF** OOPSLA'98, Vancouver, Conference post-proceedings, (October 1998).

[3] Bézivin, J., Lennon, Y. & Nguyen Huu Nhon, C. **From Cobol to OMT – a Reengineering Workbench Based on Semantic Networks** TOOLS USA'95, Santa Barbara, USA, (August 1995), Prentice Hall, pp. 137-152.

[4] Draskic J., Le Goff J.-M., Willers I., Estrella F., Kovacs Z, McClatchey R., Zsenei M. **Using a Meta-Model as the Basis for Enterprise-Wide Data Navigation**
http://computer.org/conferen/proceed/meta/1999/papers/58/rmcclatchey.html,
(1999).

[5] Ermine, J.L. **Les systèmes de connaissances** Hermes, (1996).

[6] Exertier, F. **Présentation des Enterprise JavaBeans** Workshop INRIA sur les composants, Bull Software, Valfréjus, (février 1999)

[7] Foote, B. & Yoder, J. **Metadata and Active Object Models** International Conference on Pattern Languages of Programs, Monticello, Illinios, (August 1998)

[8] Jackson, M.A. **Principles of Program Design** Academic Press, (1975)

[9] Microsoft, Microsoft Repository Product Information, **Open Information Model Overview**, (1999).

[10] Kovacs Z., McClatchey R., Le Goff J.-M., Baker N. **Patterns for Integrating Products and Process Models**, EDOC'99, Mannheim, Allemagne, (1999).

[11] Lissandre, M. **Maîtriser SADT** Armand Colin, (1990)

[12] Loos P., Allweyer T. **Process Orientation and Object-Orientation – An Approach for Integrating UML and Event-Driven Process Chains**, Paper 144, Publication of the Institut für Wirtschaftsinformatik, University of Saarland, Saarbrücken, Germany.

[13] Loos P., Allweyer T. **Process Orientation in UML through Integration of Event-Driven Process Chains** ,UML'98, Mulhouse, France, (juin 1998).

[14] Mills, H.D. **Chief Programmer Teams : Principles and Procedures**, IBM Corporation, Report N° FSC 71-5108 (Gaithersburg, MD., IBM Federal System Division, (1971)

[15] OMG/MOF **Meta Object Facility (MOF) Specification**. AD/97-08-14, Object Management Group, Framingham, Mass., (September 1997).

[16] OMG/SPE Analysis and Design PTF, **Software Process Engineering Request for Information**, Version 1.0, OMG Document ad/98-10-08, (13 November 1998).

[17] Otman, G. **Les représentations sémantiques en terminologie** Masson Sciences Cognitives, (1996).

[18] Robinson, K. & Berrisford, G. **Object-Oriented SSADM** Prentice Hall, (1994)

[19] Schulze, W. **Fitting the Worflow Management Facility into the Object Management Architecture** Business Object Workshop III, OOPSLA'97, Jeff Sutherland ed., (1997).

[20] Wirth, N. **On the Composition of Well-Structured Programs** International Computing Symposium 1973, Davos, Switzerland, (September 1973).

7 Lexique.

AD/Cycle----------Application Development Cycle (IBM)
ADTF----------Analysis and Design Task Force (OMG)
ATIS----------A Tools Integration Standard (DEC)
CCM----------CORBA Component Model (OMG)
CDIF----------CASE Data Interchange Format
COM----------Component Object Model (Microsoft)
CWMI----------Common Warehouse Metamodel Interchange
DCOM----------Distributed COM (Microsoft)
DTD----------Document Type Definition (XML)
EIA----------Electronics Industry Associates
EJB----------Enterprise Java Beans
EPC----------Event-driven Process Chains
IDL----------Interface Definition Language (OMG)
IRDS----------Information Resource Dictionary System
MDC----------Meta-Data Coalition
MOF----------Meta-Object Facility (OMG)
OCL----------Object Constraint Language
OIM----------Open Information Model (Microsoft)
OMG----------Object Management Group
PCTE----------Portable Common Tool Environment
PDM----------Product Data Management
PWG----------Process Working Group (OMG)
RFP----------Request For Proposal
RTIM----------Repository Type Information Model
SMIF----------Serial Model Interchange Format (OMG)
SPE----------Software Process Engineering
Uml----------Méta-modèle Microsoft correspondant à UML 1.0 de l'OMG
UML----------Unified Modeling Language
Umx----------Extensions propres à Microsoft de Uml
W3C----------World Wide Web consortium
XIF----------XML Interchange Format (Microsoft)
XMI----------XML Model Interchange (OMG)
XML----------eXtended Markup Language