N° d'ordre : 3059

THESE

PRESENTEE A

L'UNIVERSITE BORDEAUX 1

ECOLE DOCTORALE DES SCIENCES PHYSIQUES
ET DE L'INGENIEUR

Par M. Matthieu ROQUE

POUR OBTENIR LE GRADE DE

DOCTEUR

SPECIALITE : PRODUCTIQUE

Contribution à la définition d'un langage générique de modélisation d'entreprise

Soutenue le 15 novembre 2005

Après avis de MM.

J.P. Bourey	Rapporteur
Professeur à l'Ecole Centrale de Lille	
H. Pingaud	Rapporteur
Professeur à l'Ecole des Mines d'Albi Carmaux	

Devant la commission d'examen formée de :

MM.

J.P. Bourey	Rapporteur
Professeur à l'Ecole Centrale de Lille	
H. Pingaud	Rapporteur
Professeur à l'Ecole des Mines d'Albi Carmaux	
G. Doumeingts	Directeur de thèse
Professeur Emérite à l'Université Bordeaux 1	
B. Vallespir	Co-directeur de Thèse
Professeur à l'Université Bordeaux 1	
M. Petit	Examinateur
Professeur à l'Université de Namur	
T.Despoix	Examinateur
Co-gérant de la société Openxtrem	
D. Chen	Examinateur
Maître de Conférences à l'Université Bordeaux 1	

— 2005 —

A mamie

Remerciements

Les travaux présentés dans ce mémoire ont été menés au Laboratoire d'Automatique, Productique Signal et image (LAPS) de l'Université Bordeaux I. A ce titre, je tiens à remercier Monsieur Guy Doumeingts, Professeur Emérite à l'Université Bordeaux 1, pour m'avoir accueilli au sein de son équipe.

Je tiens à exprimer toute ma gratitude à Monsieur Bruno Vallespir, Professeur à l'Université Bordeaux I. Ces remarques pertinentes, la clarté de ses idées ont largement contribué à la réalisation de ce document.

Je souhaite également remercier Monsieur Hervé Pingaud, Professeur à l'Ecole des Mines d'Albi Carmaux et Monsieur Jean-Pierre Bourey Professeur à l'Ecole Centrale de Lille, d'avoir accepté de rapporter ce mémoire de thèse, ainsi que pour l'attention et les remarques constructives qu'ils ont apportées.

Je souhaite exprimer toute ma reconnaissance à Monsieur Michael Petit, Professeur à l'Université de Namur, pour avoir examiné mes travaux et dont les commentaires avisés m'ont permis d'entrevoir d'autres pistes de recherche.

Que Monsieur David Chen, Maître de Conférences HdR à l'Université Bordeaux 1, trouve ici toute ma reconnaissance pour sa participation à mon jury de thèse.

Je tiens à remercier Monsieur Thomas Despoix, Co-gérant de la société Openxtrem, d'avoir accepté d'examiner ce mémoire de thèse.

Enfin, je veux également remercier mes « potes » qui m'ont toujours soutenu en me disant « c'est quand que tu trouves un vrai boulot ». Ma thèse étant finie je vais pourvoir m'atteler à cette tâche.

Table des matières

Liste des figures

Figures du chapitre 1 : Contexte et problématique

Figures du chapitre 2 : Etat de l'art

Figures du chapitre 3 : Principes, démarche et illustration empirique

Figures du chapitre 4 : Méta-modélisation et écriture formelle

Figures du chapitre 5 : Exemple d'élaboration d'un UEML

Liste des tableaux

Tableaux du chapitre 1 : Contexte et problématique

Tableaux du chapitre 2 : Etat de l'art

Tableaux du chapitre 4 : Méta-modélisation et écriture formelle

Tableaux du chapitre 5 : Exemple d'élaboration d'un UEML

Introduction générale

Durant les vingt dernières années, la modélisation d'entreprise s'est imposée comme un moyen puissant au service de l'organisation industrielle dans une optique d'amélioration des performances. Un grand nombre de méthodes a alors émergé notamment aux Etats-Unis et en Europe. Pour l'Europe, les programmes de recherche de la Commission Européenne ont largement permis de développer et de diffuser ces outils, ceci depuis le premier Programme Cadre de Recherche et Développement (programme Esprit). Il en découle aujourd'hui un domaine scientifique reconnu au niveau international avec une finalité applicative industrielle affirmée.

Ce développement a conduit à un nombre important d'outils et de méthodes possédant tous une spécificité mais partageant avec d'autres une zone de recouvrement non négligeable, que ce soit en terme de concepts manipulés ou de domaines applicatifs. Ainsi, face à cette diversité, nous entrons aujourd'hui dans une deuxième période qui conduit plus à un rapprochement, voire une intégration, des approches existantes plutôt qu'à l'émergence de nouvelles. Les initiatives auxquelles nous assistons aujourd'hui illustrent ces tentatives de rapprochement en développant un langage unifié de modélisation d'entreprise (Unified Enterprise Modelling Language).

Dans ce cadre, cette thèse consiste à apporter des premiers résultats concernant la définition d'un langage unifié de modélisation d'entreprise. L'objectif de la recherche n'est pas de rendre le langage UEML opérationnel et de l'intégrer dans une démarche de modélisation, comme c'est le cas pour la majorité des langages initiaux. En effet, ces derniers sont utilisables dans le cadre de l'étude pratique par le fait qu'ils sont intégrés dans les démarches de modélisation (ingénierie système) reconnues et qui ont fait leurs preuves. En revanche, le langage unifié permettra une interopérabilité des modèles même lorsque ceux-ci ont été réalisés avec des langages différents. L'objectif de la recherche n'est donc pas de proposer un langage UEML complet, mais plutôt de faire ressortir une certaine problématique, ainsi

qu'une démarche d'élaboration de ce langage. Le sujet étant relativement nouveau, cette thèse se veut être donc une thèse prospective dans le domaine.

Ainsi, le **chapitre 1** de ce mémoire présente la problématique de ce sujet de thèse. Cette problématique sera divisée en trois parties principales. La première partie présentera brièvement l'entreprise ainsi que son système de production, sujet principal de notre étude. Ensuite, nous aborderons le thème de la modélisation d'entreprise, cadre dans lequel s'inscrit notre problématique. Enfin, la dernière partie traitera de l'intérêt proprement dit de ce sujet de thèse. Elle présentera les besoins et les enjeux économiques qui motivent ces travaux de recherche.

Dans le **chapitre 2**, nous présenterons différentes approches qui visent d'une manière ou d'une autre, une unification des langages. Les travaux sur la normalisation, par exemple, fournissent des résultats intéressants (l'ISO 14528, l'ISO 15704, l'EN/ISO 19439, l'EN/ISO 19440, ODP, PSL...). Ensuite, nous exposerons une architecture permettant de comparer les langages. Cette architecture a été développée dans le cadre d'un projet Européen : le réseau thématique UEML (UEML-IST-2001-34229), auquel nous avons participé. Nous montrerons par son utilisation que les différents langages existants ne permettent cependant pas de répondre à notre problématique. Nous parlerons également de la problématique concernant l'intégration des bases de données car elle est relativement similaire à la notre.

Le **troisième chapitre** de ce mémoire, traitera de la première partie de notre contribution. Dans ce chapitre, nous exposerons, dans un premier temps, les principes et la démarche sur lesquels nous nous basons concernant le langage UEML. Dans un deuxième temps, nous présenterons un des aspects très important pour la construction de notre langage UEML : la « traduction de composants ». Nous montrerons un exemple simple de traduction entre une activité SADT et une activité GRAI (composants des réseaux GRAI). Dans ce chapitre, cette traduction s'appuiera sur une approche empirique.

La démarche que nous utilisons dans cette thèse est basée sur la méta-modélisation des langages. Ainsi, dans le **chapitre 4,** nous présenterons les concepts relatifs à la méta-modélisation et nous montrerons également qu'il est important de considérer les aspects du triangle sémiotique (signifiant, signifié et référent) des composants des langages. Enfin,

nous dégagerons certaines situations que l'on rencontre lors de la comparaison des composants des langages. Nous exposerons ces résultats d'une manière plus formelle que dans le chapitre précédent, notamment par l'utilisation d'une approche ensembliste.

Enfin, dans le **cinquième** et dernier **chapitre** de ce mémoire, nous traiterons un exemple d'élaboration d'un langage unifié de modélisation d'entreprise. Cet exemple a été réalisé lors du réseau thématique UEML, en partant de trois langages de modélisation d'entreprise : GRAI, IEM (Integrated Enterprise Model) et EEML (Extended Enterprise Modelling Language) et en procédant par une approche ascendante.

La conclusion de ce mémoire, outre le bilan des propositions, présentera les perspectives envisagées pour la suite de ces travaux.

Chapitre I

Contexte et problématique

Chapitre I

Contexte et problématique

1.1 Introduction

Les travaux présentés dans cette thèse s'inscrivent dans le domaine de la modélisation d'entreprise et concernent plus particulièrement le développement d'un langage unifié de modélisation d'entreprise. Afin de mieux comprendre les enjeux économiques qui motivent ces travaux de recherche, il est nécessaire de bien en fixer le contexte. C'est pourquoi, dans ce premier chapitre, nous allons tout d'abord présenter brièvement ce que l'on entend par entreprise et par son système de production. Nous montrerons ainsi la complexité de fonctionnement de ce type d'organisation et donc la nécessité de posséder un langage de modélisation permettant de la décrire afin de la maîtriser. Ce modèle étant donné grâce à la modélisation d'entreprise, nous aborderons, par la suite, ce sujet proprement dit. Nous exposerons son but ainsi que ces domaines d'applications. En s'appuyant sur ces propos, nous exposerons la problématique spécifique à la définition d'un langage unifié de modélisation d'entreprise. Nous présenterons dans un premier temps le contexte dans lequel ces travaux se placent et pourquoi la nécessité de développer un langage unifié de modélisation d'entreprise se fait sentir aujourd'hui. Enfin, nous dirons quelques mots sur le « devenir » de la modélisation d'entreprise, dans les années futures.

1.2 L'entreprise et son système de production

1.2.1 L'entreprise

La notion d'*entreprise,* telle qu'elle est comprise dans le cadre de la modélisation d'entreprise, et plus généralement, dans le cadre de la productique, réfère à un ensemble organisé d'activités mises en œuvre par des ressources socio-techniques, dans le cadre d'une finalité identifiée. Dans de tels systèmes, la dimension financière est généralement présente, que ce soit d'un point de vue gain, ou plutôt de consommation de ressources financières [GRP/GT5, 1999].

Nous considérons l'entreprise comme un système, au sens systémique du terme. Dans ce cadre là, un élément fondamental est le principe de non séparabilité, où l'ensemble A + B n'est pas simplement et uniquement considéré comme une addition. La simple connaissance de A et de B, ne suffit pas pour en déterminer la somme. En d'autres termes, l'entreprise ne se résume pas à l'addition de ses parties visibles. Elle dépend également de liens non observables, de liens informels entre collaborateurs, clients et fournisseurs, de liens impossibles à identifier. N'importe lequel des éléments de l'entreprise peut avoir une influence sur l'ensemble : la réclamation d'un client, la maladie d'un salarié, la hausse du prix des produits, la panne d'une machine, par exemple, ont des répercussions sur la bonne marche de toute l'entreprise.
L'entreprise est un système « ouvert » à son environnement, finalisé et dirigé. Elle poursuit des buts (profit, puissance, pérennité ...), s'organise pour les atteindre (définition de plans d'action, de budgets ...), se dote de structures d'exécution, de direction et de contrôle. L'entreprise est un ensemble de sous-systèmes en interaction.

Pour l'entreprise contemporaine, l'organisation du travail n'est plus centrée sur le poste de travail et sur les performances individuelles, mais sur la cogérance, la coopération et la communication entre différents acteurs, services et partenaires. En outre, le problème de la gestion des connaissances et de la capitalisation du savoir-faire se pose de manière récurrente dès lors que l'entreprise se voit confrontée au départ de travailleurs expérimentés ou à l'arrivée de nouveaux collaborateurs.

Enfin, c'est à différents niveaux que les problèmes associés à l'organisation et à la définition du travail dans l'entreprise peuvent s'exprimer : au niveau des clients, des fournisseurs, de la concurrence et enfin au niveau des salariés de l'entreprise.

- Face au client, il s'agit d'offrir les meilleurs produits et services à des prix compétitifs : le client est alors le maître mot de l'entreprise; de lui dépend la survie de cette dernière.

- Face aux fournisseurs, l'entreprise doit rechercher les meilleurs contrats, les meilleures coopérations, les meilleures alliances. De plus, dans l'optique conjoncturelle de la réduction des coûts, cette dernière est souvent amenée à réduire (voire éliminer) certaines de ses activités en les sous-traitant. Emergent alors d'autres problèmes tels que les licenciements ou la réorientation d'une partie du personnel.

- Face à la concurrence, l'entreprise se doit de savoir se positionner, reconnaître ses atouts et ses faiblesses, déterminer ce qui la distingue vis-à-vis de ses concurrentes.

- Enfin, face aux salariés, de par la conjoncture économique et les objectifs de l'entreprise, les problèmes (licenciements) qui peuvent se poser n'ont souvent pas de solution évidente.

Dans le cadre de nos travaux, nous nous limitons à la notion d'entreprise d'un point de vue de la productique et nous nous focalisons donc sur le système de production de l'entreprise.

1.2.2 Le système de production dans l'entreprise

Le système de production au sens où nous l'entendons n'est donc qu'un sous-ensemble d'une entité plus grande : l'entreprise pour laquelle on définit des finalités multiples; nous retiendrons notamment qu'elle remplit un rôle économique, mais également social et humain. Parmi ces trois points, l'aspect économique a très souvent pris le pas sur les deux autres.

Dans ce contexte là, le système de production vise à remplir la fonction « Métier » de l'entreprise, c'est-à-dire, l'ensemble des activités que l'entreprise se charge d'effectuer pour satisfaire les besoins d'un client, ou éventuellement d'un partenaire. Pour assumer son rôle de

« fournisseur de solutions », il met en œuvre un ensemble de ressources, humaines et techniques, qui vont apporter de la valeur aux flux des produits transformés tout au long du processus de génération.

La production est une transformation de ressources appartenant à un système productif et conduisant à la création de biens ou de services. Les ressources mobilisées à cette fin peuvent être de quatre types : des équipements (bâtiments, machine…), des hommes (opérateurs intervenant directement dans le processus de transformation ou contribuant d'une manière ou d'une autre à son bon déroulement), des matières (matières premières, composants…) et des informations techniques ou procédurales (gammes, nomenclatures, consignes, procédures…) ou relatives à l'état et à l'utilisation du système productif (ce qui permet de programmer la production et de réagir aux perturbations observées) [Giard, 2003].

Dans l'environnement concurrentiel et fluctuant de l'entreprise, quelques critères qui permettent de caractériser la performance d'un système de production sont :

- La productivité, à savoir la quantité de produits par unité de temps, par ressource utilisée ;
- La flexibilité, qui caractérise la capacité du système de production à changer de configuration en fonction de la demande ;
- La qualité, qui mesure l'aptitude du produit à satisfaire les besoins du client.

De plus, le système de production doit s'intégrer aux autres systèmes de l'entreprise qui le sollicitent, il doit continuellement s'adapter.

Ainsi, le système de production est un système complexe. Cette complexité et son rôle peuvent en partie s'expliquer par sa position dans l'entreprise. Une entreprise peut être décomposée en un ensemble de fonctions formant un réseau, dont les éléments de liaison sont des flux de matières, d'informations, de décisions et de finances.
Suivant la taille et les principes de gestion de l'entreprise, ce découpage en fonctions et en services peut être différent. Mais dans tous les cas, comme l'indique la Figure 1.2-1, les relations entre ces différentes fonctions entraînent une forte connexion de la fonction

production avec les autres fonctions de l'entreprise, ainsi qu'avec son environnement. [Doumeingts, 1994].

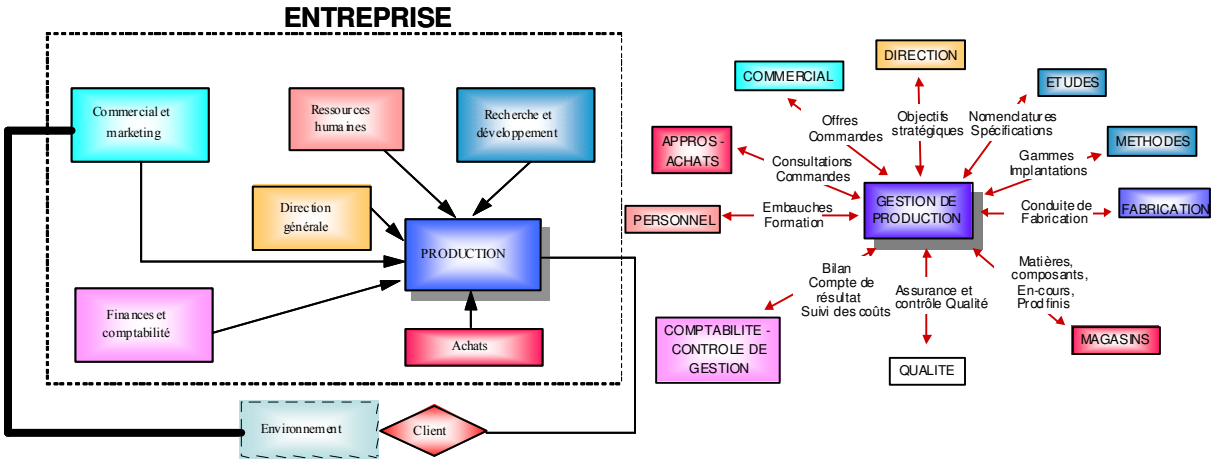


Figure 1.2-1 : Les différentes fonctions de l'entreprise et leurs liaisons avec la production

De plus, les objectifs des services avec lesquels la fonction production est en relation, sont en général contradictoires avec les siens. On peut, par exemple, noter que les besoins de la fonction production d'une entreprise, en terme du classique triptyque coût, qualité, délai, sont contradictoires avec ceux de la fonction commerciale (et donc de ceux du client) (Tableau 1.2-1).

	FONCTION PRODUCTION	***FONCTION COMMERCIALE***
TEMPS	Posséder un système de production le plus stable possible.	Délai de réaction le plus court possible
QUALITE	Il est plus difficile de produire un produit de qualité	Il est plus facile de vendre un produit de qualité
COUTS	Les contraintes de coûts sont difficiles à tenir en production	Un produit est plus facile à vendre s'il est bien positionné sur son marché

Tableau 1.2-1 : Système de production et fonction commerciale

Le fait que ces deux fonctions aient des objectifs opposés rajoute une difficulté supplémentaire à la compréhension du système de production. En effet, afin d'avoir un système de production performant, il est nécessaire que ces deux fonctions essaient de

collaborer, afin d'aller vers un compromis permettant de satisfaire au mieux les objectifs de chacun et donc de travailler dans « le même sens », plutôt que de s'opposer. Ce problème se retrouve donc également avec les autres services avec lesquels le système de production est en relation.

Ainsi, le système de production peut être vu comme la combinaison de trois grandes groupes en interactions : des ressources humaines, des ressources matérielles (machines, bâtiments, etc.) et d'autres éléments immatériels (organisations, savoir-faire, brevets, etc.). La diversité des éléments qui composent ces trois groupes et la multiplicité des relations et interactions qui existent entre ces éléments créent un univers d'une grande complexité. Le bon fonctionnement de l'entreprise dépend donc de la capacité des décideurs à parvenir à faire agir tous ces éléments de manière efficace et cohérente avec leurs objectifs.

La difficulté de compréhension du fonctionnement d'une entreprise est accentuée par le fait que l'ensemble du savoir est distribué en son sein. De même, un autre problème qui se pose est la perception que possède chaque acteur d'une entreprise, du milieu dans lequel il évolue, c'est à dire donc de l'entreprise elle-même. Par exemple, le directeur d'une entreprise ne possède pas le même niveau de détail d'informations que le responsable du bureau d'études, car il n'en fait pas le même usage. De même, le responsable d'un atelier d'usinage ne parlera à priori pas du travail de ses ouvriers avec la même précision ou la même rigueur que les employés eux-même. Il a une vision agrégée de son atelier. Ainsi, les différents décideurs possèdent des connaissances différentes du système dans lequel ils évoluent. Cependant, afin que l'ensemble des acteurs d'une entreprise puisse tenir compte des objectifs et contraintes des autres, il est impératif qu'ils aient une vision commune de l'état actuel de celle-ci. Cette image peut ainsi leur permettre de comprendre le fonctionnement de l'entreprise et ainsi de gérer la diversité des relations et interactions entre les différents éléments de ce système. Cette vision évolue en même temps que l'entreprise en prenant en compte les aspects dynamiques. Cette image peut être donnée grâce à la modélisation d'entreprise.

1.3 La modélisation d'entreprise

Dans le cadre de la productique, la modélisation d'entreprise concentre généralement ses efforts sur le *système de production*, partie de l'entreprise correspondant aux activités de création de valeur. Comme, dans la pratique, il est rarement possible de faire une distinction nette entre les activités à valeur ajoutée et les autres, le domaine d'étude couvrira souvent le système de production ainsi que les fonctions logistiques connexes (ingénierie produits,

maintenance, etc.). Même si pendant longtemps, seule la production de produits tangibles a été envisagée, la tendance actuelle est de prendre en compte de la même manière la production de produits tangibles, la production de services et la production mixte (produits tangibles + services). La nouvelle donne économique, favorisant la mise en réseaux des acteurs économiques au profit d'un projet collectif, conduit aujourd'hui la modélisation d'entreprise à pousser un cran plus loin ce point de vue en s'intéressant aux réseaux d'activités technico-économiques de dimensions dépassant les frontières statutaires de l'entreprise. Dans ce sens, la modélisation d'entreprise a et aura un rôle majeur dans l'analyse des réseaux complexes de transactions envisagés dans la thématique émergente de l'entreprise étendue [GRP/GT5, 1999].

L'entreprise est donc une réalité toujours plus complexe qu'il est nécessaire de modéliser pour, d'une part, la rendre intelligible et, d'autre part, autoriser le raisonnement d'un acteur ayant un projet bien défini en son sein. En d'autres termes, un modèle d'entreprise a pour objectif de formaliser tout ou partie de l'entreprise dans le but de comprendre ou d'expliquer une situation existante ou pour réaliser, puis valider, un projet conçu [Braesch *et al*., 1995]. Un modèle est toujours construit sur la base d'un langage que celui-ci soit informel (langage naturel, par exemple), semi-formel[1] (langage au formalisme essentiellement graphique par exemple) ou formel (langage mathématique). La plupart du temps, les modèles basés sur un langage informel sont utilisés pour décrire une situation existante tandis que les modèles basés sur un langage formel permettent la vérification des propriétés fixées dans un projet donné [Chapurlat *et al.*, 1999]. De même, un modèle d'entreprise est toujours associé à une finalité et il doit, suivant les besoins, être capable de prendre en compte les aspects structurels, fonctionnels et comportementaux. Outre ces aspects, il doit également être capable d'appréhender le point de vue particulier d'un acteur. Enfin, une démarche spécifique est souvent associée à chaque langage de modélisation. Cette démarche explicite les différentes étapes nécessaires à la construction et à l'exploitation du modèle (cycle de vie) ; chaque étape étant souvent caractérisée par un niveau d'abstraction (niveaux conceptuel, organisationnel ou technique).

[1] Un langage semi-formel s'appuie généralement sur un vocabulaire et une syntaxe formels (les éléments du langage et les règles syntaxiques sont parfaitement définis) mais ne dispose pas de sémantique opérationnelle lui permettant de vérifier les propriétés comportementales.

1.3.1 Décomposition du système de production

La théorie des systèmes [Simon, 1960, 1977] [Le Moigne, 1974, 1977, 1990] et [Mélèse, 1972] permet d'identifier dans les systèmes socio-technico-économiques deux types d'activités : des activités pilotées (ou opérantes) qui transforment des entités et des activités de pilotage ou de conduite, qui pilotent ces transformations (Figure 1.3-1).

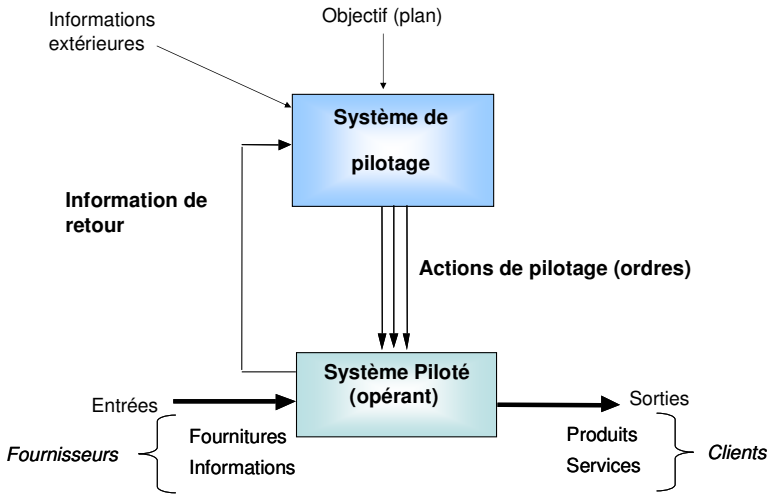


Figure 1.3-1 : Décomposition d'un système socio-technico-économiques

En se basant sur ces théories, il est possible de décomposer le système de production en trois sous-systèmes [Doumeingts, 1998] : le *Système Physique de Production* (SPP) ou système piloté, le *Système de Décision* et *le Système d'Information*, ces deux derniers étant regroupés dans le Système de Gestion de Production (SGP) qui pilote le système physique (Figure 1.3-2).

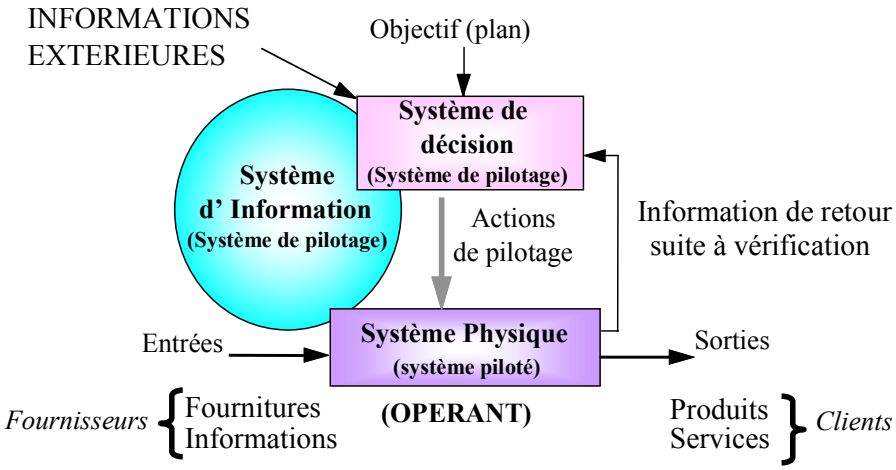


Figure 1.3-2 : Décomposition du système de production

En d'autres termes, le SPP transforme les matières premières et produits semi-finis en produits finis. Pour effectuer cette transformation, il est piloté par le SGP qui transforme les informations à caractère commercial en ordres de fabrication et en ordres d'approvisionnement. Ce système est bien entendu bouclé, puisqu'en retour il reçoit les informations du SPP pour pouvoir effectivement piloter ce dernier.

1.3.1.1 Le système physique de production

Le système physique de production est traversé par un flux de matières premières ou de produits semi-finis qu'il transforme en produits finis par ajout de valeur.
Le système physique décrit la transformation du flux physique, c'est-à-dire la transformation des produits (un produit sera immatériel dans le cas du service), par les ressources (machines, hommes, moyens divers). Ce flux part du fournisseur, « Entité » qui fournit le « Produit » à transformer, et aboutit au client (destinataire du produit) [Doumeingts, 1998]. Il peut aussi partir du client : par exemple pour la demande de la réalisation d'un produit.

1.3.1.2 Le système de décision

Le système de décision transmet au système physique de production des ordres ou commandes, résultats de nombreuses prises de décision et de traitements d'informations. En d'autres termes, le système de décision élabore l'ensemble des décisions qui permettent de piloter le système physique, en s'appuyant sur le système d'information.

1.3.1.3 Le système d'information

Ce système est chargé de collecter, stocker et distribuer, l'ensemble des informations nécessaires au système de production. Les informations peuvent être d'origines externes au système de production (informations concernant l'environnement) ou internes (liens entre le système de décision et le système physique).

Ainsi, ces différents systèmes agissent afin que l'ensemble atteigne les objectifs globaux assignés au système de production.

1.3.2 Domaines d'utilisation de la modélisation d'entreprise *[GRP/ GT5, 1999]*

La Modélisation d'Entreprise intervient durant toutes les phases de la vie de l'entreprise : de sa création jusqu'à son démantèlement et la réutilisation éventuelle de certaines de ses entités en passant par son évolution.

La **conception** d'un système de production va utiliser la modélisation d'entreprise comme outil de spécification dans le sens où ce sont les modèles construits qui vont permettre l'expression [Doumeingts *et al.*, 1987], [Chen *et al.*, 1999], [Zanettin, 1994] :

- des besoins de l'entreprise cliente et des performances attendues,
- de l'organisation fonctionnelle du système futur,
- de la spécification et de l'intégration des composants techniques, humains et organisationnels.

Le défi que doit relever la modélisation d'entreprise dans ce cadre est d'être capable :

- de traduire les attentes de l'entreprise cliente en terme de performances et de fonctionnalités futures (analyse des besoins),
- d'évaluer le système et les solutions futures,
- d'être un support à l'activité de conception du système futur (en terme de fonctions et d'architecture),
- de permettre une traduction la moins coûteuse possible vers les métiers en charge de la spécification et de l'implantation des solutions (ingénierie, informatique, organisation humaine, etc.).

Dans ce cadre, la modélisation d'entreprise doit permettre de faire émerger l'ensemble des connaissances partagées par les partenaires (internes à l'entreprise et externes), partenaires possédant des objectifs, des compétences et des points de vue différents. Elle devra ensuite permettre de les intégrer afin de fournir à l'ensemble des acteurs une base sémantique unique.

L'évaluation des systèmes de production vise à connaître les performances de ces derniers. Nous avons déjà évoqué les multiples dimensions d'un système de production, ces dimensions possèdent chacune leur propre outils d'évaluation. Aussi, si « *l'évaluation mono-disciplinaire »* est aujourd'hui relativement bien fournie en outils, il en va autrement de l'évaluation globale du système. Permettre de mettre face à face des performances techniques, économiques et socio-humaines dans le cadre d'une démarche unifiée d'évaluation est un problème auquel ne correspond pas aujourd'hui de réponse totalement satisfaisante et pour lequel la modélisation d'entreprise a un rôle d'intégration à jouer [Vallespir *et al.*, 1992], [Vallespir *et al.*,1993].

L'évaluation est en général utilisée pour comparer les performances existantes à des performances attendues. Toutefois, il existe d'autres configurations basées sur l'évaluation. Il faut citer notamment le benchmarking qui a comme objectif de comparer un système de production ou une de ses parties avec d'autres cas similaires afin d'en déduire des voies d'amélioration [Vallespir *et al.*, 1999].

Par la suite, si l'évaluation des performances fait ressortir certaines faiblesses du système de production, on peut utiliser la modélisation d'entreprise en vue d'en améliorer l'efficacité. On parle alors de restructuration (re-engineering) du système de production. Ainsi, la restructuration des systèmes de production a pour objectif l'amélioration des performances d'un système existant [Doumeingts *et al.*, 1995]. Il est notable que, même dans le cas d'un système complètement nouveau, la conception de celui-ci telle qu'évoquée plus haut, partira toujours de connaissances de bases. En effet, outre l'expression des besoins, l'entreprise cliente aura une image diffuse de ce qu'elle attend du système futur sur la base de sa propre expérience (et, plus exactement, de l'expérience des femmes et des hommes de l'entreprise impliqués dans le projet). De plus, le contexte économique fait qu'il est beaucoup plus courant de faire évoluer des installations existantes que d'en créer de nouvelles. Aussi, les entreprises

d'aujourd'hui sont beaucoup plus souvent demandeuses d'action de restructuration que d'actions de création.
Cependant, l'essentiel de ce qui a été présenté dans le cadre de la conception reste vrai pour la restructuration. Il faut quand même ajouter à la capacité d'évaluer le système et les solutions futures celle d'évaluer le système existant.

On peut finalement dresser la liste (non exhaustive) suivante, présentant quelques domaines d'utilisations de la modélisation d'entreprise :

- disposer d'une référence concernant le fonctionnement de l'entreprise, en apportant un modèle d'entreprise permettant à l'ensemble des acteurs d'avoir une vision commune du fonctionnement du milieu dans lequel il évolue,

- recherche d'améliorations de performance (détection des points à améliorer et des points forts du système étudié, restructuration, re-engineering ...),

- établissement de spécifications et de cahiers des charges pour développer des applications informatiques, choisir des progiciels du marché : ERP (Enterprise Resources Planning), SCM (Supply Chain Management), …

- conception et implantation d'un système d'Indicateurs de Performance,
- réalisation d'études de Benchmarking de processus ou de Benchmarking d'Entreprise,

- simulation de processus,

- gestion des connaissances (acquisition, mémorisation, réutilisation),

- ...

1.4 Contexte et nécessité d'un langage UEML

Pour bien fixer le contexte et donc la nécessité de développer un langage UEML, nous allons organiser ce paragraphe en deux parties. Dans une première partie, nous montrerons que le développement de la modélisation d'entreprise a donné naissance à beaucoup de langages et

donc qu'il est nécessaire de créer aujourd'hui un langage unifié de modélisation d'entreprise. Dans une deuxième partie, nous montrerons que bien que le contexte industriel a et va évoluer au cours des années, il est raisonnable de penser que le domaine de la modélisation d'entreprise reste un domaine qui a son utilité, même si ses applications peuvent évoluer au fil des années. Ainsi, les travaux de recherche effectués dans cette thèse se justifient d'autant plus.

1.4.1 Développements de la modélisation d'entreprise

Les premiers développements en modélisation d'entreprise ont été menés aux Etats-Unis dans les années 1970 et ont notamment produit SADT, SSAD, IDEF0 ou les Data Flow Diagram. Pour l'Europe, les programmes de la Commission Européenne ont largement permis de développer et diffuser ces outils, ceci depuis le premier Programme Cadre de Recherche et Développement (programme Esprit, années 80). Ainsi, un grand nombre de langages de modélisation d'entreprise sont apparus en Amérique du nord et en Europe. Nous pouvons citer, par exemple, CIM-OSA, IEM, IDEFx, METIS ou ARIS Toolset [Petit *et al.*, 2002], [Vernadat, 1999a].

Au niveau français, évoquons les méthodes présentées dans le cadre d'une école sur la modélisation d'entreprise organisée en mai 2002 par le groupe « Modélisation d'entreprise » du Groupement de Recherche en Productique : ACNOS [El Mhamedi *et al.*, 1997], [El Mhamedi, 2002], GRAI [Doumeingts, 1984], [Vallespir *et al.*, 1997], [Vallespir *et al.*, 2002], MECI [AICOSCOP, 1990], [Pourcel *et al.*, 2002], OLYMPIOS [Braesch, 2002], [Theroude, 2002]. Si l'on ajoute GIM, élaborée autour de GRAI dans le cadre de projets Esprit, et les participations à CIM-OSA [AMICE, 1993], nous pouvons remarquer que la recherche française a joué un rôle significatif dans ces développements.
La conséquence de ce foisonnement initial est qu'il existe aujourd'hui trop de langages de modélisation d'entreprise hétérogènes disponibles sur le marché.

Ce constat peut être expliqué par :

- *Les diverses bases théoriques sur lesquelles les langages de modélisation ont été élaborés.* Par exemple, CIM-OSA a été développé principalement par des chercheurs

venant de l'informatique, alors que la méthode GRAI est basée sur la théorie des systèmes, la théorie du contrôle et la gestion de production.

- *Les domaines d'utilisation.* Par exemple, MERISE et M* [Vernadat, 1999a] ont été développées spécifiquement pour la conception des systèmes d'information, alors que IDEF3 [Mayer *et al.*, 1992] est destinée à la modélisation et la reconception des processus.

Cependant, les problèmes qu'entraînent cette situation sont [Chen *et al.*, 2002], [Roque *et al.*, 2002], [Roque *et al.*, 2004] :

- une difficulté, voire une impossibilité, de traduire un modèle exprimé dans un langage dans un autre langage,

- une difficulté pour une entreprise d'utiliser un progiciel de modélisation d'entreprise si ce dernier est basé sur un langage différent de celui adopté par l'entreprise.

- il est difficile pour les utilisateurs de les comprendre vraiment et de choisir le plus adéquat à une situation donnée.

Ainsi, les concepts manipulés dans la modélisation d'entreprise proviennent d'un ensemble très vaste au sein duquel nous pouvons citer la gestion de production, les ressources humaines, l'économie, etc. Enfin, bien que basés sur des concepts communs issus de théories plus générales comme la systémique, la théorie des graphes ou de l'information, les langages sont interprétables de manières différentes lorsqu'ils sont mis concrètement en œuvre. D'une manière générale, les langages de modélisation d'entreprises semblent construits de composants semblables. Cependant, ces composants peuvent posséder des noms distincts et il est ainsi difficile de les comparer. En effet, ces langages possèdent donc des composants ayant des syntaxes et une sémantique différentes, qui souvent, ne sont pas définies formellement [Petit et al., 1997].

Ainsi, depuis quelques années, des initiatives apparaissent cherchant à identifier les besoins et à définir un cadre de développement [conférence internationale ICEIMT'97], [Petit *et al.*,

1997], [Panetto et al., 2001], [Vernadat, 1999b et 2001]. Les organismes de normalisation (CEN et ISO) participent également à ces initiatives [CEN ENV 40003, 1990], [CEN ENV 12204, 1995], [CEN ENV 13550, 1999], [CEN TC310/WG1, 2002], [Chen *et al.*, 2001], [Chen, 2002], [ISO FCD, 2000], [ISO IEC, 1999], [ISO TC154/WG1, 1999], [ISO TC184/SG5, 1999a, 1999b]. Cet ensemble d'initiatives est regroupé dans la littérature sous le vocable d'UEML (Unified Enterprise Modelling Language). Citons également les travaux plus récents de l'« Interest group on UEML » de la Task Force IFAC-IFIP [Vernadat, 1999b] ou ceux du groupe de travail « Modélisation d'entreprise » du Groupement de Recherche en Productique [GRP/GT5, 1999], [Roque *et al.*, 2002].
Toutes ces initiatives s'orientent aujourd'hui vers un consensus entre les différents acteurs et les différents organismes de normalisation travaillant dans le domaine de l'ingénierie et de l'intégration d'entreprise. Ce consensus devrait donc se matérialiser par le développement d'un langage pivot de haut niveau, permettant à plusieurs modeleurs d'échanger des informations, des données et des connaissances, contenues et déjà prises en compte au sein de leurs propres langages.

Il est alors admis que le développement d'un tel langage contribuera à :

- une définition claire de la sémantique commune des langages et une meilleure délimitation du domaine de la modélisation et de l'ingénierie d'entreprise,

- une meilleure interopérabilité et capacité de communication entre acteurs de la modélisation en environnement hétérogène,

- une meilleure définition du corpus scientifique de la modélisation et de l'ingénierie d'entreprise et donc une augmentation de la visibilité de ces dernières dans le cadre de la communauté scientifique,

- un vocabulaire accepté et utilisé par les organismes de normalisation à tous les niveaux (national, européen et international), travaillant dans le domaine [Chen *et al.*, 2000].

1.4.2 Evolution du contexte industriel et modélisation d'entreprise [Vallespir, 2003]

Afin, de justifier davantage les travaux de recherche sur l'élaboration d'un langage unifié de modélisation d'entreprise, il est nécessaire de regarder l'évolution du contexte industriel actuel. Ainsi, nous allons montrer que la modélisation d'entreprise sera utile dans le futur et donc que ces travaux de recherche, concernant notre langage unifié, seront utiles par la suite. La définition des objectifs de la modélisation d'entreprise est certainement la question fondamentale qui va guider les évolutions futures.

Au fil des années, le contexte industriel a suffisamment changé pour se poser la question de l'utilisation de la modélisation d'entreprise aujourd'hui. Pour cela, nous évoquerons deux points. Le premier concerne le contenu des projets industriels et le deuxième la manière de mener ces projets.

1.4.2.1 Le contenu des projets industriels.

En effet, la modélisation d'entreprise est apparue à une époque où il n'existait encore aucun support aux projets d'organisation ou de réorganisation des systèmes industriels. Il faut noter également que ces projets d'organisation participaient bien souvent en fait à des projets d'informatisation. En effet, au départ le modèle d'entreprise devait aider au développement d'applications informatiques dédiées (avec un passage du modèle au programme le plus aisé possible voire automatique : notion d'exécutabilité du modèle). Aujourd'hui, il devrait pouvoir aider à la paramétrisation d'un ERP ou même à sa personnalisation en fonction des contraintes de l'entreprise.

D'autre part, sont apparus depuis d'autres exigences de la part du marché auxquelles le milieu industriel a répondu par les démarches qualité. Il est intéressant de voir que l'écriture de procédures qualité est conceptuellement similaire à l'informatisation puisque les deux ont vocation à figer un comportement et à rendre celui-ci reproductible dans une certaine mesure. Ainsi, le rôle que peut jouer la modélisation d'entreprise par rapport à l'informatisation peut être également joué par rapport à la définition et l'écriture de procédures qualité.

En conclusion, il est aujourd'hui nécessaire de revoir la notion d'exécutabilité pour généraliser cette notion à la génération de toute procédure (informatique ou manuelle) à partir d'un modèle (Figure 1.4-1).

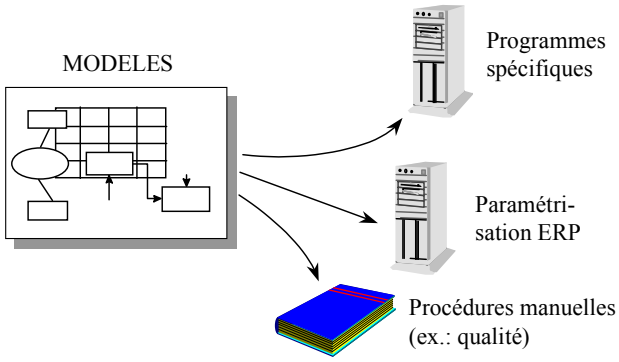


Figure 1.4-1 : Exécutabilité généralisée : génération de procédures à partir d'un modèle

Ainsi, même en considérant que cette évolution est un enjeu important, le développement d'une activité sur ce thème n'est actuellement pas vraiment intéressante, si ce n'est à moyen terme. En effet, ce type de développement serait aujourd'hui particulièrement entravé par la dispersion des langages. La partie opérationnelle de ces travaux serait en effet à multiplier par autant de fois qu'il y a de langages. Aussi, nous pensons que des travaux de recherche sur ce sujet ne sont raisonnablement envisageables que sur la base d'un langage standard, tel que notre langage unifié de modélisation d'entreprise.

1.4.2.2 Le déroulement des projets industriels.

Par ailleurs, la modélisation a été conçue, au départ, pour être un support à des projets. Les projets étaient envisagés il y a quelques décennies comme étant exceptionnels et d'ampleur conséquente. Cette façon de voir correspondait à un environnement économique stable. Ce n'est plus le cas aujourd'hui. En effet, le contexte est suffisamment mouvant aujourd'hui pour nécessiter de passer de grands projets pouvant être amortis sur de longues périodes à des projets moins ambitieux, plus rapidement amortis mais plus fréquents. On passe ainsi de la notion de projet de reconception à celle de gestion de l'évolution.

Qu'en est-il de la modélisation d'entreprise dans ce cadre ?

Lorsque les projets étaient notablement espacés dans le temps, les acteurs de l'entreprise impliqués dans le projet se posaient des questions de représentation et d'évaluation

uniquement à ces moments-là. La modélisation d'entreprise, dans l'hypothèse où elle était utilisée, était redécouverte à chaque expérience.

Si maintenant les projets se rapprochent dans le temps, si même l'entreprise est perpétuellement en démarche d'évolution, l'activité de représentation et d'évaluation et, partant, l'utilisation de la modélisation d'entreprise devient permanente. Ces activités et cette utilisation sortent de l'exceptionnel pour devenir récurrentes et intégrer les fonctions normales de gestion.

La conséquence immédiate pour l'entreprise est le besoin de s'approprier les outils qui lui permettront de s'auto-modéliser et de s'auto-évaluer en permanence (Figure 1.4-2).

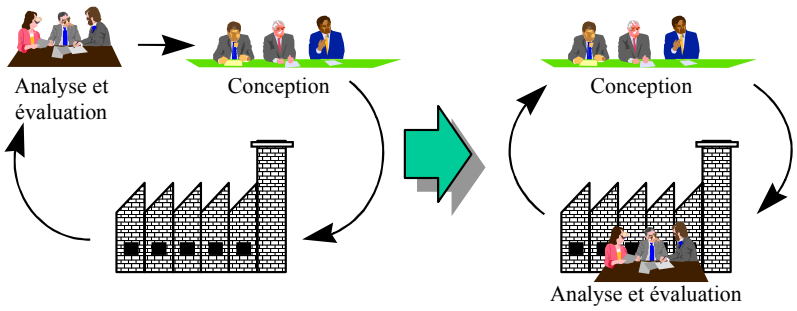


Figure 1.4-2 : Internalisation des fonctions d'analyse et d'évaluation

Aussi, une perspective d'évolution importante concernant la modélisation d'entreprise est l'appropriation de cette dernière par les entreprises elles-mêmes. Ainsi, si la modélisation est aujourd'hui adoptée en phase de conception des systèmes, il lui reste à s'imposer en phase d'exploitation.

1.5 Conclusion

La tendance actuelle dans les approches de modélisation d'entreprise cherche à combiner, de manière plus ou moins intégrée, différents langages de base, afin de créer plusieurs vues d'un même modèle (par exemple CIMOSA, GRAI/GIM, ARIS, …). Une telle approche combinée permet de tirer avantage des bénéfices de chaque langage de base et offre à l'analyste les moyens de représenter de manière plus précise et plus complète ses besoins face aux objectifs

de son modèle. Des modèles décrits avec des langages de base complètement distincts doivent, d'une manière ou d'une autre, être corrélés. Pour ce faire, les langages, eux-mêmes, doivent pouvoir être intégrés, ou être « interopérables ». Cependant, actuellement, l'intégration de plusieurs langages de base est, dans la plupart des cas, mise en oeuvre dans un même outil logiciel support (c'est-à-dire par une approche propriétaire ayant des recouvrements non nul avec d'autres approches). Aucun outil (au moins dans le domaine de la modélisation d'entreprise) ne met en œuvre actuellement une interopérabilité de ses propres modèles avec des modèles créés à l'aide d'autres langages supportés par d'autres outils.

Ainsi, dans le premier chapitre de cette thèse, nous avons dégagé la problématique concernant la définition et l'élaboration d'un langage unifié de modélisation d'entreprise. Nous avons montré qu'au cours de ces dernières années beaucoup de langages de modélisation d'entreprise sont apparus. Tous ces langages ont été développés afin de traiter une problématique particulière, dans un domaine précis. Ils ont donc tous une nécessité et une raison d'exister. Cependant, le nombre important de langages est un peu déroutant pour l'analyste. Il devient en effet très difficile de choisir le bon langage dédié au problème à résoudre. De plus, l'analyste ne peut pas être un spécialiste de tous ces langages de modélisation d'entreprise. Par ailleurs, d'une manière générale, ils semblent composés des concepts semblables mais avec des noms distincts, il est ainsi difficile de les comparer car les langages ont des syntaxes et une sémantique différentes, qui souvent ne sont pas définies formellement. Le développement d'un langage unifié de modélisation d'entreprise devient donc une nécessité. Ce langage servant de langage « pivot », permettra entre autre, à l'analyste de comprendre un modèle élaboré avec un langage qu'il ne maîtrise pas.

Nous allons donc dans le chapitre suivant, faire un tour d'horizon des différents travaux de recherches qui au fil des années, ont tenté d'une manière ou d'une autre d'intégrer des langages.

Chapitre II

Etat de l'art

Chapitre II
Etat de l'art

2.1 Introduction

Nous venons de voir qu'un grand nombre de langages de modélisation d'entreprise ont été proposés lors des dernières décennies. Aussi, est rapidement apparu le besoin de rapprocher, fédérer voire intégrer ces outils pour obtenir des ensembles méthodologiques plus puissants et cohérents. Suite à la phase d'émergence désordonnée d'outils, nous pouvons identifier trois grands stades dans le cadre du processus de rapprochement des langages, comme souligné dans [Vallespir, 2003]. En effet, la recherche en modélisation d'entreprise a donné naissance à un ensemble d'outils répondant à une problématique spécifique à chaque stade. On peut ainsi, dégager trois grandes étapes dans l'évolution des outils de modélisation d'entreprise, qui sont des réponses à une évolution de la problématique que l'on a voulu traiter au fil des années. On peut ainsi identifier les trois stades suivants :

- premier stade : la fédération et l'interopérabilité des langages - les cadres de modélisation,
- deuxième stade : l'intégration conceptuelle (tel que GERAM),
- troisième stade : l'intégration microscopique (tel que UEML)

La Figure 2.1-1 illustre ces différents stades, en classant les différents langages que nous allons considérer dans cet état de l'art.

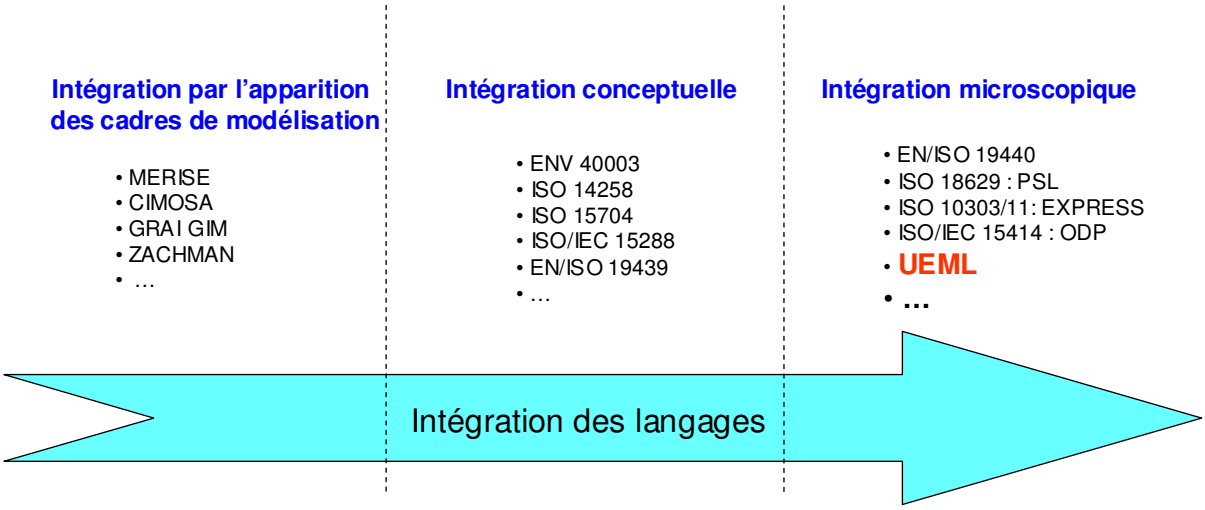


Figure 2.1-1 : Intégration des langages

Ainsi, nous allons dans une première partie de ce chapitre, détailler ces stades en exposant les besoins qui ont nécessité de créer de nouveaux langages et quelques résultats qui ont été obtenus.

Dans une deuxième partie, nous présenterons une architecture qui a été développée dans le cadre du projet UEML [UEML IST - 2001 – 34229], auquel nous avons participé. Nous utiliserons cette architecture afin de pouvoir positionner les langages, dans le but de montrer qu'aucun des langages de modélisation d'entreprise actuels ne peut être promu au rang de langage unifié de modélisation d'entreprise.

Enfin, dans une dernière partie, nous aborderons le sujet de l'intégration des bases de données informatiques. En effet, cette intégration pose une problématique quelque peu similaire à la nôtre.

2.2 Stades dans l'évolution des outils de modélisation d'entreprise

2.2.1 Premier stade de rapprochement : la fédération et l'interopérabilité des langages - les cadres de modélisation

Nous avons vu dans le chapitre précédent qu'une entreprise est un système complexe. Ainsi, rapidement les langages de modélisation d'entreprise, ou plus généralement les méthodes de modélisation d'entreprise ont dû permettre de représenter l'entreprise selon plusieurs points de vue. Ainsi, ces langages et méthodes de modélisation permettent de couvrir l'ensemble des besoins d'une entreprise en tenant compte par exemple :

- des divers **objectifs** de la modélisation (analyse des performances du système, évolution d'un système existant, conception d'un nouveau système),
- des **points de vue** considérés (système physique, système de conduite),
- des **niveaux d'analyse** ou de remise en cause (niveau fonctionnel ou sémantique, niveau des composants supportant ces fonctions),
- des **niveaux de détail** (analyse du système comme un tout ou focalisation sur un point précis).

Cependant, il est également important de pouvoir représenter l'entreprise en fonction de certains types de concepts de modélisation tels que :

- les **activités** (de toute nature : transformation physique, traitement d'information, stockage, transport, etc.) qui permettent de représenter les fonctionnalités du système, elles-mêmes découlant de la finalité (raison d'être) du système. La représentation des flux (quelque soit leur nature : physique, informationnelle, financière) permet de comprendre l'activité comme un système consommateur et générateur. L'ensemble permet la *représentation fonctionnelle* du système.

- **les événements** afin de traduire la dépendance temporelle du système avec son environnement. La connaissance de la durée des activités vient compléter ce concept et permet la *représentation dynamique* du comportement du système.

- **les ressources** (techniques ou humaines) pour représenter le lien entre organe et activité. Elles permettent une *représentation organique* du système.

Aussi, face à ce constat, lorsqu'une approche méthodologique a vocation à posséder une large couverture de modélisation, elle utilise plusieurs langages. Le problème de la cohérence des différents modèles est alors posé. Une solution est de proposer un cadre de modélisation qui explicite le positionnement relatif des modèles, et les zones de recouvrement.

Ainsi, les cadres de modélisation correspondent sans conteste à un premier stade d'intégration des langages de modélisation.

2.2.1.1 MERISE

La première approche ayant posé clairement ce concept est certainement MERISE [Tardieu *et al.*, 1983] qui, en proposant trois niveaux d'abstraction et deux vues, en arrive à un cadre de modélisation à six domaines de modélisation (Figure 2.2-1).

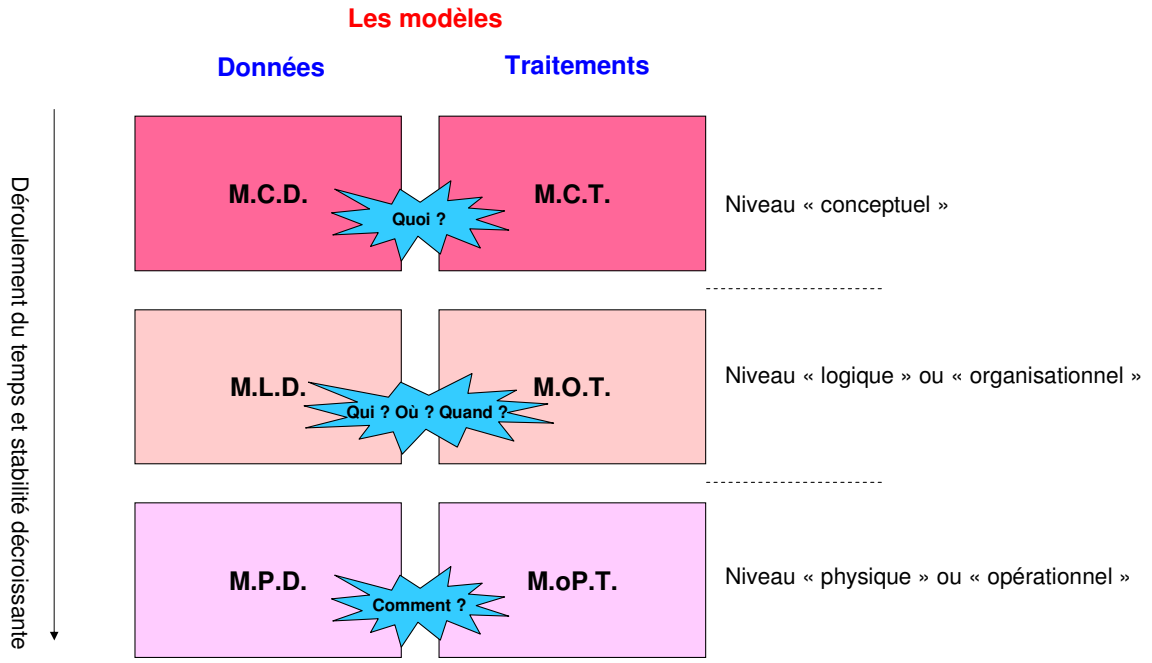


Figure 2.2-1 : Cadre de modélisation de Merise

Chacun de ces domaines utilise un langages (pour les niveaux les plus abstraits) ou plusieurs dans le cas de niveaux les moins abstraits.

- Le **« niveau conceptuel** » (le « quoi ? »), aboutissant aux M.C.D. (Modèle conceptuel des données) et M.C.T. (Modèle conceptuel des traitements). A ce stade, données et traitements sont étudiés de manière parallèle, dissociée (en principe par des équipes différentes),

- le « **niveau logique** » pour les données et le « niveau organisationnel » pour les traitements (le « Qui ? », le « Quand ? », le « Où ? ») correspondent aux M.O.T. (Modèle Organisationnel des Traitements) et M.L.D. (Modèle Logique des Données). En fait, l'équipe qui a travaillé sur le M.C.T. passe à un degré de détails beaucoup plus fouillé des traitements (le M.O.T.), donnant, à son tour, une vision « en puzzle » de l'organisation des données. Cette vision est alors confrontée au schéma global qu'est le M.C.D. et de ce rapprochement (parfois houleux), résulte une version corrigée de l'organisation des données, permettant de la finaliser sous la forme du M.L.D.

- enfin, le « **niveau physique** » pour les données aboutissant à la création des tables, et le « niveau opérationnel » pour les traitements (le « comment ? ») enclenchent une analyse détaillée de chaque traitement et des développements. Ce niveau, dépendant étroitement de chaque contexte concret, ne fait en principe pas l'objet d'exposé dans les ouvrages consacrés à Merise.

2.2.1.2 CIM-OSA

D'autres méthodes ont suivi la même voie dans le domaine de la modélisation d'entreprise.
Elles proposent des vues différentes en nature et en nombre, des niveaux différents, mais le concept sous-jacent de cadre de modélisation reste le même. Le cube de CIM-OSA (CIM Open System Architecture) [AMICE, 1993] (Figure 2.2-2) est certainement le cadre de modélisation le plus médiatisé. CIM-OSA est une architecture permettant de construire et analyser des systèmes intégrés de production. Elle a été développée par le Consortium AMICE dans le cadre de projets ESPRIT.

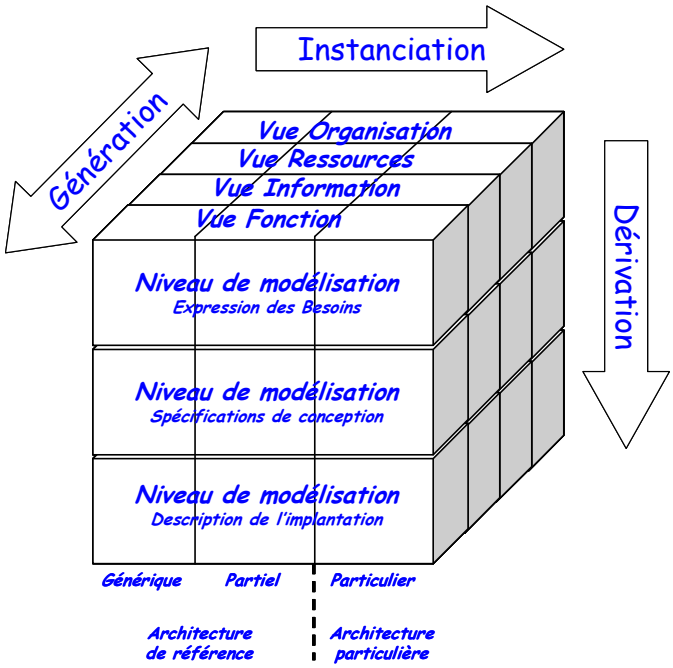


Figure 2.2-2 : Le cadre de modélisation de CIM-OSA

Le cadre de modélisation formalise trois principes fondamentaux et orthogonaux pour la modélisation en entreprise suivant une structure à trois axes.

Les trois axes retenus sont les suivants :

- L'axe de **généricité** se compose de trois niveaux : un niveau générique, où sont définis les primitives de base du langage de modélisation, un niveau partiel contenant des modèles partiels, c'est à dire des structures prédéfinies et réutilisables pour un domaine d'application donné, et un niveau particulier correspondant aux modèles spécifiques de l'entreprise. Les niveaux génériques et partiels constituent l'architecture de référence de CIMOSA alors que le niveau particulier correspond à l'architecture particulière d'une entreprise donnée (développée pour des besoins particuliers).

- L'axe de **dérivation** (relatif au cycle de vie du système) préconise de modéliser l'objet d'étude suivant trois niveaux de modélisation : un niveau de définition des besoins permettant l'écriture du cahier des charges dans le langage de l'utilisateur final, un

niveau des spécifications de conception permettant de spécifier et analyser dans le détail des solutions répondant aux besoins exprimés et un niveau de description de l'implantation (ou implémentation) permettant de décrire précisément l'implantation de la solution retenue.

- L'axe de **génération** définit quatre vues essentielles de modélisation permettant d'accéder au modèle, en se focalisant sur certains aspects et en négligeant les autres (ceci permet de gérer la complexité du modèle). D'autres vues peuvent être définies si nécessaires.

Les vues retenues dans CIMOSA sont :

- La **vue fonction** qui sert à décrire la fonctionnalité et le comportement de l'entreprise en termes de processus, d'activités et d'opérations, soit ce qu'il y a faire à différents niveaux de détail.

- La **vue information** qui sert à décrire les objets de l'entreprise, leurs relations et leurs différents états possibles, c'est à dire quels sont les objets traités et comment ils sont gérés.

- La **vue des ressources** qui sert à décrire les moyens nécessaires à mettre en oeuvre pour réaliser les fonctions de l'entreprise, leur rôle et leur mode de gestion, c'est à dire qui fait quoi, quand et comment.

- La **vue organisation** qui sert à décrire la distribution des responsabilités et des autorités dans les prises de décision, c'est à dire qui est responsable de quoi.

2.2.1.3 GRAI - GIM

De même, la méthode GIM (GRAI Integrated Methodology) [Doumeingts et al., 1993] est également basée sur ce principe. Son cadre de modélisation (Figure 2.2-3) assure la cohérence des formalismes mis en œuvre en s'appuyant sur deux axes :

- un axe « *vues* » qui permet de prendre en compte plusieurs points de vue issus d'une décomposition systémique : vue décisionnelle, vue informationnelle, vue physique auxquelles s'ajoute une vue fonctionnelle permettant de présenter un modèle simple du système étudié et les relations de ce dernier avec son environnement,

- un axe « *niveau d'abstraction* » s'appuyant sur une approche multi-strates et permettant la modélisation du système en fonction de plusieurs sémantiques compte tenu de la profondeur d'analyse que l'on désire avoir (niveau conceptuel : pas de prise en compte des options d'organisation ou techniques, niveau structurel : prise en compte des options d'organisation, niveau réalisationnel : prise en compte des options techniques),

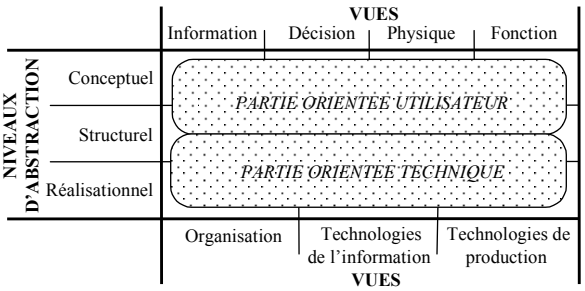


Figure 2.2-3 : Le cadre de modélisation de GIM

GIM fait apparaître un concept original par rapport à d'autres méthodes en proposant une partie orientée utilisateur et une partie orientée technique. La première doit permettre d'intégrer l'ensemble des besoins exprimés par les utilisateurs et de déboucher sur des spécifications utilisateurs exprimés en terme de modèles décisionnels, informationnels, physiques et fonctionnels. Partant de ces spécifications utilisateurs, la partie orientée technique a pour vocation de représenter les meilleurs moyens d'implantation pour traduire les besoins utilisateurs, ces choix devant être exprimés en terme de technologie de fabrication, technologie de l'information et règles ou procédures d'organisation.

2.2.1.4 ZACHMAN

Le premier cadre de modélisation d'architecture de Système d'Information d'entreprise basé sur l'approche « Données/Traitements » fut proposé par Zachman (Figure 2.2-4) au début des

années 1980 [Zachman, 1987]. Il s'agit d'un cadre unifié d'architecture qui s'étend de la modélisation métier à l'architecture de production en passant par le développement logiciel.

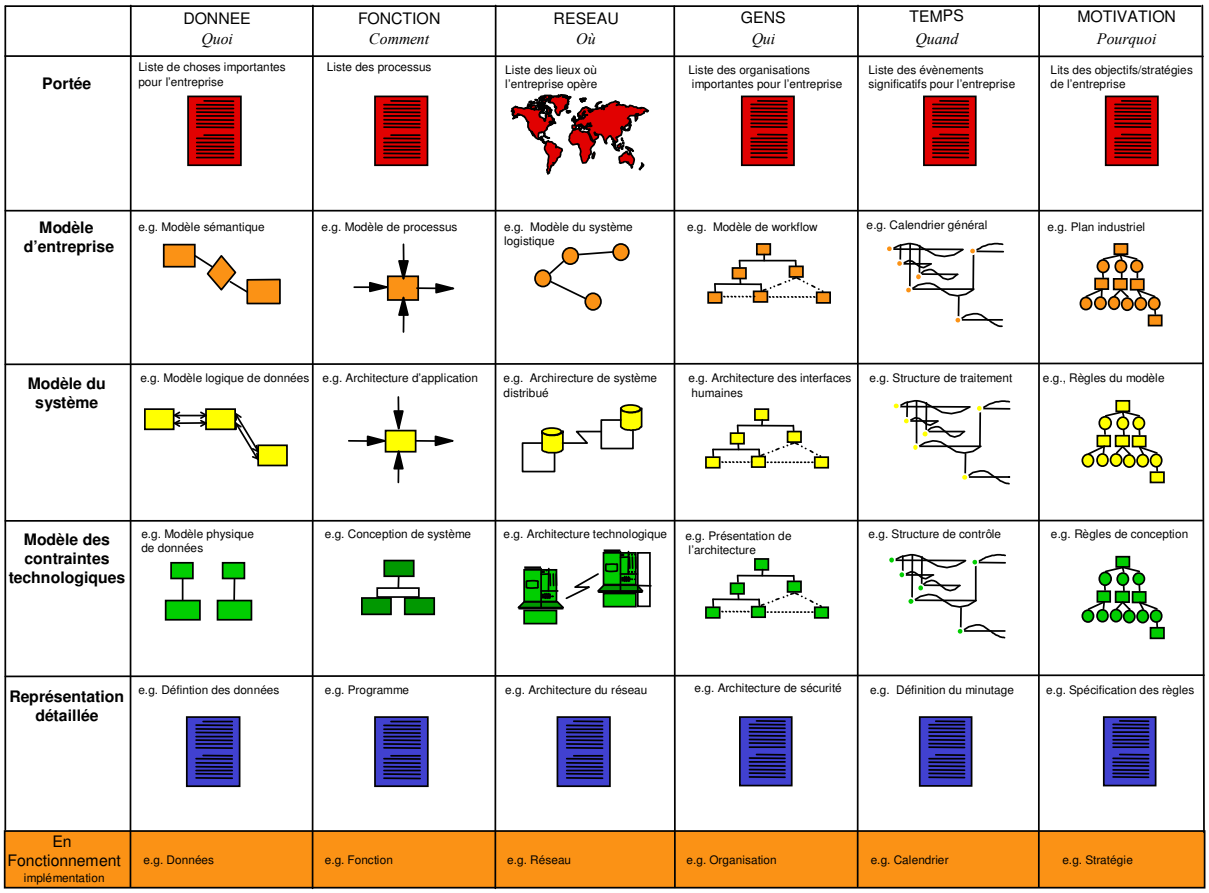


Figure 2.2-4 : Cadre de modélisation de Zachman

Ce cadre de modélisation se représente de façon matricielle en six colonnes et cinq lignes.

- Les colonnes représentent les aspects de l'entreprise : Quoi (que traite-t-on), Comment (comment traite-t-on), Où (aspects géographiques), Qui, Quand, Comment (aspects organisationnels),

- les lignes représentent les vues selon les différents acteurs de l'entreprise : les enjeux stratégiques, puis le modèle métier, le modèle fonctionnel SI, les modèles conceptuels et physiques d'architecture…

A chaque cellule de la matrice, sont associés des documents formalisés d'architecture; par exemple, le langage Entités/Associations sera utilisé pour le « Modèle Conceptuel de Données» de la cellule « Données/Conception ».

2.2.1.5 Conclusion

Ainsi, les cadres de modélisation correspondent bien à un premier stade d'intégration des langages de modélisation. Cependant, il est important de noter tout de même que bien que ces cadres de modélisation soient effectivement un premier pas vers l'intégration des langages, il existe souvent un problème de cohérence entre les différents langages. En effet, les intersections entre eux ne sont pas forcément explicitement définies. Nous en reparlerons dans le chapitre 4 de ce mémoire.

2.2.2 Deuxième stade de rapprochement : l'intégration conceptuelle - Normalisation de la modélisation d'entreprise - Architectures de référence

Le domaine de la normalisation en modélisation d'entreprise a donné au fil des années un ensemble d'architectures qu'il nous semble intéressant d'étudier. La Figure 2.2-5 représentent l'évolution des architectures de référence de la recherche en normalisation sur la modélisation d'entreprise [Chen *et al.*, 2004].

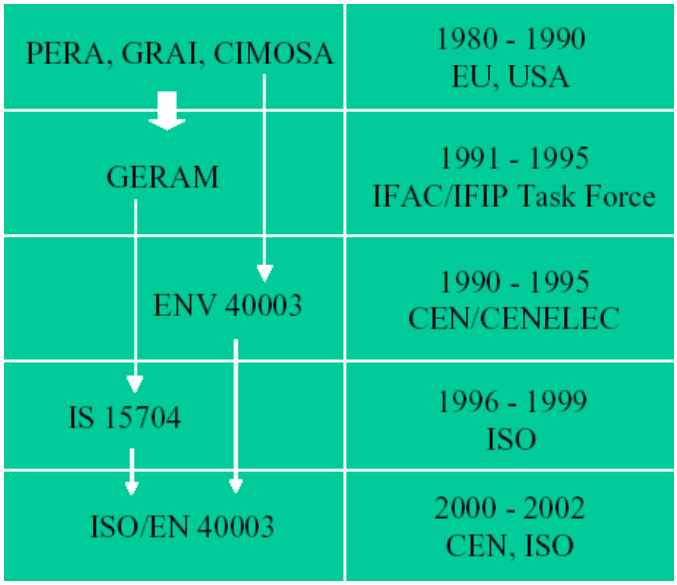


Figure 2.2-5 : Architecture en modélisation d'entreprise

Parmi les normes, l'ENV 40003 [CEN ENV 40003, 1990] et l'ISO 15704 [ISO 15704, 1998] sont plus structurées que les deux autres. La couverture de modélisation de l'ISO 15704 est plus large que celle de l'ENV 40003, mais l'ENV 40003 est l'initiative la plus ancienne dans ce domaine. Elle fut tout d'abord développée comme une pré-norme par le CEN/CENELEC au début des années 1990. Plus tard, l'ISO 14258 [ISO 14258, 1999] a été influencée par l'ENV 40003 et fait référence à un plus haut niveau d'abstraction que l'ENV 40003. Ainsi, nous allons considérer dans cette partie les normes suivantes :

- ISO 14258 [ISO 14258, 1999] : Concepts et règles pour les modèles d'entreprises.

- ISO/IEC 15288 [ISO/IEC CD 15288, 1999] : Cycle de vie du système de pilotage / Cycle de vie des processus.

- EN/ISO 19439 [ISO/CEN 19439, 2002] : Intégration, cadre de modélisation pour la modélisation d'entreprise.

2.2.2.1 L'ISO 14528

L'ISO 14528 (1999) vise à définir des concepts et des règles pour les modèles d'entreprise afin de guider et contraindre d'autres normes ou implémentations qui existent ou existeraient sur le sujet. Elle ne définit pas de normes concernant les processus d'entreprise, l'organisation ou les données, mais établit une base sur laquelle des normes en modélisation d'entreprise sont susceptibles d'être développées. Son élaboration a été influencée par l'initiative européenne ENV 40003.

Les concepts et les règles principaux définis dans l'ISO 14258 sont :

- les éléments à employer, en produisant un modèle d'entreprise, incluant la théorie des systèmes, les méthodologies dérivées de la théorie de systèmes, les facteurs de la production (tels que les personnes, le capital, les matériaux, l'information, l'énergie et les outils), la couverture de modélisation, la sémantique et la syntaxe, etc.,

- des concepts concernant les différentes phases de cycle de vie,

- la hiérarchie,
- la structure,
- le comportement,
- les vues du modèle.

Le Tableau 2.2-1 montre la relation entre des phases communes du cycle de vie et les activités « quoi », « comment » et « faire ».

	"What" Activities	"How" Activities	"Do" Activities
Plan and Build Phase (e.g. before sell/buy title transfer)	Develop goals Define strategy Define product needs	Develop requirements Define concept Design product Plan to produce product Plan to support product	Produce parts Produce product Test product Ship product
Use and operate Phase (e.g. after the sell/buy transfer)	Define support needs Define use	Define use requirements Define support requirements	Use the product Support product
Dispose and Recycle phase (e.g. after product is no longer useful)	Define recycle/dispose needs	Define recycle / dispose requirements	Recycle product Dispose product

Tableau 2.2-1 : Relation entre des phases communes du cycle de vie et les activités « quoi », « comment » et « faire ».

L'ISO 14258 pourrait être la norme de plus haut niveau sur laquelle pourrait se baser les autres normes. Le cadre de modélisation lui-même a été défini dans l'ISO 15704 et l'ENV 40003. Les concepts définis dans le tableau précédent sont aussi complémentaires à l'ENV 40003 et l'ISO 15704. L'axe du cycle de vie est raffiné, en particulier les phases Planifier et Construire sont détaillées dans l'ISO 15704. Les activités « Quoi » et « Comment » sont partiellement représentées en terme de concepts de modélisation le long de l'axe des vues dans l'ISO 15704 et l'ENV 40003.

2.2.2.2 L'ISO 15704

Dans le milieu des années 1990, avec le support de l'IFAC-IFIP Task Force sur l'intégration d'entreprise, une pré-norme a vu le jour avec une couverture de modélisation plus importante et plus générique que la précédente version de l'ENV 40003. Le résultat fut l'ISO 15704. Cette norme fut développée sur les bases de GERAM (Generalised Enterprise Reference Architecture and Methodology) [GERAM, 1999] avec la contribution significative de CIMOSA [AMICE, 1993], GIM [Doumeingts et al., 1993] et PERA (Purdue Enterprise Reference Architecture) [Williams, 1996]. PERA a été développée aux Etats-Unis par l'université de Purdue.

GERAM propose une organisation des principaux concepts constituant les approches existantes. In fine, elle s'articule autour d'un ensemble de macro-composants (langages, démarches, modèles de références, etc.) permettant de couvrir les domaines relevant de l'ingénierie et de l'intégration en entreprise. La plupart de ces composants s'appuient donc sur des travaux déjà largement reconnus en modélisation d'entreprise. La Figure 2.2-6 présente l'ensemble des composants de GERAM, publiés dans l'ISO 15704, et les liens qui les unissent.

Par exemple, GERA (Generic Enterprise Reference Architecture) fournit un cadre d'analyse et de modélisation dont la structure s'est largement inspirée de CIMOSA et PERA. Par ailleurs, EEM (Enterprise Engineering Methodology) fait référence à des méthodologies de modélisation telles que GIM ou PERA. Ces dernières s'appuient elles-mêmes sur des langages parfois supportés par des outils informatiques rassemblés dans EMT/EML (Enterprise Modeling Tools / Languages). IDEF, IDEF3, CIMOSA et la grille GRAI d'une part et ARIS [Scheer, 2002], FirstSTEP, MOOGO (IEM) d'autre part, figurent parmi les langages et outils pouvant être retenus.

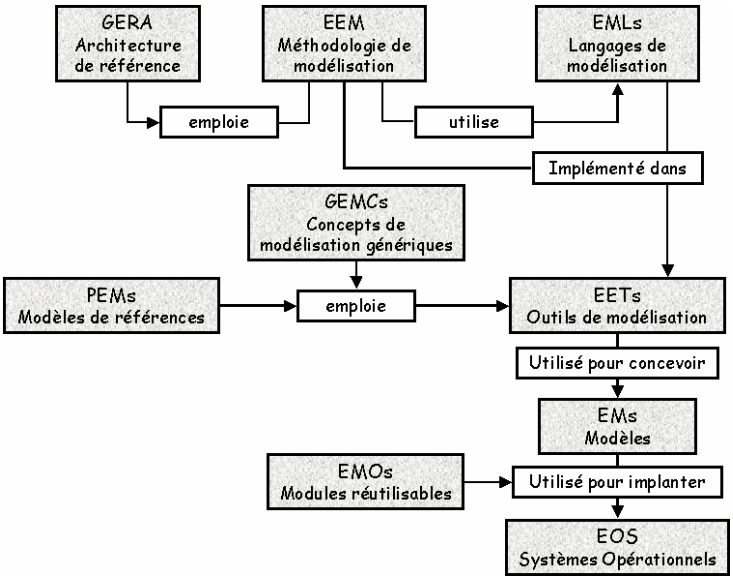


Figure 2.2-6 : Les composants de GERAM comme publiés dans l'ISO 15704

Les concepts sont formalisés sous forme d'ontologies, de méta-modèles ou de glossaires dans GEMC (Generic Enterprise Modelling Concepts). Les modèles sont, si possibles, agrégés en PEMs (Partial Enterprise Models). Enfin EMOs (Enterprise Modules) et EMs (Enterprise Models) concourent à la représentation de tout ou partie de l'entreprise de façon à guider l'implantation des systèmes opérationnels de l'entreprise (EOS : Enterprise Operating Systems) mais aussi pour en estimer les performances actuelles et futures.

2.2.2.3 EN/ISO 19439

L'ISO/CEN 19439 [ISO/CEN 19439, 2002] a été élaborée conjointement par l'ISO TC184 TC5/WG1 et le CEN TC310 WG1, et fut approuvée en 2002. Cette norme est basée sur des travaux plus anciens développés au début des années 1990 par le CEN TC310/WG1, l'ENV 40003.

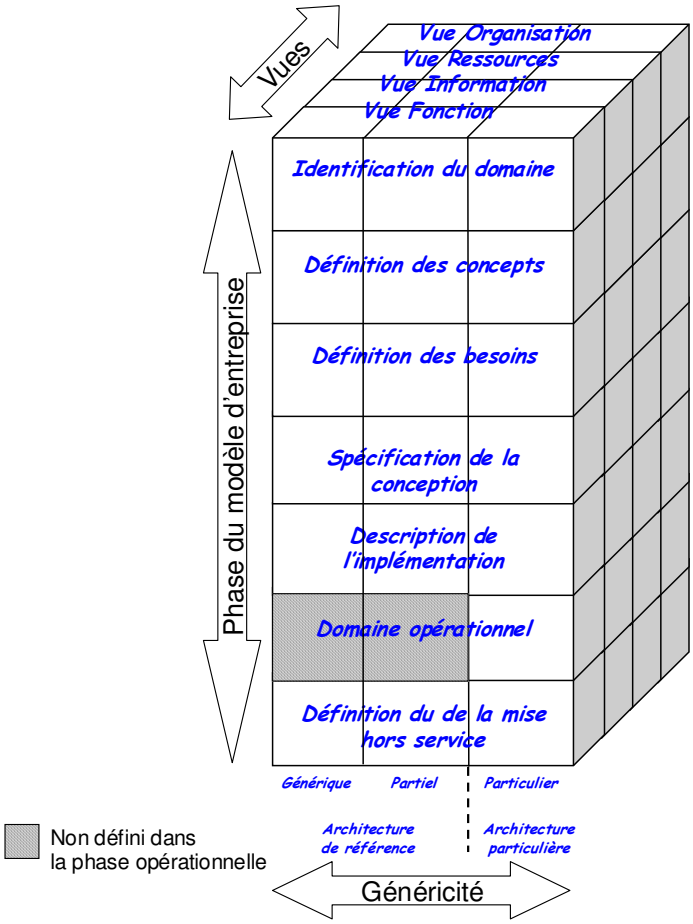


Figure 2.2-7 : Cadre de modélisation de l'ISO/CEN 19439

Elle définit des concepts génériques qui sont nécessaires pour permettre l'élaboration de modèles d'entreprise pour l'industrie et constituer une base conceptuelle pour des modèles basés sur l'ingénierie d'entreprise. Le cadre de modélisation de l'ISO/CEN 19439 possède la même dimension que celui de l'ENV 40003 (le cadre de modélisation de l'ENV 40003 est principalement basée sur celui de CIMOSA). Ainsi, la norme ISO/CEN 19439 est structurée suivant trois dimensions : Phase du modèle d'entreprise, vues du modèle d'entreprise et généricité (Figure 2.2-7).

La dimension « phase du modèle d'entreprise » est composée de :

- Identification du domaine,
- définition des concepts,

- définition des besoins,
- spécification de la conception,
- description de l'implémentation,
- domaine opérationnel,
- définition de la mise hors service.

Pour chaque phase quatre vues différentes sont définies :

- Fonction,
- information,
- ressource,
- organisation.

Trois niveaux de généricité permettent la construction de modèle d'entreprise :

- Le niveau particulier,
- le niveau partiel,
- le niveau générique.

Le niveau particulier utilise des modèles partiels définis dans le niveau partiel et/ou des éléments génériques fournis dans le niveau générique.

Cette « nouvelle » norme ENV 40003 est cohérente avec la norme ISO 15704 et est considérée comme une implémentation des besoins définis dans celle-ci. Le cadre de

modélisation est cohérent avec GERAM, qui est décrit en annexe de la norme ISO 15704. La principale modification en accord avec la pré-norme ENV 40003 est l'extension du cadre de modélisation par rapport à l'axe de phase des modèles d'entreprise, afin de couvrir des activités de plus haut niveaux telles que l'identification du domaine, la définition des concepts et la définition de la mise hors service.

2.2.2.4 Conclusion

Ainsi, ces normes, en s'inspirant des approches existantes, proposent un ensemble organisé de concepts pertinents de modélisation et d'intégration d'entreprise, ainsi que leurs relations. De ce fait, il est possible de déterminer le positionnement et les potentialités de toute approche par rapport à cet ensemble. Dans ce sens, elles participent clairement au rapprochement des méthodes existantes et peuvent être même considérées comme le premier pas d'une intégration puisqu'elles prennent les composants de méthodes existantes pour les rassembler dans un ensemble cohérent plus grand. Cependant, elles ne constituent qu'un cadre théorique de méta-modélisation, non-opérationnel en ne se penchant pas sur l'intégration des langages eux-mêmes.

2.2.3 Troisième stade de rapprochement : l'intégration microscopique - Normalisation des langages et UEML

Les travaux de recherche sur la normalisation ont également donné naissance à un ensemble de normes. Nous allons considérer les quatre suivantes :

- ISO/CEN 19440 [ISO/CEN 19440, 2002] : « Constructs » pour la modélisation d'entreprise,
- ISO/IEC 15414 [ISO/IEC CD15414, 2002] : Open Distributed Processing (ODP),
- ISO 10303-11 [ISO DIS 10303-11, 1992] : EXPRESS,
- ISO 18629-1 [ISO CD 18629-1, 2001] : Process Specifiaction Language (PSL).

Parmi eux, seulement le langage EXPRESS a été reconnu comme une norme internationale. Ce langage fut développé dans le cadre de la norme STEP [ISO 10303-1, 1993], afin de décrire des modèles d'informations dans le domaine technique, en vue de l'échange de données représentant de façon fiable et non ambiguë ces informations. A l'origine, ce langage n'était pas destiné à la modélisation d'entreprise [Chen et *al.*, 2004].

2.2.3.1 EN/ISO 19440 (2002)

L'EN/ISO 19440 définit un ensemble d'éléments orientés utilisateurs pour la modélisation d'entreprise. Elle est orientée vers la modélisation par processus. Par conséquent, les entités d'entreprise qui doivent être représentées sont des processus en prenant en compte, leur dynamique, leurs fonctionnalités, les entrées/sorties, les contrôles, les ressources, ainsi que les aspects organisationnels.

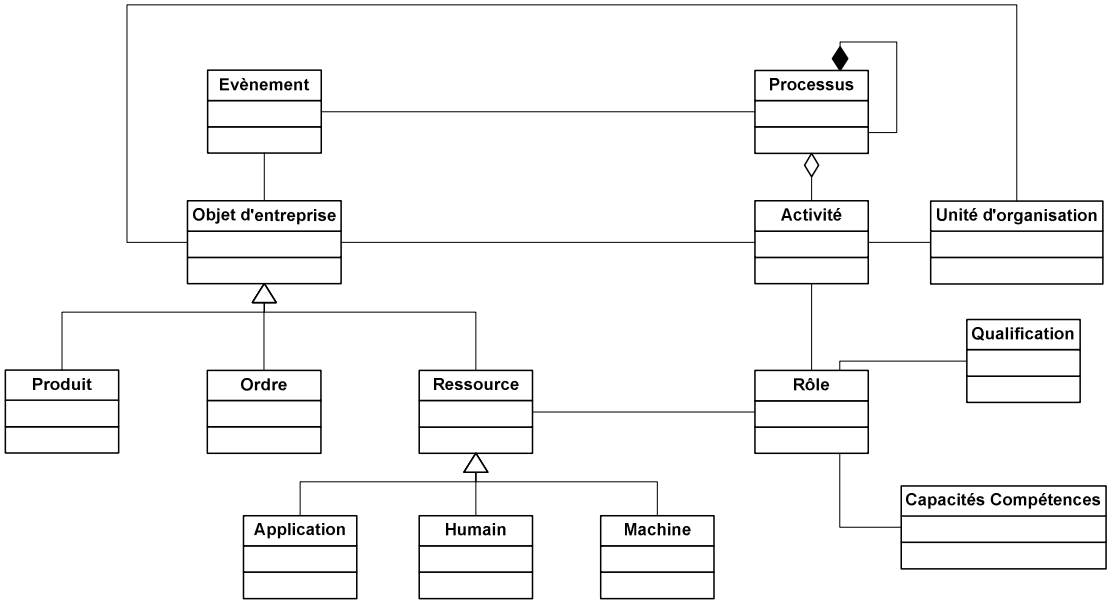


Figure 2.2-8 : ENV 10224 (uniquement pour illustration)

Le but de cette norme est d'aider l'utilisation du cadre de modélisation défini dans l'EN/ISO 19439. Cette norme est basée sur des travaux plus anciens, l'ENV 10224, qui fut adoptée comme une pré-norme européenne en 1995. La Figure 2.2-8 montre une vue d'ensemble des composants et relations tels qu'ils sont définis dans l'ENV 12204. La norme ENV 12204 est principalement basée sur l'approche CIMOSA et sur IEM [Spur *et al.*, 1996].

L'ensemble complet des éléments du langage de modélisation défini dans l'EN/ISO 19440 (sans les relations) est montré sur la Figure 2.2-9. On peut noter que beaucoup d'éléments de l'ENV 10224 se retrouve dans ce langage.

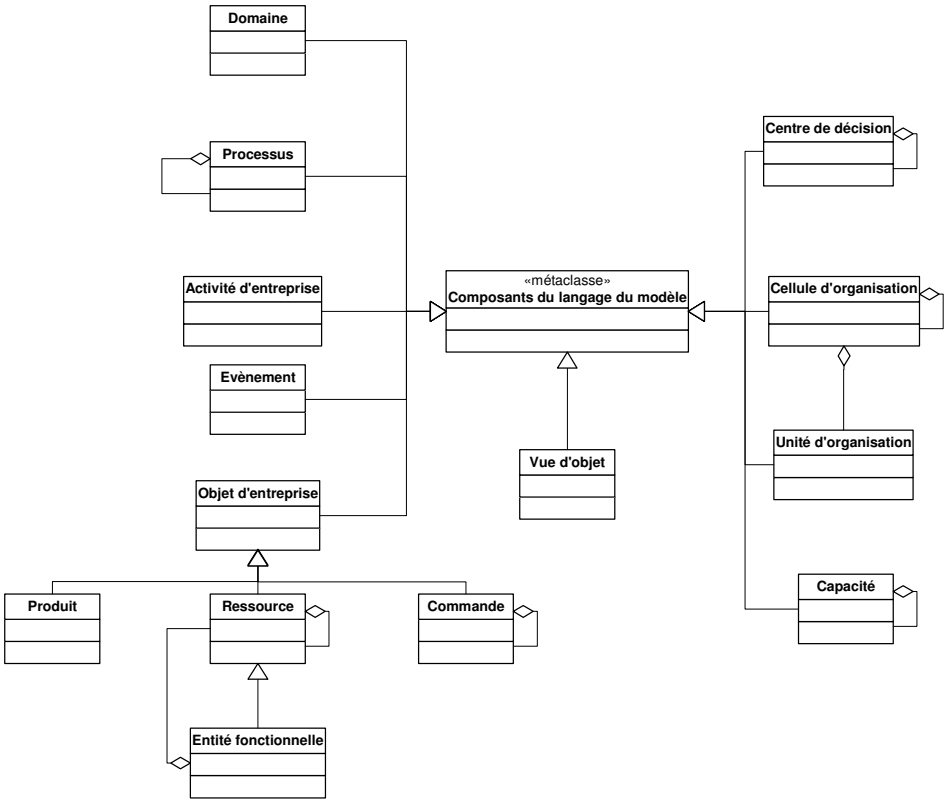


Figure 2.2-9 : Ensemble des éléments de l'EN/ISO 19440

2.2.3.2 ISO/IEC 15414 : Open Distributed Processing (ODP)

La Figure 2.2-10 présente une partie du méta-modèle concernant le point de vue de l'entreprise défini dans ODP. L'objectif principal d'ODP est d'identifier des rôles et des politiques des objets d'entreprise (ou des groupes d'objets) en termes de :

- Permission (ce qui peut être fait),
- prohibition (ce qui ne doit pas être fait),
- engagement (ce qui doit être fait),
- décisions de sécurité.

Le terme « objet » dans ODP signifie un « modèle d'entité ». Une entité est n'importe quel élément concret ou abstrait présentant un intérêt. Un objet est caractérisé par son comportement et de façon duale par son état. Selon le point de vue, l'accent peut être mis sur le comportement ou sur l'état. Quand l'accent est mis sur le comportement, on dit qu'un objet exécute des fonctions et offre des services.

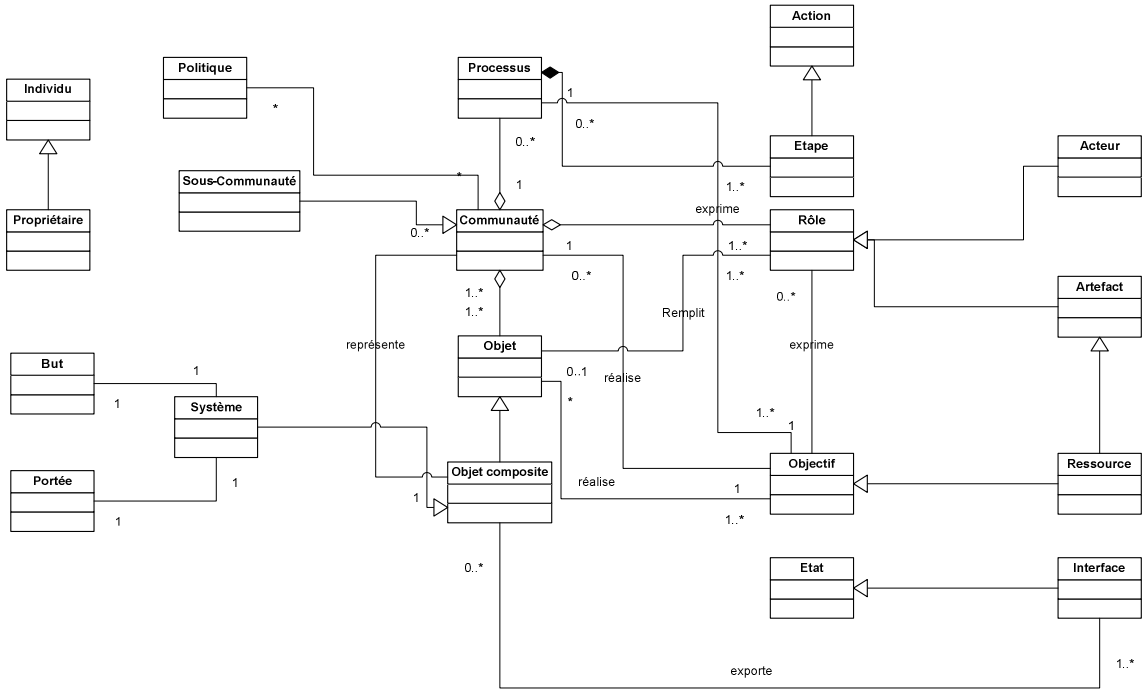


Figure 2.2-10 : Méta-modèle de l'ISO/IEC 15414

L'ISO/IEC 15414 est presque similaire à l'EN/ISO 19440 dans le sens où toutes les deux identifient des concepts d'entreprise de niveau élevé et cela sur des vues multiples. Cependant, elles ne possèdent pas les mêmes points de vues, ni la même couverture de modélisation. L'EN/ISO 19440 est une approche orientée processus ayant pour but de produire un modèle de processus opérationnel de supervision et de pilotage. Le langage ODP est focalisé sur des concepts tel que le but, la couverture et les politiques d'un système informatisable. Toutefois, on peut noter que langage ODP vient davantage en tant que complément plutôt qu'en contradiction.

2.2.3.3 EXPRESS

EXPRESS est le langage de modélisation de données conçu dans le cadre du projet STEP. Son objectif principal est la description de modèles d'informations dans le domaine technique, en vue de l'échange de données représentant de façon fiable et non ambiguë ces informations.

Dans le langage EXPRESS, l'accent principal est mis sur la précision du modèle et tout particulièrement sur les contraintes que doivent respecter les données pour être acceptées comme conforme au modèle, ceci pour assurer la fiabilité de l'information représentée. Cependant, bien qu'EXPRESS soit un langage « dit » formel, il permet seulement la spécification de schéma conceptuel de données. Il est basé sur le langage entité-relation étendu et est complémentaire à IDEF1x. EXPRESS a été développé pour la spécification de données produit mais peut néanmoins être également utilisé comme un outil pour la spécification de n'importe quel type de modèle conceptuel. Il ne possède cependant pas une couverture de modélisation suffisante pour pouvoir être utilisé comme un langage unifié de modélisation d'entreprise.

2.2.3.4 PSL - Le besoin de sémantique

Les approches existantes de modélisation par processus souffrent d'un manque de définition de la sémantique de la terminologie des processus.

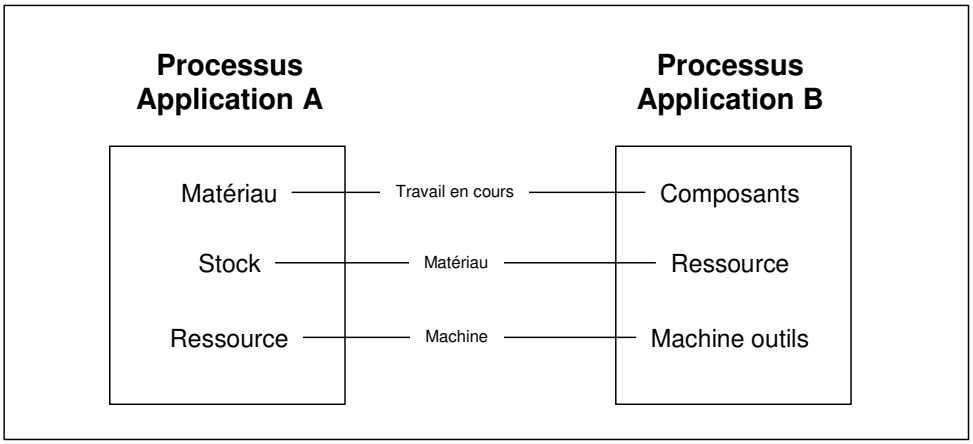


Figure 2.2-11 : Problème de sémantique et PSL

Ce problème amène à des interprétations et des utilisations contradictoires des informations. Les obstacles à l'interopérabilité résultent du fait que les systèmes qui soutiennent les fonctions dans beaucoup d'entreprises ont été créés indépendamment, et ne partagent pas la même sémantique pour la terminologie de leurs modèles de processus.

Considérons la Figure 2.2-11. Dans un cas, en utilisant un système de planification de processus (l'application A) une personne ou une machine est considérée comme une *ressource*. Dans la planification de processus une ressource est un élément qui accomplit une

tâche donnée. Dans le cas de l'application B, nous avons un système de workflow dans lequel *une ressource* est principalement considérée comme une information qui est employée pour prendre les décisions nécessaires. Le terme ressource ne représente pas la même chose dans les deux applications. Si on veut intégrer les deux applications, il est donc nécessaire de considérer à la fois la syntaxe et la sémantique dans le processus de traduction. Une solution serait une ontologie fournissant des définitions explicites et non ambiguës pour les termes utilisés. [Schlenoff *et al*., 2000]

Ainsi, dans ce but, PSL a pour finalité de développer une représentation neutre d'information de processus de fabrication, basée sur des théories et des principes formels. L'objectif est de permettre l'intégration des applications industrielles. C'est un format d'échange conçu pour aider à échanger automatiquement l'information de processus parmi une grande variété d'applications, telles que la définition de processus, la planification, l'ordonnancement ou des outils de simulation.

Approche pour l'élaboration de PSL

L'approche utilisée pour élaborer le langage PSL contient les cinq phases suivantes :

1. Collecte des fonctionnalités,
2. état de l'art sur les approches de modélisation par processus,
3. création du langage,
4. exécution et validation pilotes,
5. soumission comme norme.

Le résultat de la première phase est un ensemble complet de fonctionnalités pour la spécification des processus.
Dans la deuxième phase, vingt-six représentations de processus existantes ont été identifiées comme candidats pour PSL. Ces représentations on été analysées en tenant compte des fonctionnalités recueillies dans la première phase (Figure 2.2-12).
La plupart des candidats étudiés ont été développés pour leurs domaines d'application respectifs et presque toutes ces représentations se sont concentrées sur la syntaxe de la spécification de processus, plutôt que sur la signification des termes.

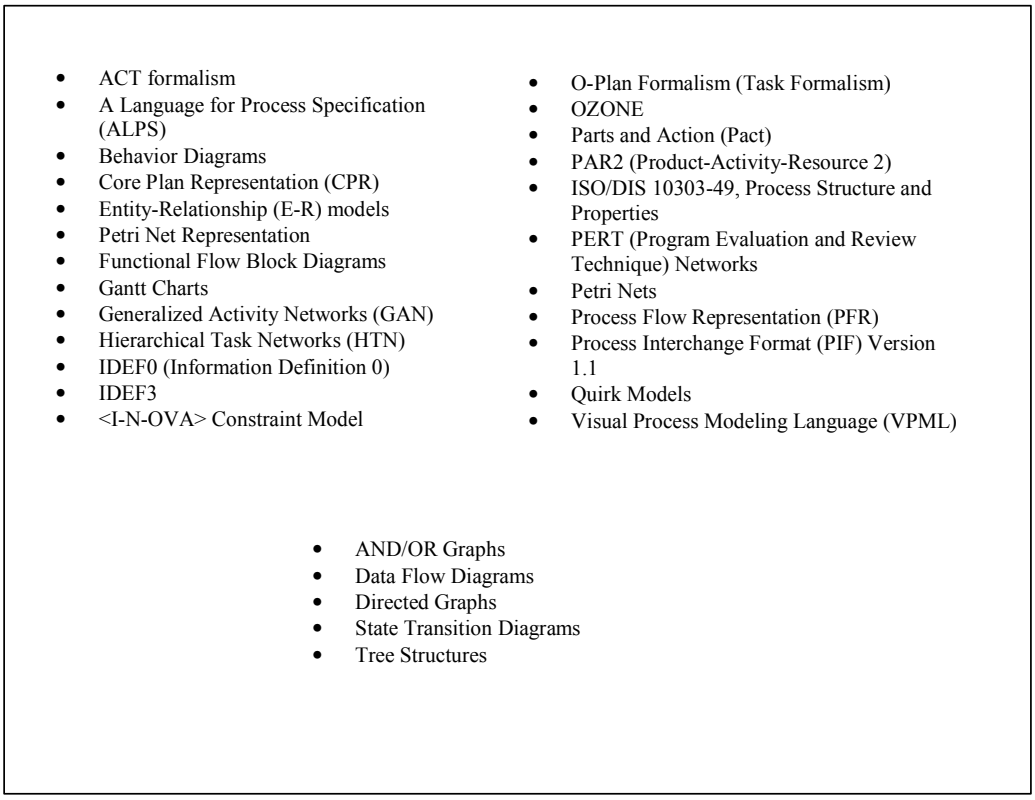


Figure 2.2-12 : Représentations existantes de modélisation par processus

Et comme nous l'avons dit précédemment, la définition de la syntaxe n'est pas suffisante pour permettre l'échange de l'information entre différentes applications, la sémantique doit également être définie. Ainsi, la troisième phase concernant la création du langage, s'est beaucoup concentrée sur le développement d'une couche formelle de la sémantique pour PSL. Le premier composant de PSL est une ontologie conçue pour représenter des concepts de base qui, selon PSL, sont nécessaires pour décrire les processus industriels. L'objectif est une ontologie partagée, où un ensemble arbitraire de termes devrait pouvoir s'inclure.

Le langage de PSL

Un langage est un lexique (un ensemble de symboles) et une grammaire (la spécification de la façon dont ces symboles peuvent être combinés pour créer des formules correctes). Le lexique se compose de symboles logiques (telle que la logique booléenne) et de symboles non logiques. Pour PSL, la partie non logique du lexique se compose des expressions (constantes, symboles, fonction et attributs) choisies pour représenter les composants de base de l'ontologie de PSL. La grammaire fondamentale utilisée pour PSL est basée principalement sur la grammaire de KIF (Knowledge Interchange Format), langage formel basé sur la logique

de premier ordre développé pour l'échange de la connaissance parmi différentes applications informatiques ayant des représentations disparates.

L'ontologie de PSL

La base du langage de spécification de processus est l'ontologie de PSL, qui fournit une définition rigoureuse et non ambiguë des composants nécessaires pour spécifier des processus industriels afin de permettre l'échange d'information. La base de l'ontologie est un ensemble de composants orientés processus qui sont communs à TOUTES LES applications manufacturières. Ces concepts constituent le noyau de l'ontologie de PSL et incluent des concepts tels que OBJECT, ACTIVITY, ACTIVITY OCCURRENCES, et TIMEPOINTS. Cependant, ces concepts seuls ne permettent l'échange d'information que pour des processus très simples. Par conséquent, cette ontologie inclut un mécanisme pour permettre des extensions afin d'assurer la robustesse de l'ontologie.

Traduction avec PSL

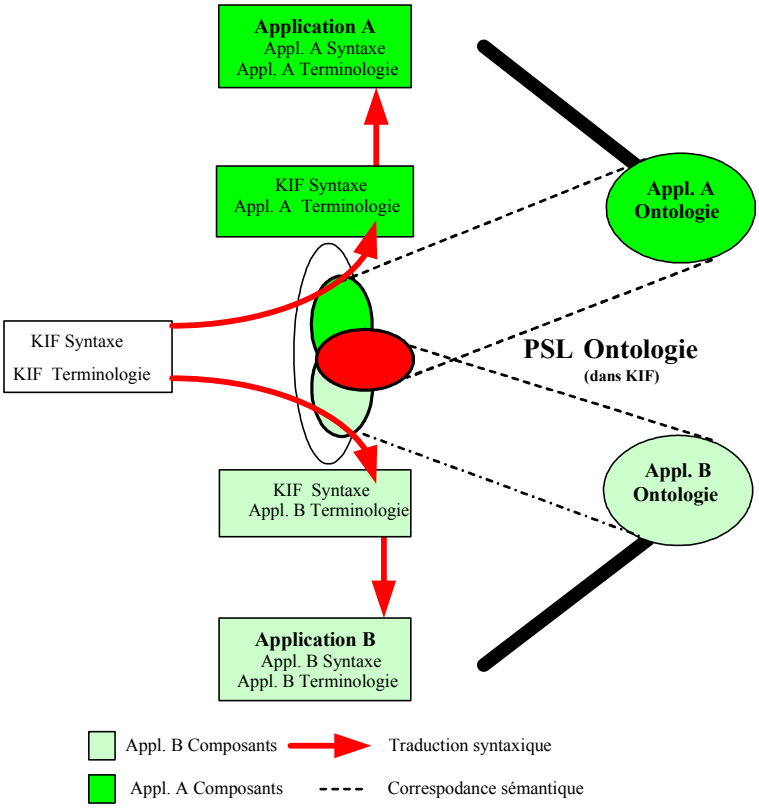


Figure 2.2-13 : Traduction avec PSL

La traduction en employant PSL est un processus en deux étapes (Figure 2.2-13). Dans un premier temps, il y a la traduction syntaxique, puis ensuite vient la traduction sémantique. La traduction syntaxique est faite avec un analyseur qui convertit la syntaxe des d'applications en syntaxe de PSL. Cette première phase de la traduction maintient la terminologie intacte. Puis, dans la phase sémantique de la traduction, la terminologie des applications (et ses ontologies) est traduite pour employer seulement l'ontologie de PSL. Le schéma suivant illustre la traduction entre deux applications distinctes en utilisant PSL en tant que langage intermédiaire.

- Le point de départ est deux applications différentes A et B. Nous voulons intégrer ces deux applications pour que l'information contenue dans l'application A soit disponible pour l'application B.

- Les deux applications ont leurs propres syntaxes et terminologies prédéfinies, liées à leur représentation. En outre, ces deux applications ont des ontologies différentes liées à leurs représentations. Cette ontologie peut être définie de nombreuses manières telles qu'une définition formelle ou être cachée derrière un manuel épais.

- Pour intégrer ces deux applications nous voulons employer l'ontologie de PSL. Ceci signifie que tous les composants relatifs aux processus doivent être définis avec l'ontologie de PSL. Si nécessaire une extension peut être définie dans PSL afin de pouvoir définir le composant manquant.

- La première étape de l'intégration est de définir les liens qui existent entre les composants définis avec les ontologies des deux applications et celle de PSL.

- L'information qui pourra être échangée entre ces deux applications est l'information commune représentée dans l'ontologie de PSL.

- La première étape de la traduction est la traduction syntaxique où la syntaxe de l'application A est traduite en syntaxe KIF. Dans cette phase de la traduction, on maintient la terminologie intacte. Le traducteur syntaxique pour cette étape doit seulement être écrit une fois et il peut être réutilisé par la suite.

- Ensuite, vient la traduction sémantique, où la terminologie de l'application A est automatiquement convertie en terminologie de PSL. Le résultat de cette étape sera une instance d'un fichier neutre de PSL, qui contient l'information à envoyer de l'application A à l'application B.

- Pour rendre l'information réellement disponible pour l'application B, les deux dernières étapes sont faites dans l'ordre inverse. D'abord, la traduction sémantique et ensuite celle syntaxique. Ainsi, l'information contenue dans l'application A est compréhensible pour l'application B, et elle peut également l'utiliser.

A l'origine, le développement de PSL n'a pas eu pour finalité de faire de ce langage un langage de modélisation de processus, mais un langage d'échange, ce qui est similaire à ce que l'on veut faire avec UEML. Par exemple, une application basée sur le langage IDEF3 peut employer PSL pour échanger des modèles de processus avec une application utilisant les réseaux de Petri, de la même manière que STEP peut être employé pour échanger des modèles de produits parmi des systèmes de CAO.

2.3 Architecture pour la comparaison des approches de modélisation d'entreprise

Afin de pouvoir comparer les approches de modélisation d'entreprise, nous avons besoin de quelques critères. Pour cet état de l'art, nous avons choisi d'utiliser l'architecture qui a été développée dans le réseau thématique UEML [IST - 2001 – 34229], auquel nous avons participé. Cette architecture a été inspirée par un certain nombre de cadres de modélisation existants. On trouve ainsi, entre autre, GERAM (des simplifications importantes on été cependant faites par rapport à GERAM) et le cadre de modélisation de Zachman pour l'architecture d'entreprise. Ainsi, ces différents cadres de modélisation ont permis de définir un ensemble de dimensions sur lesquelles les approches de modélisation d'entreprise peuvent être positionnées. Cette architecture permet de mettre en évidence les forces et les faiblesses des différentes approches et ainsi permet de conclure sur les besoins de recherches et développements concernant les approches en modélisation d'entreprise.

Les dimensions de cette architecture sont :

- La dimension des composants du monde de l'ingénierie d'entreprise,
- la dimension de cycle de vie d'entreprise,
- la dimension de généricité,
- la dimension des vues.

La première dimension s'applique à n'importe quelle approche de modélisation d'entreprise. Les trois autres s'appliquent seulement aux approches qui proposent un langage de modélisation d'entreprise, un outil ou une méthodologie.

2.3.1 La dimension des composants du monde de l'ingénierie d'entreprise

La Figure 2.3-1 illustre le modèle des composants du monde de l'ingénierie d'entreprise, tel qu'il a été défini lors du réseau thématique UEML. Les quatre niveaux, représentés par des rectangles blancs, correspondent à quatre types d'informations qui sont créées, gérées et modifiées par des outils informatiques, représentés par des rectangles gris clair. Ces outils utilisent les informations en amont, afin de créer les informations en aval. Ces informations sont classées suivant l'environnement auxquelles elles appartiennent.

Ainsi, on trouve :

- **L'environnement opérationnel d'entreprise** qui correspond à l'environnement dans lequel des modèles sont exécutés à l'aide des outils opérationnels d'entreprise, afin de créer et gérer des instances d'entreprises, suivant la terminologie utilisée dans ce réseau thématique. Ces instances correspondent à des objets ou informations que l'on trouve dans l'entreprise telles que des données de simulation.

- **L'environnement d'ingénierie d'entreprise** qui correspond à l'environnement dans lequel des outils d'ingénierie d'entreprise sont utilisés, afin de créer et contrôler des modèles d'entreprise, sur la base des méthodologies d'ingénierie d'entreprise.

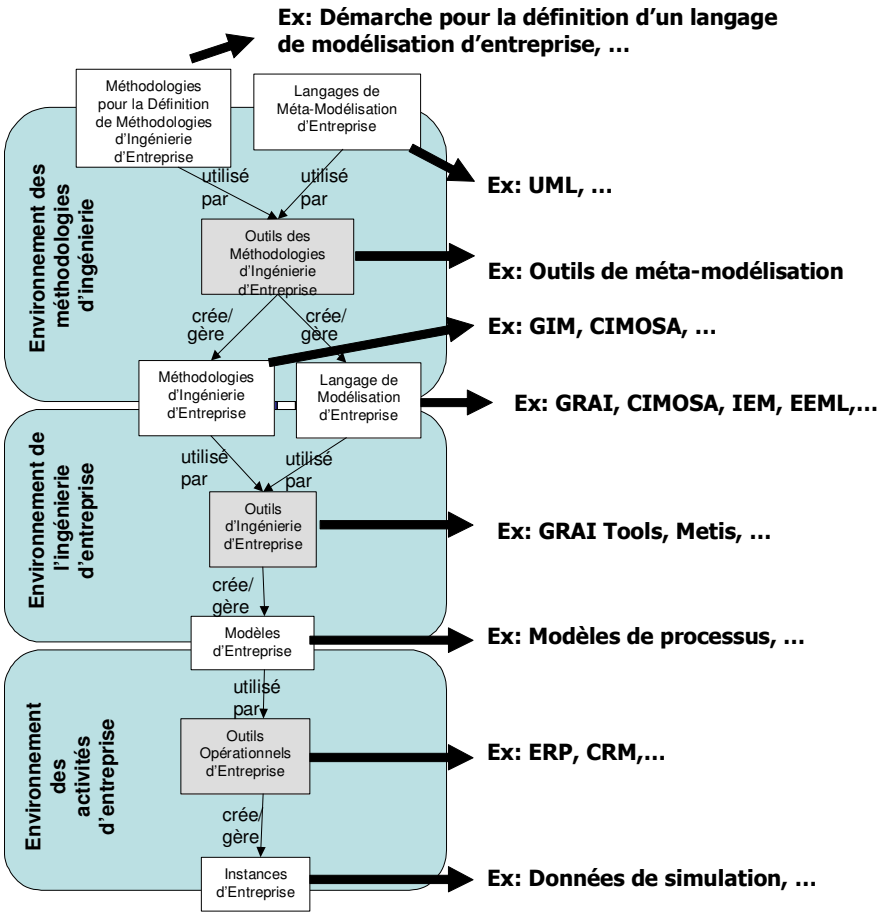


Figure 2.3-1 : Composants du monde de l'ingénierie d'entreprise

- **l'environnement des méthodologies d'ingénierie** qui correspond à l'environnement dans lequel les langages de modélisation d'entreprise et les méthodologies sont créées et adaptées, afin de respecter les besoins des entreprises modélisées. Dans cet environnement, les outils d'ingénierie en méthodologies d'entreprise sont utilisés et basés eux-mêmes sur des langages de méta-modélisation et des méthodologies pour la définition des méthodologies d'ingénierie d'entreprise.

De ces trois environnements, seulement les deux premiers sont définis dans GERAM.

Parmi les composants définis précédemment, on peut en choisir un certain nombre, afin de caractériser les approches de modélisation d'entreprise. Comme nous nous focalisons principalement sur la modélisation d'entreprise et la définition des langages de modélisation d'entreprise, nous nous concentrons sur les composants suivants :

- Outils d'ingénierie d'entreprise (OIEs), Méthodologies d'Ingénierie d'Entreprise (MIEs) et Langages de Modélisation d'entreprise (LMEs) afin d'identifier les approches en modélisation d'entreprise traitant de l'ingénierie d'entreprise,

- Outils des Méthodologies d'Ingénierie d'Entreprise (OMIEs), Langages de Méta-Modélisation d'Entreprise (LMMEs) et Méthodologies pour la Définition de Méthodologies d'Ingénierie d'Entreprise (MDMIEs) afin d'identifier les approches en modélisation d'entreprise traitant des méthodologies d'ingénierie.

En conséquence, on peut caractériser une approche relative à la modélisation d'entreprise en fonction des composants qu'elle possède. On peut noter que quelques approches possèdent des composants dans plus d'un environnement. C'est le cas, par exemple, d'outils de modélisation d'entreprise qui permettent aux utilisateurs d'adapter le langage de modélisation d'entreprise utilisé (par exemple en étendant certains concepts par l'ajout d'attributs additionnels). Ces approches peuvent être caractérisées par les composants suivants :

- Outils d'Ingénierie d'Entreprise (OIEs) parce qu'ils proposent un outil pour modéliser l'entreprise,

- Langages de Modélisation d'Entreprise (LMEs) parce que l'outil utilise un langage de modélisation d'entreprise,

- Outils des Méthodologies d'Ingénierie d'Entreprise (OMIEs) parce qu'elles fournissent une fonctionnalité de l'outil permettant de modifier le LME utilisé,

- Langages de Méta-Modélisation d'Entreprise (LMMEs) parce qu'elles donnent accès à l'utilisateur à des langages de méta-modélisation afin de lui permettre de modifier le LME).

- Méthodologies d'Ingénierie d'Entreprise (MIEs), si elles fournissent un support méthodologique pour la création des modèles,

- Méthodologies pour la Définition de Méthodologies d'Ingénierie d'Entreprise (MDMIEs), si elles fournissent un support méthodologique pour la personnalisation du LME par méta-modélisation.

2.3.2 La dimension de cycle de vie

Puisque nous nous focalisons sur la modélisation d'entreprise (et non sur les opérations d'entreprise), les phases du cycle de vie sur lesquelles nous nous concentrons sont les cinq premières dimensions de GERAM. Les deux premières phases de GERAM ont été fusionnées en une seule, pour des raisons de simplification.

On a donc les phases suivantes :

- **Initialisation et définitions des objectifs :** C'est l'ensemble des activités qui identifient le contenu d'une entreprise particulière, en considérant ses frontières et ses relations avec ses environnements internes et externes.

- **Définition des besoins :** les activités nécessaires afin de développer les descriptions des besoins opérationnels des entités d'entreprises, les processus significatifs et l'ensemble de ses besoins en terme de fonctionnalités, comportements, informations et capacités.

- **Conception :** Les activités qui soutiennent les spécifications de l'entité (entreprise) avec tous ses composants répondant aux exigences d'entité (entreprise).

- **Implémentation :** Les activités qui définissent toutes les tâches qui doivent être effectuées pour construire ou reconstruire l'entité (entreprise).

Ces phases ne doivent pas être considérées strictement comme des activités exécutées dans l'ordre mais elles décrivent plutôt différents niveaux de détails dans la définition de

l'entreprise. Nous les employons pour décrire, par exemple, un langage de modélisation d'entreprise en caractérisant son niveau de détail et d'adéquation avec l'objet des différentes étapes de la définition de l'entreprise.

La classification selon cette dimension est applicable aux LMEs, aux OIEs et aux MIEs.

2.3.3 La dimension de généricité

La dimension de généricité, inspirée par GERAM, permet de classifier les approches en fonction de leurs applicabilités dans diverses situations. Quelques approches peuvent être très génériques (applicables à un ensemble large d'entreprises et de situations). À l'opposé, d'autres peuvent seulement être employées pour un ensemble restreint de situations.

Ainsi, on peut dégager les trois types de généricités suivantes :

- **Générique :** l'approche est applicable à un ensemble large de situations (potentiellement toutes sortes d'entreprises).

- **Partiel :** l'approche est spécialisée pour un ensemble de situations ou un domaine particulier d'application (par exemple une approche consacrée aux entreprises manufacturières ou une approche consacrée au secteur tertiaire).

- **Détail :** l'approche est très spécialisée et consacrée à un nombre limité de situations (et probablement d'une entreprise seule).

La classification selon cette dimension est applicable aux LMEs, aux OIEs et aux MIEs.

2.3.4 La dimension des vues

Cette dimension permet de caractériser les LMEs, les OIEs et les MIEs selon les aspects de l'entreprise qu'ils permettent de modéliser. Selon GERAM, les vues sont utilisées afin de « réduire l'apparente difficulté des modèles entreprises résultants ». Les vues retenues sont :

- **La vue de Processus / fonction** représente les fonctionnalités et le comportement des processus d'entreprise.

- **La vue information** collecte la connaissance concernant les objets de l'entreprise (matériels et informationnels) qui sont employés et générés au cours des opérations d'entreprise.

- **La vue de ressource** représente les ressources (humaines et agents techniques ainsi que les composants technologiques) de l'entreprise utilisées au cours des opérations d'entreprise.

- **La vue d'organisation** représente les responsabilités et les autorités sur toutes les entités identifiées dans les autres vues (processus, information, ressource). Elle couvre la structure de l'organisation d'entreprise en organisant les unités d'organisation identifiées en plus grandes unités, tels que des départements, des divisions, des sections, etc...

D'autres vues sont possibles. Dans ce cas, un nom est donné à la vue et elle est brièvement décrite afin de permettre de caractériser l'approche en modélisation d'entreprise.

On peut finalement représenter cette architecture par la Figure 2.3-2.

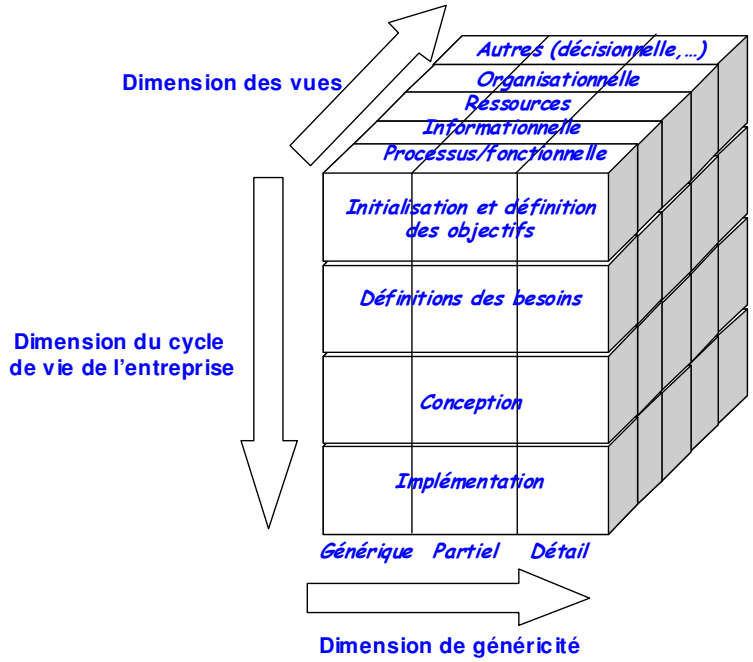


Figure 2.3-2 : Les trois dimensions de l'architecture

2.3.5 Positionnement des approches de modélisation d'entreprise

Ainsi, nous avons vu qu'il existe beaucoup d'approches en modélisation d'entreprise (méthodologies, langages et outils) couvrant partiellement *les dimensions de l'architecture* que nous venons de définir. Ainsi, par exemple, la norme ENV12204 fournit un langage de modélisation d'entreprise, mais pas d'autres composants (aucune outil, méthodologie ou capacité à méta-modéliser). Par ailleurs, GERAM et l'ISO/IEC 15288 définissent des guides méthodologiques pour l'ingénierie d'entreprise (concepts pour le processus l'ingénierie d'entreprise) mais ne fournissent pas à proprement parler de langages de modélisation. Toutes ces approches ne se focalisent par sur la même étape du cycle de vie de l'entreprise. Par exemple, CIM-OSA se concentre principalement sur la définition de besoins fonctionnels. De même, elles possèdent une applicabilité différente selon leur niveau de généricité. En effet, PSL se concentre principalement sur la modélisation des processus industriels alors que l'approche IEM (dont nous parlerons dans le chapitre 5) est très générique et peut être appliquée pour modéliser n'importe quel genre de processus. Toutes ces approches possèdent un ensemble de vues particulières suivant leurs objectifs de modélisation. GIM couvre toutes les vues prédéfinies dans notre architecture et également une vue décisionnelle, alors que PSL couvre principalement la vue processus. CIMOSA couvre les vues fonctionnelle, informationnelle, ressource et organisationnelle.

Par ailleurs, une fonctionnalité attendue pour UEML est de pouvoir servir de langage commun à l'échange. Ainsi, il doit d'une manière ou d'une autre être une intégration ou une intersection de langages existants. Le méta-modèle d' UEML pourrait donc être défini en considérant les méta-modèles de langages existants en essayant de les « intégrer » en un seul langage. Ce problème a beaucoup de similitudes avec un problème bien connu et bien étudié en informatique : le problème de l'intégration de schéma dans les bases de données. Cette comparaison a été réalisée dans [Petit, 2003].

2.4 L'intégration de bases de données et la définition d'UEML

L'intégration de bases de données consiste à prendre en entrée un ensemble de bases de données (schémas et populations), et à produire en sortie une description unifiée des schémas

initiaux (le schéma intégré) et les règles de traduction (mapping) qui vont permettre l'accès aux données existantes à partir du schéma intégré.
L'intégration de bases de données est un problème complexe. Un nombre considérable d'articles en ont étudié les différents aspects : il en résulte une multitude de contributions techniques, quelques méthodologies et quelques prototypes. Nous présentons ici les résultats de cette comparaison et nous invitons le lecteur à consulter [Parent *et al.*, 2000].

Lors de l'intégration de bases de données, on peut distinguer les phases suivantes :

- La pré-intégration, une étape dans laquelle les schémas en entrée sont transformés de différentes manières pour les rendre plus homogènes (sur les plans sémantique et syntaxique),

- la recherche des correspondances, une étape consacrée à l'identification des éléments semblables dans les schémas initiaux et à la description précise de ces liens inter-schémas,

- l'intégration, l'étape finale qui unifie les types en correspondance en un schéma intégré et produit les règles de traduction associées entre le schéma intégré et les schémas initiaux.

2.4.1 Etape 1 : La pré-intégration

2.4.1.1 Etape de pré-intégration de base de données

Dans la plupart des cas, les bases de données à intégrer auront été développées indépendamment et seront par conséquent hétérogènes. La différence la plus évidente est lorsque les bases de données existantes ont été installées avec des Systèmes de Gestion de Basses de Données utilisant des modèles de données différents (relationnel, orienté-objets, ...). Cependant, même en admettant que toutes les bases de données soient converties dans un même modèle, il est évident que des différences subsisteront dans la richesse de la description obtenue. En effet, certains modèles de données possèdent dans leurs concepts plus de sémantique que d'autres modèles. Par exemple, un schéma entité-relation utilise des concepts séparés pour les entités et pour les associations, tandis que le schéma relationnel équivalent va

décrire les mêmes données sans faire de distinction explicite entre entités et relations. Pour parvenir à un même niveau de compréhension des deux schémas, on doit être à même d'identifier, dans le schéma relationnel, quelles relations décrivent des entités et quelles relations décrivent des associations. Par ailleurs, une autre source évidente d'hétérogénéité réside dans le non-déterminisme du processus de modélisation. Deux concepteurs qui doivent représenter la même réalité avec le même modèle de données vont inévitablement créer deux schémas différents.

2.4.1.2 Préparation à la définition d'UEML

Ainsi, de la même manière que dans un problème d'intégration de base de données, la définition d'UEML exige l'ensemble et la définition des schémas à intégrer. Dans ce cas, les schémas correspondent à l'ensemble des méta-modèles des langages de modélisation d'entreprise considérés pour l'intégration. Les méta-modèles de ces langages ne sont pas nécessairement directement disponibles. De plus, la plupart des langages sont décrits en termes de syntaxe, mais ne font pas toujours référence explicitement aux relations qui existent parmi les différents composants, ainsi qu'aux contraintes qui s'appliquent pour obtenir des modèles valides écrits dans ces langages. Un premier exercice nécessaire, est donc de définir avec précision les méta-modèles de ces langages. Dans certains cas, les méta-modèles peuvent être obtenus en partant d'une application informatique support. Dans d'autres cas, les méta-modèles doivent être élaborés à la main sur la base de la littérature décrivant le langage.

2.4.2 Etape 2 : La recherche des correspondances

2.4.2.1 Recherche et définition des correspondances dans les bases de données

La prochaine étape dans la méthodologie consiste à établir ce qui est commun dans les bases de données qui sont candidates à l'intégration. Comme [Parent *et al*., 2000] le suggère, les correspondances doivent être définies à deux niveaux. Au ***niveau des données*** (population de la base de données), les correspondances parmi les instances des classes présentes dans une base de données et celles représentées dans les autres doivent être identifiées. Pour établir ces correspondances, la sémantique des instances doit être considérée. Deux instances sont considérées comme ayant la même sémantique si elles décrivent le même élément du monde réel. Au ***niveau du schéma*** de la base de données, les correspondances parmi les classes sont

établies si les correspondances parmi des instances s'appliquent à un ensemble significatif d'instances de ces classes. La correspondance peut ainsi être généralisée au niveau de la classe.

2.4.2.2 Recherche et définition des correspondances entre langages de modélisation d'entreprise

Pour la définition d'UEML, nous devons établir des correspondances parmi les classes définies dans les différents méta-modèles. Mais comme dit précédemment, nous ne pouvons définir des correspondances à ce niveau, que si des correspondances identifiées sur des exemples peuvent être généralisées. Ceci signifie que nous devons d'abord étudier des correspondances parmi des éléments de modèles crées avec les langages différents et les généraliser par la suite, si les éléments possèdent la même sémantique. Dans notre cas, la sémantique des éléments est plus compliquée que dans le cas de bases de données, car les modèles d'entreprise représentent habituellement des ensembles d'éléments ou d'événements du monde réel plutôt que des éléments individuels. Par exemple, un élément tel que la classe d'objet représente une population entière d'une base de données, dont les éléments eux-mêmes ont une correspondance dans le monde réel. Une correspondance parmi les langages peut donc être établie seulement si les éléments des modèles créés avec ces langages représentent le même ensemble d'éléments du monde réel. Le même principe s'applique à la comparaison de la sémantique du comportement des éléments des modèles, tels que les processus. Dans ce cas-ci, la sémantique est bien plus complexe à comparer puisque les processus ont une sémantique d'un point de vue dynamique, représentant potentiellement des ensembles infinis de comportements. La comparaison de la sémantique doit s'assurer, dans ce cas-ci, que les ensembles de comportements de processus pour tous les modèles correspondent. Cette comparaison peut être réalisée automatiquement si la sémantique des modèles de processus est définie formellement.

2.4.3 Etape 3 : L'intégration

2.4.3.1 Intégration des schémas de bases de données

La troisième et dernière étape est de définir le schéma intégré de la base de données en considérant l'inclusion des éléments des schémas originaux dans le schéma intégré. Ce processus traite systématiquement chaque correspondance identifiée dans l'étape 2 et décide

quels éléments sont à inclure dans le schéma intégré. Des conflits potentiels doivent être résolus à ce niveau. Un exemple simple de conflit est un « conflit de description » qui se produit quand deux classes correspondantes ont un ensemble différent d'attributs. Dans ce cas-ci, le conflit doit être résolu en décidant lesquels de ces attributs doivent être inclus dans le schéma intégré. Pour résoudre ces conflits, différentes stratégies peuvent être adoptées selon l'objectif poursuivi pour l'intégration. Par exemple, une stratégie peut être de rechercher à obtenir un schéma intégré complet ou alors de se contenter d'un schéma simple.

2.4.3.2 Intégration des méta-modèles des langages dans le méta-modèle d'UEML

La stratégie concernant l'apparition des conflits pendant l'intégration doit être définie et dépend de l'objectif d'UEML. Un objectif raisonnable d'UEML peut être l'interopérabilité d'un ensemble d'outils supports à des langages de modélisation d'entreprise existants. Dans ce cas-ci, une stratégie globale adaptée serait d'inclure seulement un composant dans le méta-modèle d'UEML s'il existe les composants correspondants dans les méta-modèles d'au moins deux langages à intégrer. Par conséquent, si un composant ou un attribut est spécifique à un langage, il ne sera pas utile pour les autres. Si l'objectif est plutôt d'obtenir un modèle intégré de modélisation d'entreprise, une stratégie plus complète concernant l'élaboration du méta-modèle d'UEML sera plus appropriée. Dans tous les cas, une définition explicite des correspondances entre les composants des méta-modèles des langages originaux et les composants correspondants dans le méta-modèle d'UEML sont utiles.

Suivant les commentaires ci-dessus, dans le réseau thématique UEML [IST - 2001 – 34229], nous avons défini un scénario complexe (une étude de cas) qui a été représenté dans trois langages (à savoir GRAI, EEML, IEM). Ce scénario joue le rôle des données. Un méta-modèle a été défini sous forme d'un modèle de classes UML pour chacun des ces langages. Ces méta-modèles jouent le rôle des schémas d'une base des données à intégrer. Par analogie avec l'intégration des schémas dans les bases de données, nous avons donc comparé les méta-modèles sur la base des objets de modèle faisant partie des modèles du scénario. Ces travaux seront présentés dans le chapitre 5 de ce mémoire.

2.5 Conclusion

Ainsi, la recherche en modélisation d'entreprise a donc donné lieu au fil des années à de nombreux cadres de modélisation et langages pour la modélisation d'entreprise. Cependant, aucun langage n'est à l'heure actuelle assez « complet » pour être promu au rang de « langage unifié de modélisation d'entreprise ». Il possède tous entre eux des intersections quand au domaine de l'entreprise qu'ils permettent de modéliser, mais il n'existe pas de langage suffisamment intégré pour permettre de représenter l'information contenu dans les divers autres langages. Par ailleurs, de nombreux efforts de recherche on donné naissance à diverses approches qui, bien que n'étant pas encore assez abouties pour être un langage unifié, apportent des résultats intéressants sur le sujet. Ainsi, et comme spécifié dans [Berio *et al.,* 2004] et [Panetto et al., 2004] le langage PSL et la norme ENV12204/CEN/ISO 19440 sont deux des principaux travaux dont l'objectif est la définition d'un noyau de langage commun pour la modélisation d'entreprise. Cependant, ces efforts ne fournissent pas, à l'heure actuelle, une solution satisfaisante ni aux problèmes importants, ni aux problèmes pratiques. PSL propose une approche basée sur la logique mais ne règle pas clairement la question de base de mécanismes d'échange entre un langage quelconque et PSL lui-même. Les mécanismes d'échange devraient exprimer, par exemple, que le composant d'activité appartenant à un langage de modélisation d'entreprise correspond au composant d'activité en PSL. Puisqu'il est un langage logique déclaratif, PSL permet de découvrir des incohérences entre des modèles exprimés dans des langages distincts. Cependant, il n'offre pas de solution pour empêcher ni résoudre ces incohérences. Le CEN/ISO 19440 est simplement un ensemble de composants utiles pour comprendre le domaine de la modélisation d'entreprise et il définit l'ensemble des composants qu'un langage doit permettre de représenter. Néanmoins, sa syntaxe n'est pas définie suffisamment rigoureusement pour qu'il puisse servir de format d'échange entre des outils de modélisation.

Ainsi, grâce à cet état de l'art, nous avons pu dégager certains points et notamment le fait que les langages actuels ne peuvent pas être utilisés pour être un langage unifié de modélisation d'entreprise. Par conséquent, ce langage doit être défini. Par ailleurs, la définition d'UEML est en quelques points semblable à la définition d'autres langages. En informatique, il est habituel de commencer la définition d'un langage par l'identification des fonctionnalités de bases que ce langage devra fournir. Cet ensemble de fonctionnalités peut alors servir de base pour la

définition de ce langage. Il est fréquent, pour définir cet ensemble de fonctionnalités, qu'un modèle statique de données soit construit, ce modèle s'appelle un méta-modèle. Ainsi, le méta-modèle d'UEML doit être élaboré de sorte qu'il puisse être défini et compris avec précision.

Dans ce but, nous allons dans le chapitre 3, donner notre point de vue sur l'élaboration d'un langage unifié de modélisation et apporter une contribution pour résoudre les difficultés inhérentes à cette problématique.

Chapitre III

Principes, démarche et illustration empirique

Chapitre III
Principes, démarche et illustration empirique

3.1 Introduction

*Dans ce mémoire, le terme **composant** fait référence à tout élément de modélisation, tel que le composant d'activité par exemple, appartenant à un langage. Chaque langage de modélisation est donc composé de plusieurs composants. Un composant peut être considéré comme atomique et donc non décomposable, mais il peut également être un composant composite construit à partir d'autres composants. Par exemple, le composant d'activité peut être vu comme un élément de modélisation indécomposable mais il peut s'avérer intéressant de le considérer comme composé d'autres composants plus élémentaires, tel que son coût par exemple. La nécessité de décomposer certains composants sera détaillée ultérieurement.*

Dans le chapitre 1 de ce mémoire, nous avons montré la nécessité de définir un langage unifié de modélisation d'entreprise. Nous allons dans ce chapitre, aborder dans un premier temps les principes sur lesquels nous nous basons concernant le langage UEML. Ces principes présentent notre point de vue sur le sujet, et peuvent différer de certains travaux développés

par différents groupes (réseau thématique UEML [IST - 2001 – 34229], ECEIMT ([Petit *et al*., 1997], [Panetto *et al.*, 2001], [Vernadat, 1999b], [Vernadat, 2001]) et ne sont pas forcément partagés par l'ensemble de la communauté scientifique. Dans un deuxième temps, nous présenterons un des aspects très important pour la construction de notre langage UEML : la « traduction de composants ». Nous montrerons un exemple simple de traduction entre une activité SADT et une activité GRAI (composants des réseaux GRAI). Dans ce chapitre, cette traduction s'appuiera sur une approche empirique.

3.2 Principes adoptés

La position scientifique que nous retenons n'est pas de développer un nouveau langage devant remplacer ceux existants et actuellement utilisés, mais plutôt d'être capable d'interpréter ces derniers. Celui-ci pourrait être un « Esperanto » [Vernadat, 2001] dans le domaine de la modélisation et de l'ingénierie d'entreprise. C'est-à-dire un langage neutre qui servirait d'interface de communication entre plusieurs langages de modélisation d'entreprise existants. Ainsi, il ne constitue pas le langage ultime de modélisation d'entreprise remplaçant tous les précédents mais un méta-modèle standard largement accepté par les utilisateurs comme par les développeurs d'outils informatiques. En d'autres termes, il n'a pas vocation à être intégré dans *« la boîte à outils de l'analyste »*, aussi, il n'est pas contraint par des critères d'utilisabilité opérationnelle.

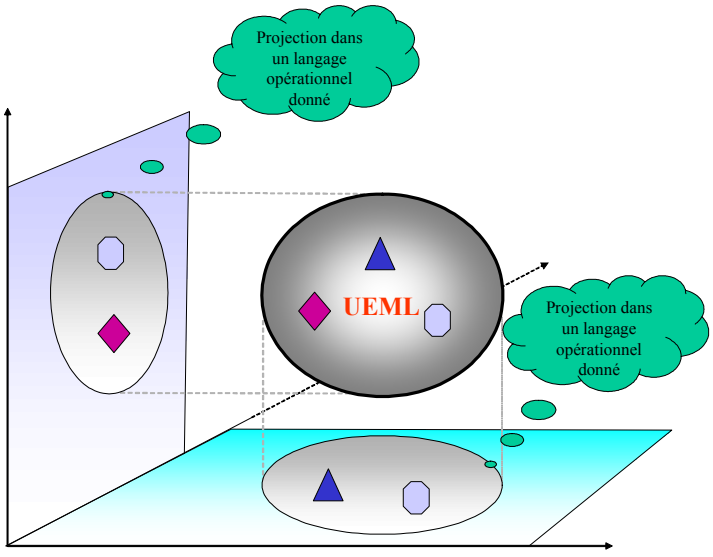


Figure 3.2-1 : Position d'UEML par rapport aux langages opérationnels (projection)

Le résultat serait donc, dans ce cas, un langage compatible avec les langages opérationnels les plus couramment utilisés. En conséquence, puisque notre objectif n'est pas de rendre ce

langage opérationnel, il ne sera pas intégré dans une démarche de modélisation comme c'est le cas pour la majorité des langages initiaux. En revanche, ce langage unifié permettra une interopérabilité des modèles même lorsque ceux-ci ont été réalisés avec des langages différents. Les composants de modélisation du langage unifié ainsi que les mécanismes d'interprétation des modèles initiaux devront être décrits dans un langage formel. Enfin, puisque UEML doit devenir une base conceptuelle générale, cela signifie implicitement que la couverture de modélisation d'UEML doit être supérieure aux couvertures de modélisation individuelles. UEML apparaît ainsi comme ***l'union*** des langages existants et, inversement, les langages de modélisation existant deviennent alors des interfaces opérationnelles, c'est-à-dire des projections d'UEML (Figure 3.2-1).

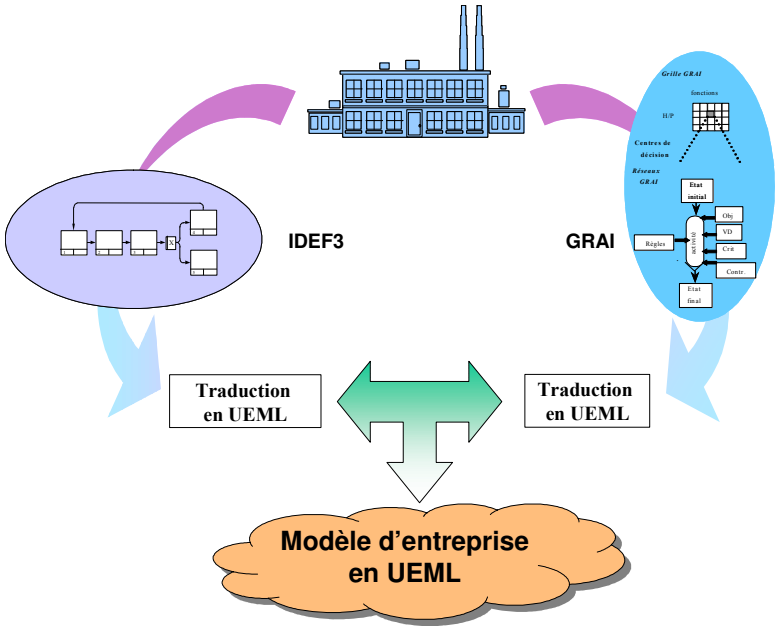


Figure 3.2-2 : Modèle d'entreprise avec UEML

En conséquence, UEML aura une plus grande couverture de modélisation que l'ensemble des différents langages existants. De ce fait, avec ce langage unifié, il sera possible d'établir un modèle d'une entreprise en utilisant les compétences de différents langages. Par exemple, comme illustré sur la Figure 3.2-2, les aspects décisionnels d'une entreprise peuvent être modélisés en employant la grille et les réseaux GRAI [Doumeingts, 1984], [Vallespir *et al.*, 1997], [Vallespir *et al.*, 2002] et en employant IDEF3 [Mayer *et al.*, 1992] pour établir le modèle de processus de l'entreprise. Même si il est vrai, qu'actuellement, il est déjà possible de procéder de cette manière afin de modéliser une entreprise, rien n'assure cependant que les différents modèles soient cohérents entre eux. La projection des différents modèles d'entreprise sur un modèle UEML pourra permettre d'assurer cette cohérence. On obtiendra

ainsi un modèle plus global de l'entreprise et on pourra lui faire subir différents traitements (simulation, …).

Nous retrouvons ainsi une logique un peu similaire à celle présidant l'existence des cadres de modélisation (cf. chapitre 2).

Un modèle UEML contient, dans un format neutre, tous les composants nécessaires à l'expression des fonctions, décisions, processus, activités, ressources et comportements de l'entreprise. Une vue, au sens cadre de modélisation, n'est qu'une projection du modèle UEML en fonction d'un point de vue spécifique (Figure 3.2-3), [Vallespir, 2003].

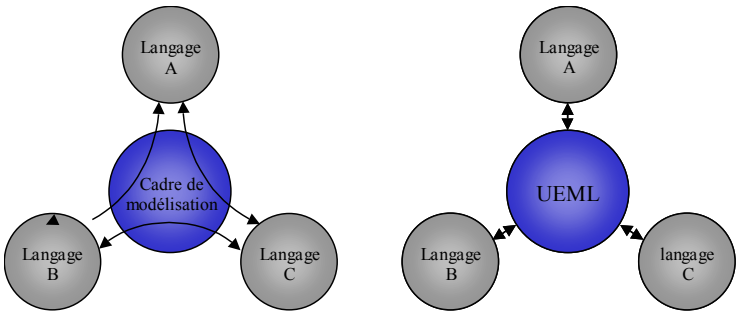


Figure 3.2-3 : Intégration - l'approche par cadre de modélisation et UEML

3.3 Démarche de développement d'UEML

Afin d'élaborer notre langage unifié de modélisation d'entreprise, nous pouvons dégager un ensemble de démarches. Une première ébauche de ces démarches a été développée dans [Chen *et al.*, 2002] et elle est à la base de ce travail. Ainsi, on peut distinguer trois approches différentes : l'approche descendante, ascendante ou hybride.

3.3.1 L'approche ascendante

L'approche ascendante commence par l'analyse des langages de modélisation d'entreprise existant. Ces langages sont le point de départ de l'élaboration du langage UEML. L'approche peut être structurée en quatre étapes (Figure 3.3-1).

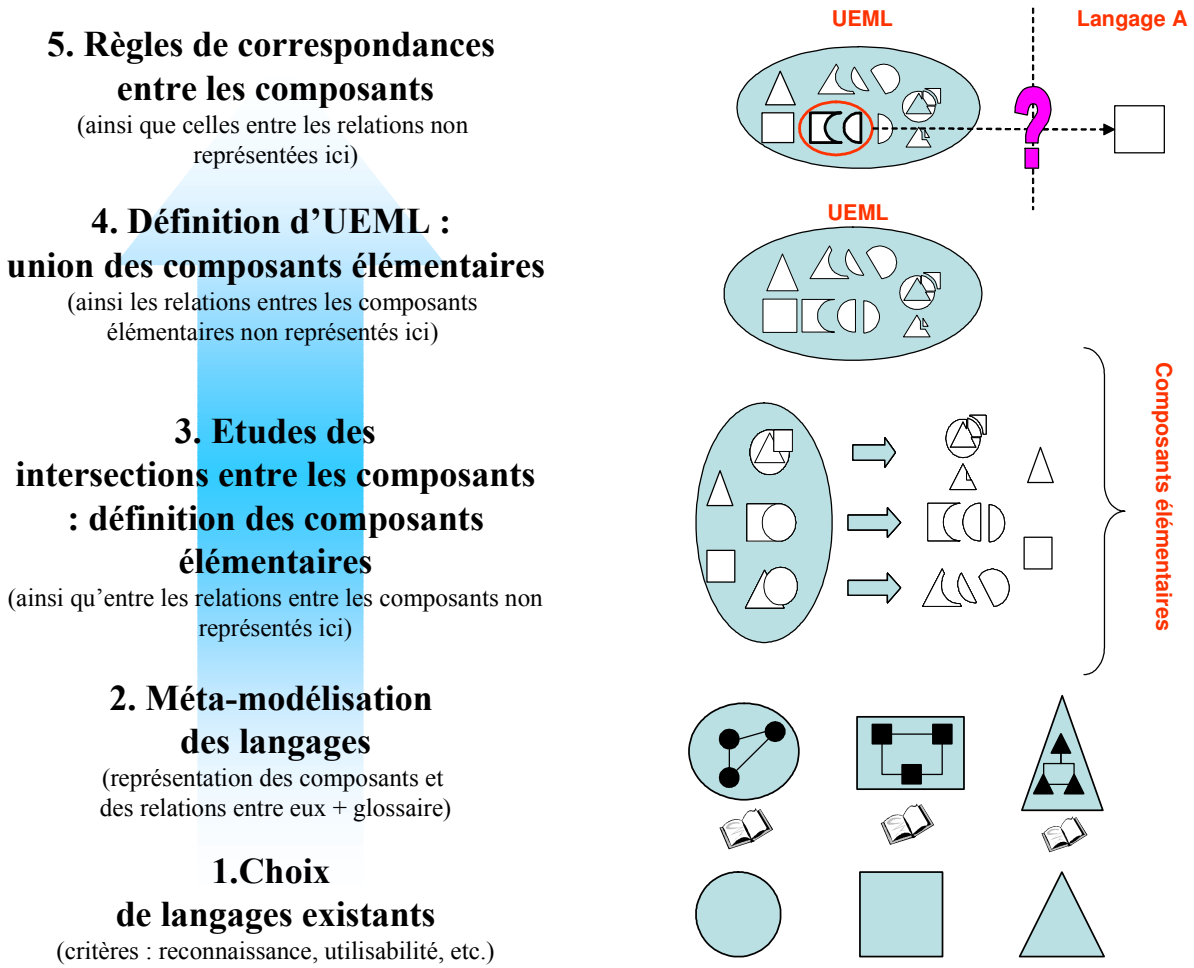


Figure 3.3-1 : Approche ascendante

(1) Dans une première étape, on choisit des langages de modélisation existants qui nous semblent pertinents pour l'élaboration de notre langage UEML. Ce choix peut s'effectuer sur plusieurs critères, comme la reconnaissance de tel ou tel langage dans la communauté. On peut aussi choisir un langage en fonction de son utilisabilité, de sa couverture de modélisation, des composants intéressants qu'il possède etc… A ce niveau là, il n'y a à proprement parler, aucune considération des fonctionnalités attendues de notre langage.

(2) Ensuite, dans une deuxième étape, il est nécessaire de comparer, d'une part les composants des différents langages afin de faire ressortir les composants communs, et d'autre part l'ensemble des liaisons qui existent entre ces composants. Dans ce but, il est nécessaire de décomposer et d'étudier chaque langage en identifiant tous ses composants et les liaisons entre eux. Notre approche étant basée sur la méta-modélisation (nous en parlerons plus en détail dans le chapitre 4), c'est à ce niveau là, que l'ensemble des

méta-modèles concernant les langages doit être défini à l'aide du diagramme de classe UML. De même, l'utilisation de langage plus formel, tel que le langage OCL, peut s'avérer être d'une aide précieuse, afin d'éviter que les méta-modèles construits ne véhiculent trop d'ambiguïtés. La construction d'un glossaire définissant chaque composants des langages est ici également indispensable. Il sera en effet utilisé dans l'étape suivante.

(3) Dans une troisième étape, nous devons déterminer un ensemble de composants élémentaires en comparant les composants des langages considérés. De plus, dans cette étape nous devons également étudier l'ensemble de liens qui relient les différents composants des langages. Ces différentes tâches permettent ainsi de définir un ensemble de règles de correspondances entre les composants des langages et les différentes associations entre les composants.

(4) Enfin, dans une dernière étape, il est nécessaire de construire le méta-modèle du langage UEML. Nous retrouvons, dans ce méta-modèle, tous les composants définis à l'étape récédente reliés par un ensemble d'associations.

3.3.1.1 Avantage de l'approche ascendante

L'avantage de l'approche ascendante est d'être rapide et d'éviter de redécouvrir l'existant. On part de langages actuels qui ont déjà une certaine maturité dans le domaine de la modélisation d'entreprise.

3.3.1.2 Inconvénient de l'approche ascendante

L'inconvénient est que rien n'assure que les langages choisis soient représentatifs. En effet, il est par exemple impossible de faire de la simulation sur un modèle UEML, si les langages qui ont été considérés pour élaborer le langage unifié, ne prennent pas en compte les aspects temporels. Par ailleurs, rien n'assure la cohérence du langage obtenu en terme de modélisation d'entreprise.

3.3.2 L'approche descendante

Pour répondre au problème présenté précédement, on peut élaborer le langage UEML en considérant une approche descendante. Cette approche est basée sur une analyse conceptuelle. Elle peut également être structurée en quatre étapes (Figure 3.3-2).

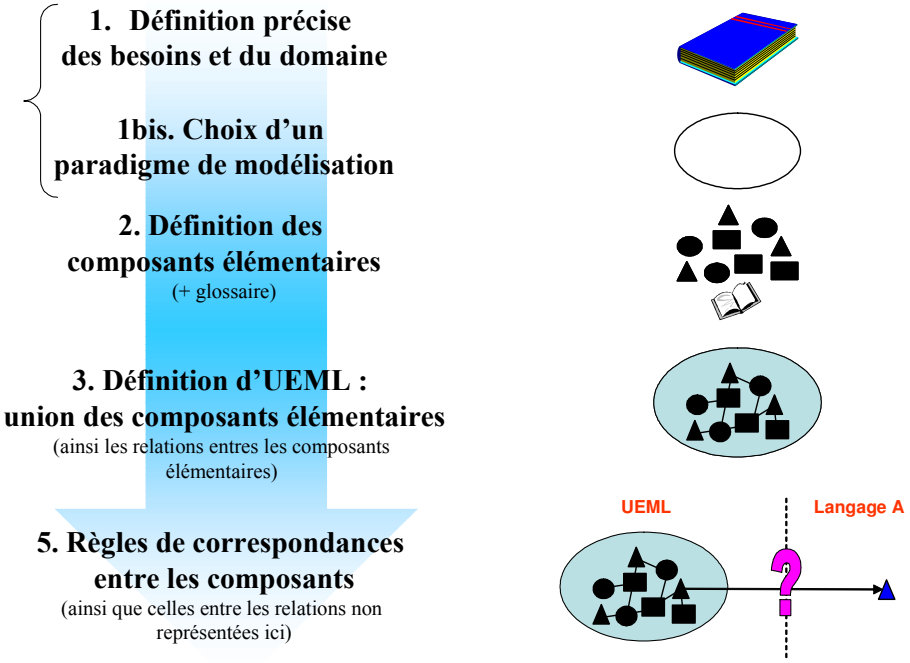


Figure 3.3-2 : Approche descendante

(1) Dans la première étape, l'important est de donner un « cahier des charges » raisonnable, concernant les fonctionnalités attendues d'UEML, ainsi que le domaine (couverture de modélisation) que l'on souhaite couvrir[2].

(1bis) Ensuite, afin de pouvoir construire notre langage unifié nous avons besoin d'un paradigme de modélisation. Ce paradigme est la base théorique sur laquelle sera basé notre langage. Un choix possible peut être d'adopter une approche systémique.

(2) Dans une prochaine étape, il est nécessaire d'effectuer une recherche conceptuelle concernant le langage UEML. Cette recherche a pour but de dégager un ensemble de composants qui nous semblent pertinents et indispensables et qui

[2] Des travaux ont été réalisés concernant les fonctionnalités du langage UEML, dans le cadre du réseau thématique UEML [Knothe *et al.*, 2003]

de ce fait, doivent trouver leur place dans UEML. Une première étape de cette recherche conceptuelle peut être de travailler, par exemple, dans le sens de l'ENV 12204, en la considérant comme une première version du méta-modèle UEML [Vernadat, 2001].

(3) Enfin, dans une dernière étape, la structure globale du langage UEML doit être définie en établissant sa syntaxe. Les différentes interactions entre les composants (associations) doivent être définies à ce niveau.

3.3.2.1 Avantage de l'approche descendante

L'avantage de l'approche descendante est la cohérence théorique du langage obtenu (la cohérence de l'approche ascendante étant, comme vu précédemment, très difficile à établir avant utilisation) qu'elle apporte.

3.3.2.2 Inconvénients de l'approche descendante

Les résultats que l'on peut obtenir avec ce type d'approche sont sans nul doute les meilleurs et les plus pertinents, bien plus que ceux que l'on pourrait obtenir avec une approche ascendante. Cependant, elle n'est, à notre sens, pas utilisable dans l'état car ce type d'approche demande un temps de développement assez conséquent. De plus, la définition des règles de correspondance entre ce langage UEML et les langages existants peut être délicate et ainsi demander beaucoup d'efforts.

3.3.3 Approche hybride

Ainsi, un compromis avantageux entre ces deux approches, consiste à employer une approche hybride, composée des étapes suivantes (Figure 3.3-3) :

(1) Définition précise des besoins fonctionnels (fonctionnalités) qu'UEML doit fournir.
(2) Choix des langages existant en cohérence avec les fonctionnalités requises et, parallèlement, choix d'un paradigme (2bis) afin d'assurer la cohérence théorique.
(3) Définition d'un ensemble de composants élémentaires obtenu par décomposition des langages choisis.
(4) L'union des composants élémentaires permet d'éliminer certaines redondances possibles parmi les composants ainsi que de définir les composants génériques correspondants au langage UEML.

(5) Etablissement de la syntaxe autour des composants élémentaires.

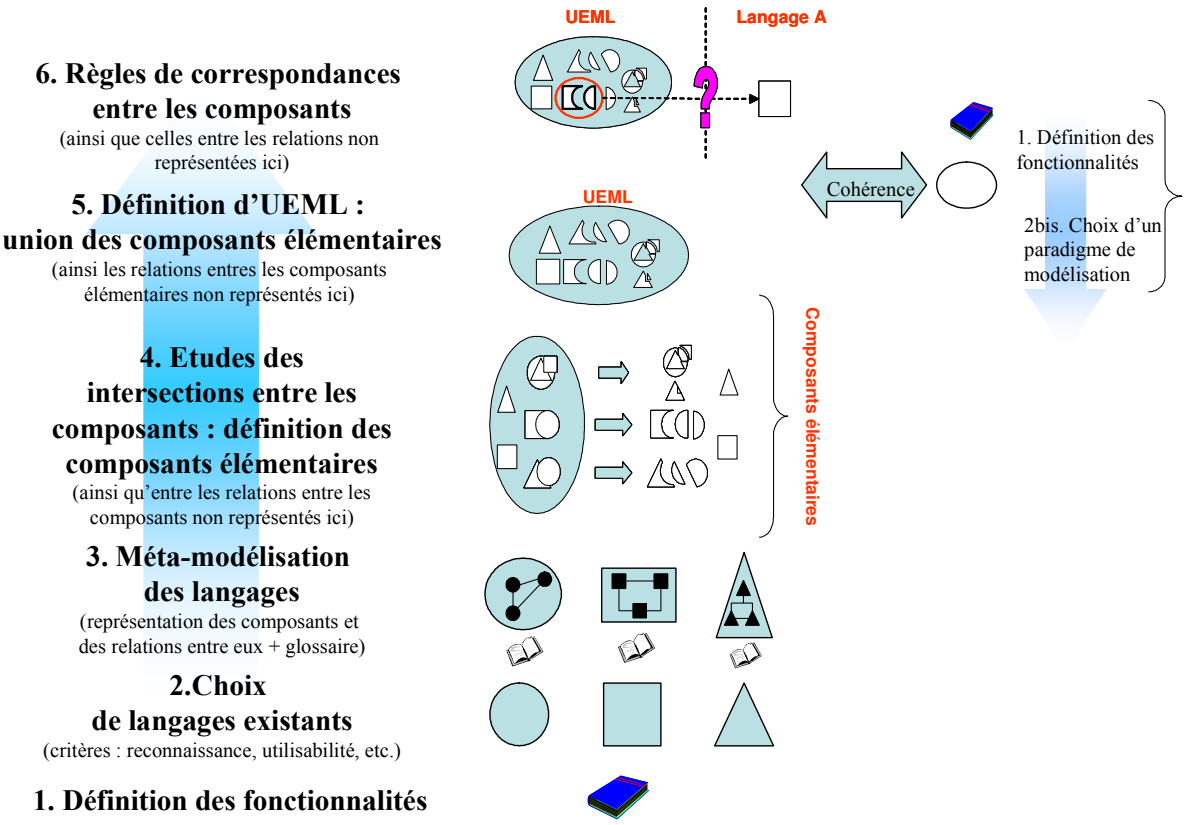


Figure 3.3-3 : Approche hybride proposée pour le développement d'UEML

Cette dernière approche semble en effet mieux adaptée au développement de notre langage UEML.

3.4 Problématique de la traduction : Approche empirique

Si l'on veut traduire les composants d'un langage dans un autre langage, il est nécessaire de comparer entre eux les composants afin d'identifier leur équivalence. Cependant, il est difficile de comparer les composants directement car beaucoup de langages de modélisation d'entreprise ne possèdent pas de définition mathématique formelle de leur sémantique [Petit *et al.*, 1997].

Ainsi, nous allons, dans ce qui suit illustrer ces difficultés au travers d'un exemple simple de traduction. Considérons, l'activité SADT (activité selon les actigrammes du langage SADT) et l'activité (d'exécution) GRAI des réseaux GRAI.

3.4.1 Le langage SADT : l'actigramme

Le langage SADT (Stuctured Analysis and Design Technique) [Vernadat, 1999] permet de représenter les aspects fonctionnels d'un système. Il permet, en effet, de décrire un système par les activités qu'il réalise, ainsi que les objets échangés entre ces activités. Ce langage est avant tout un langage de communication d'idées et adopte pour cela une syntaxe simple.
La description se fait suivant une approche systémique. Un système complexe, quel que soit sa nature, est ainsi décrit par une hiérarchie de systèmes plus simples.
De manière générale, SADT peut être utilisé dans tous les cas où l'on souhaite analyser et décrire un système sous son aspect fonctionnel. Le langage permet de répondre aux questions suivantes :

- Quelles fonctions sont mises en œuvre par le système ?
- Quels objets sont traités ou transformés par les fonctions ?
- Quels mécanismes ou ressources sont nécessaires à l'exécution des fonctions ?

L'actigramme SADT permet de décrire un système par les activités qu'il réalise et les objets échangés entre ces activités. Un modèle SADT sera ainsi constitué d'un ensemble d'activités mises en relation par les échanges d'objets.

3.4.1.1 Les activités

Les activités sont graphiquement représentées par un rectangle. Celles-ci possèdent des flux d'échange qui ont une fonction différente selon la position de leurs points de connexion : les entrées à gauche, les sorties à droite, les contrôles en haut, et les mécanismes en bas (Figure 3.4-1). Une activité s'interprète de la manière suivante : « une activité consomme des entrées (input I) pour produire des sorties (output O), à partir de directives de contrôle (control C), en s'appuyant sur les potentialités des mécanismes (mechanism M) ».

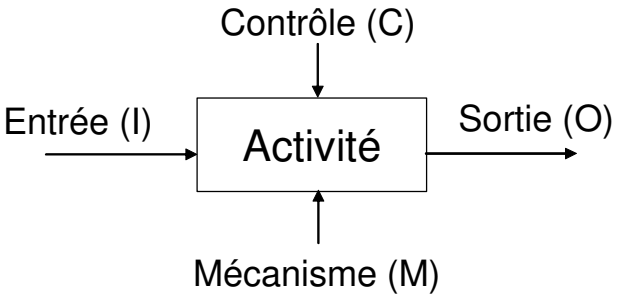


Figure 3.4-1 : Activité SADT (actigramme SADT)

Les entrées représentent les objets qui vont être transformés par l'activité. Il s'agit de la matière d'œuvre qui peut être de nature matérielle ou informationnelle.

Les sorties décrivent les objets produits par l'activité. Elles peuvent être constituées d'objets produits ou modifiés, ainsi que des rebuts générés par la transformation.

Les contrôles représentent des informations qui contraignent l'exécution de l'activité mais ne sont pas modifiés par elle. Il s'agit de règles, de directives, d'objectifs, etc.

Les mécanismes décrivent les moyens nécessaires à la réalisation de l'activité. Ils sont constitués des ressources humaines et matérielles.

Le flux circulant entre les entrées et les sorties est le flux de matière transformé par l'activité.
L'identification des activités se fait par un verbe à l'infinitif, suivi d'un complément d'objet. Ceci permet de suggérer une action, mais possède une dimension fortement subjective.

3.4.1.2 Les échanges ou inter-connectivités

Les échanges sont graphiquement définis par des flèches. Ces flèches sont orientées et connectées aux activités suivant les quatre types décrits précédemment : entrées, sorties, contrôles et mécanismes.
Ces flèches ne représentent pas des notions de causalité, mais uniquement des relations de flux entre une activité source et une activité destinataire. Une flèche peut décrire un ou plusieurs objets identiques, elle n'indique pas quand est utilisé un flux entrant, ni sa nécessité au démarrage de l'activité.

3.4.1.3 Principe de décomposition

Un des objectifs du langage est la lisibilité, afin de favoriser la compréhension et la communication au sein d'une équipe. Ainsi, tout système doit être représenté sous forme de sous-systèmes plus simples. Le principe de décomposition (Figure 3.4-2) assure cela et offre une structuration hiérarchique, partant du système dans sa globalité et en le détaillant dans les sous niveaux. Le niveau principal, noté A-0, définit globalement le système par un rectangle et une série d'entrées, de sorties, de contrôles et de supports. Il est détaillé au niveau suivant A0, par exemple en deux activités 1, 2 et 3. L'activité 2 est décrite par un niveau A2 par les

activités 21, 22 et 23. Les niveaux de modélisation peuvent être poussés jusqu'à l'obtention de niveaux de détail satisfaisants.

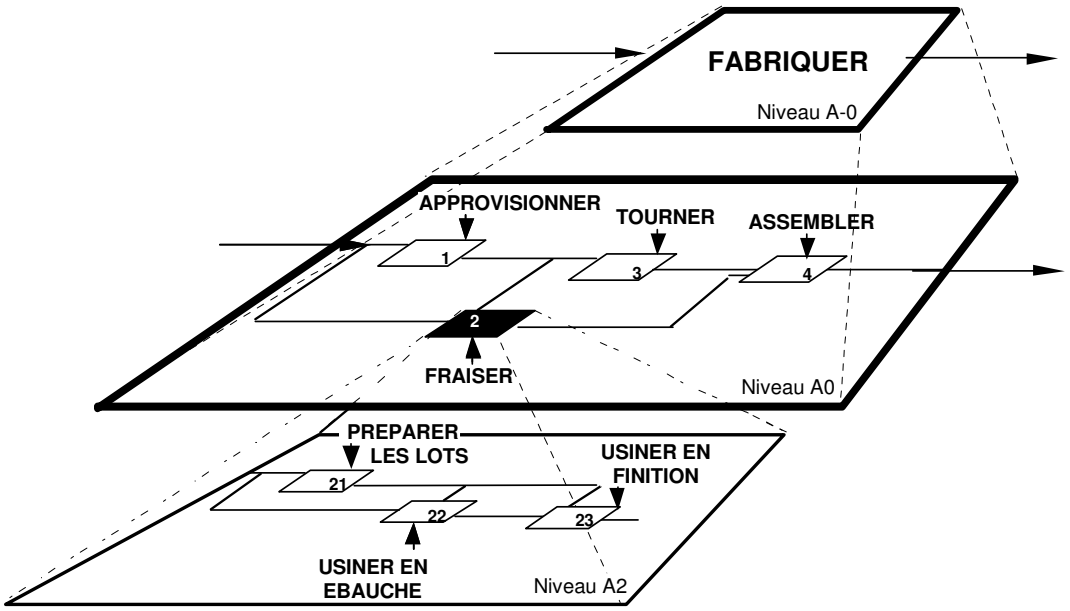


Figure 3.4-2 : Modélisation SADT multi-niveaux

Pour des raisons de lisibilité, un niveau doit contenir entre 3 et 6 activités. De même, il faut conserver la cohérence des entrées et sorties d'une activité à un niveau « n » par rapport à sa décomposition au niveau plus détaillé.

3.4.1.4 Exemple

L'exemple présenté ici est le processus correspondant à l'activité « fabriquer » comme nous pouvons le voir sur le diagramme de niveau A-0 (Figure 3.4-3).

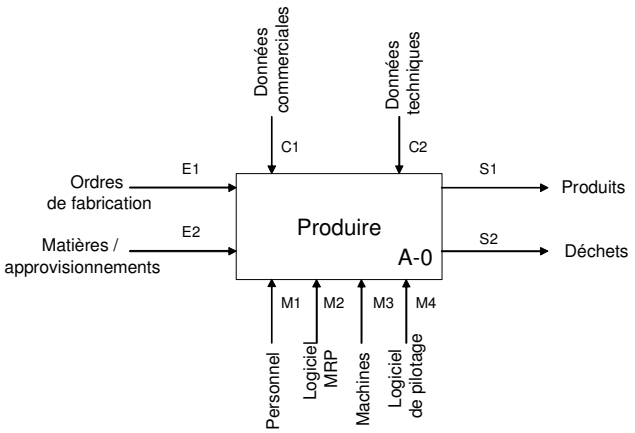


Figure 3.4-3 : Diagramme A-0

Cette activité se décompose en trois sous activités qui consistent à planifier, produire et piloter. La Figure 3.4-4 présente cette décomposition de premier niveau.

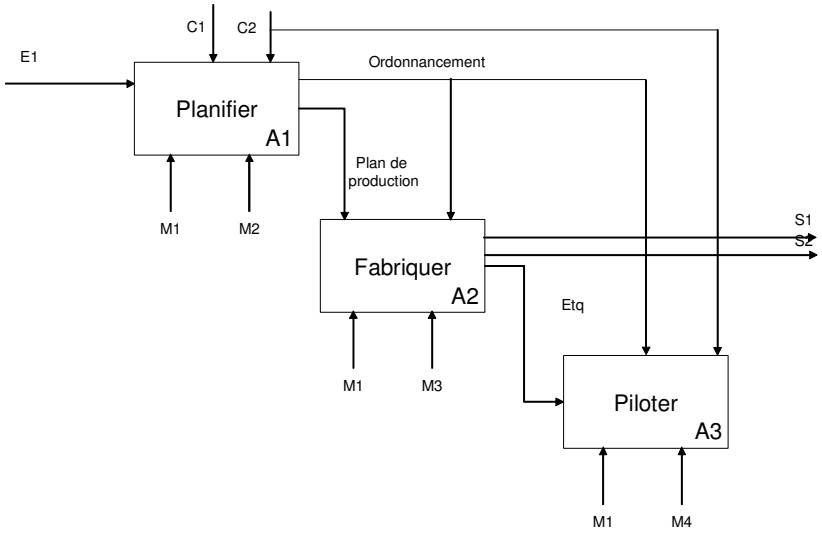


Figure 3.4-4 : Décomposition de l'activité « Produire »

La Figure 3.4-5 détaille enfin l'activité « produire », comme nous l'indique la numérotation des activités.

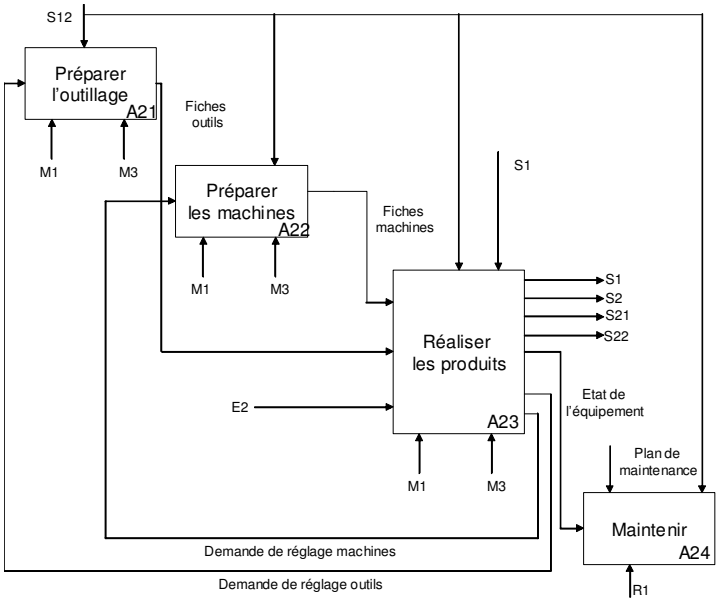


Figure 3.4-5 : Décomposition de l'activité « produire »

3.4.2 Les réseaux GRAI

Les réseaux GRAI représentent le fonctionnement de tout ou partie d'un centre de décision selon les concepts du modèle GRAI (modèle des activités d'un centre de décision [Doumeingts, 1984], [Vallespir *et al.*, 1997], [Vallespir *et al.*, 2002].

3.4.2.1 Activité GRAI

Une Activité est une partie du comportement d'un centre de décision. Une activité a un caractère dynamique et fournit une ou des entités. Les activités sont nommées et numérotées. Il existe deux types d'activités : l'activité d'exécution (Figure 3.4-6) et de décision (Figure 3.4-7).

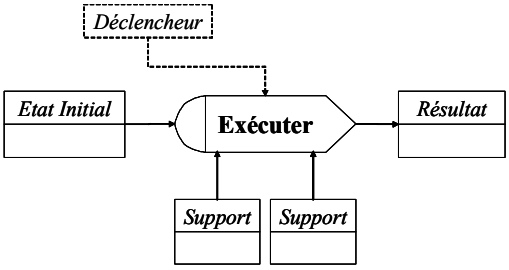


Figure 3.4-6 : Activité d'éxécution

Exécution : Activité déterministe. Activité donnant la même valeur au résultat pour les mêmes valeurs des entités convergentes (déclencheur et support).
Exemple : activité gérée complètement par une règle, procédure, programme, etc.

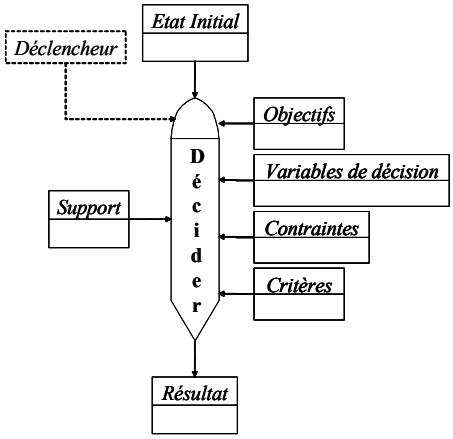


Figure 3.4-7 : Activité de décision

Décision : Activité pouvant donner plusieurs valeurs au résultat pour les mêmes valeurs des entités convergentes (déclencheur, objectifs, variables de décision, contraintes et critères).
Exemple : décision

3.4.2.2 Entités

Une activité utilise et génère des informations qui sont appelées entités (Figure 3.4-8). Une entité est donc un objet physique ou abstrait appartenant au système de conduite. Les entités sont nécessaires au déroulement des activités ou sont produites par celles-ci. *Les entités sont nommées.*

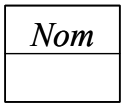


Figure 3.4-8 : Représentation graphique d'une entité

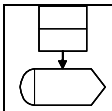
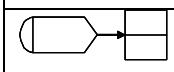
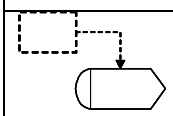
Il existe six natures de l'entité :

- **Information** : Entité de nature informationnelle quelconque.
- **Objectif** : Niveau de performance attendu suite à la réalisation de l'activité.
- **Variable de décision** : Elément sur lequel on peut jouer lors de la mise en œuvre de l'activité.
- **Critère** : aide au choix des actions sur les variables de décision.
- **Règle** : spécification du comportement de l'activité ou d'une partie de celui-ci.
- **Ressource** : moyen concret, technique, humain ou informationnel, nécessaire à la mise en œuvre de l'activité.
- **Indicateur de performance** : compte rendu sur une performance.

La nature de l'entité est précisée (ou information par défaut).

Les entités objectifs, variables de décision et critères sont uniquement et obligatoirement représentés pour les activités de décision.

Chaque entité peut jouer un rôle différent dans le déroulement d'une activité. Ainsi, trois rôles sont possibles pour les entités.

	**Support** : entité nécessaire au déroulement de l'activité.
	**Résultat** : entité produite par l'activité.
	**Déclencheur** : entité nécessaire au déroulement de l'activité et dont la mise à disposition déclenche l'activité.

Remarque sur le déclencheur :

Le déclencheur peut également être constitué par la période du niveau auquel appartient le centre de décision. Dans ce cas, aucun déclencheur n'est mentionné.

3.4.2.3 Opérateurs de renvoi

Des opérateurs de renvoi sont utilisés, lorsqu'il est nécessaire d'indiquer l'origine ou la destination d'une entité, lorsque cette origine ou cette destination est extérieure au schéma (Figure 3.4-9).

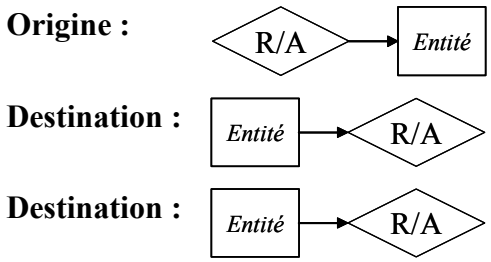


Figure 3.4-9 : Opérateur de renvoi

Situations pour lesquelles ces opérateurs sont nécessaires :

- lorsqu'une entité circule entre le système étudié et son environnement,
- lorsqu'une entité circule d'un centre de décision à un autre,
- lorsque le réseau est trop grand pour rentrer sur une page.

R : numéro du réseau,
A : numéro de l'activité émettrice ou réceptrice,
R/A : nom du système, service extérieur au domaine d'étude ou numéro du centre de décision

3.4.2.4 Opérateurs logiques

Il apparaît des combinaisons et des décompositions dans les réseaux GRAI qui se traduisent par des divergences et convergences en ET et OU des entités vers les activités et réciproquement. Ces situations sont représentées par des opérateurs logiques (Figure 3.4-10).

Les ET sont représentés par des traits doubles **(||)** et les OU par des traits simples **(|)**.

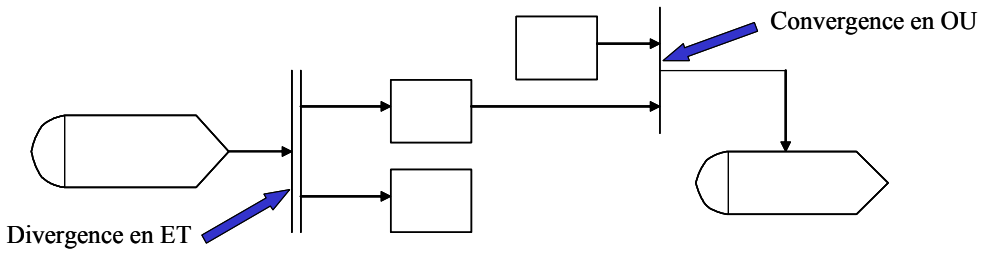


Figure 3.4-10 : Opérateurs logiques

3.4.2.5 Exemple

La Figure 3.4-11 illustre un exemple de modélisation d'un système de gestion de production d'une entreprise, par le biais des réseaux GRAI.

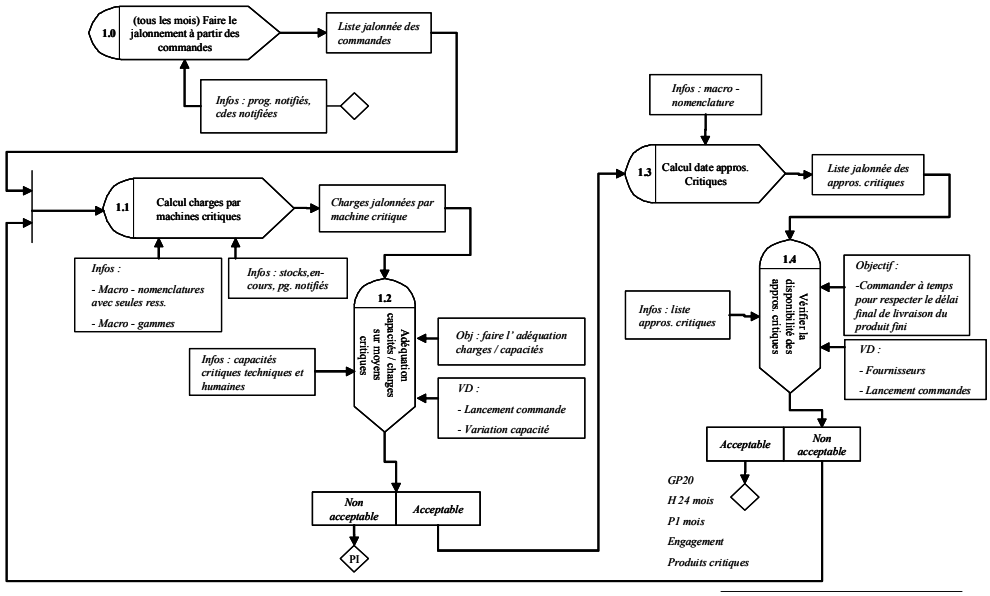


Figure 3.4-11 : Exemple de réseau GRAI

3.4.3 Traduction

Le langage GRAI et l'actigramme de SADT n'ont pour ainsi dire pas vraiment le même domaine d'application. Cependant, si les aspects méthodologiques ne sont pas pris en considération, il est clair que beaucoup de composants sont semblables dans les deux langages. C'est pourquoi, réaliser la traduction entre ces deux langages, a uns sens.

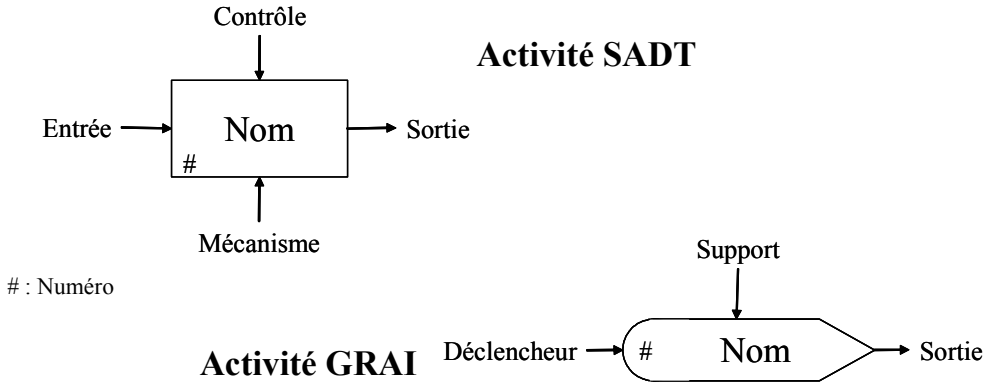


Figure 3.4-12 : Activité SADT et Activité GRAI (représentation simplifiée)

Ainsi, chacune de ces activités utilise certains composants (Figure 3.4-12). Nous pouvons les représenter de la manière suivante :

Activité SADT = {Nom, Numéro, Entrée, Sortie, Contrôle, Mécanisme}
Activité GRAI = {Nom, Numéro, Déclencheur, Sortie, Support}

Nous avons besoin à ce niveau de définir le concept de « composant élémentaire ».

Composant élémentaire pour un langage : *Un composant est un composant élémentaire pour un langage, s'il existe complètement ou pas du tout dans ce langage.*

Nous définissons ensuite un composant élémentaire d'une manière plus globale.

Composant élémentaire : *Un composant est qualifié de composant élémentaire, s'il est élémentaire pour tous les langages envisagés.*

Par exemple, prenons la Figure 3.4-13. Considérons trois langages composés d'un ensemble de composants.

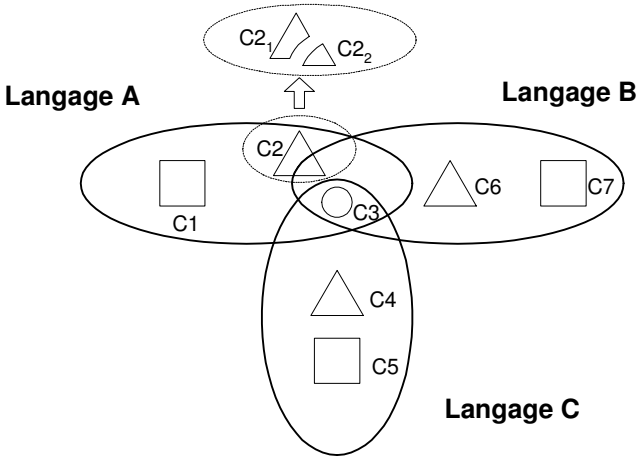


Figure 3.4-13 : Composant élémentaire

Compte tenu de notre définition d'un composant élémentaire nous pouvons dire que tous les composants sont des composants élémentaires sauf le composant C2.

En effet :

- Le composant C1 appartient entièrement au langage A et n'appartient pas du tout aux langages B et C.

- Les composants C6 et C7 appartiennent entièrement au langage B et n'appartiennent pas du tout aux langages A et C.

- Les composants C4 et C5 appartiennent entièrement au langage C et n'appartiennent pas du tout aux langages A et B.

- Le composant 3 appartient entièrement aux trois langages en même temps.

Cependant, le composant 2 appartient entièrement au langage A mais il n'appartient pas entièrement au langage B. Seulement la partie $C2_2$ appartient à ce langage. Ce composant n'est donc pas un composant élémentaire car il n'est un composant élémentaire que pour le langage A. Il sera donc nécessaire par la suite de le décomposer en composants élémentaires.
Dans notre cas, ni le composant d'activité SADT, ni celui d'activité GRAI, n'existe entièrement dans les deux langages en même temps. Ainsi, ils ne sont pas des **composants élémentaires**.

Recherche des composants communs

Afin de définir les composants élémentaires pour les deux activités, nous devons procéder à la recherche des composants communs aux deux langages, comme illustré sur la Figure 3.4-14. On peut identifier le composant d'**Activité.min = {Nom, Numéro, Sortie}** qui est un composant élémentaire puisqu'il appartient entièrement aux deux langages en même temps.

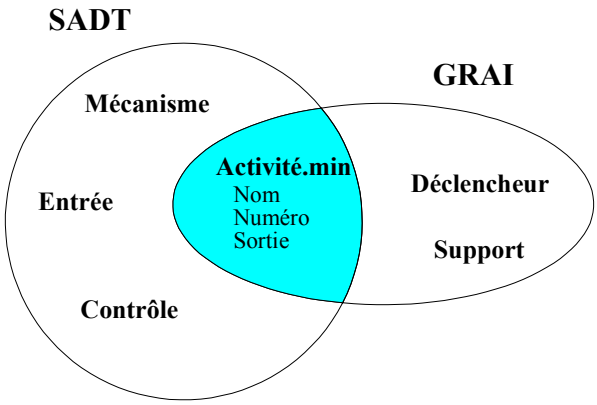


Figure 3.4-14 : Recherche des composants communs

Dans une première étape, nous pourrions conclure que les composants mécanisme, entrée, contrôle, déclencheur et support sont également des composants élémentaires, parce qu'ils appartiennent complètement ou pas du tout aux activités GRAI et SADT. Cependant, une entrée, un contrôle et un mécanisme dans SADT, sont des supports dans GRAI et un déclencheur dans GRAI est un contrôle dans SADT.
Ainsi, afin de représenter ces « règles de correspondances », nous devons ajouter un autre composant élémentaire. Ce composant s'appelle *« contrôle non déclenchant »*. Ce composant est un composant interne à UEML parce qu'il n'apparaît directement ni dans l'activité de SADT ni dans l'activité de GRAI.

Ainsi, dans une deuxième étape, en tenant compte de ces correspondances entre les composants des activités GRAI et SADT, nous pouvons conclure, que nous avons finalement seulement cinq composants élémentaires:

- Activité.min
- Déclencheur
- Mécanisme
- Entrée
- Contrôle non déclenchant

Comme UEML est une réunion, alors il contiendra tous les composants élémentaires identifiés précédemment.

Ainsi, comme illustré sur la Figure 3.4-15, ces composants élémentaires permettent de recomposer tous les composants des langages considérés.

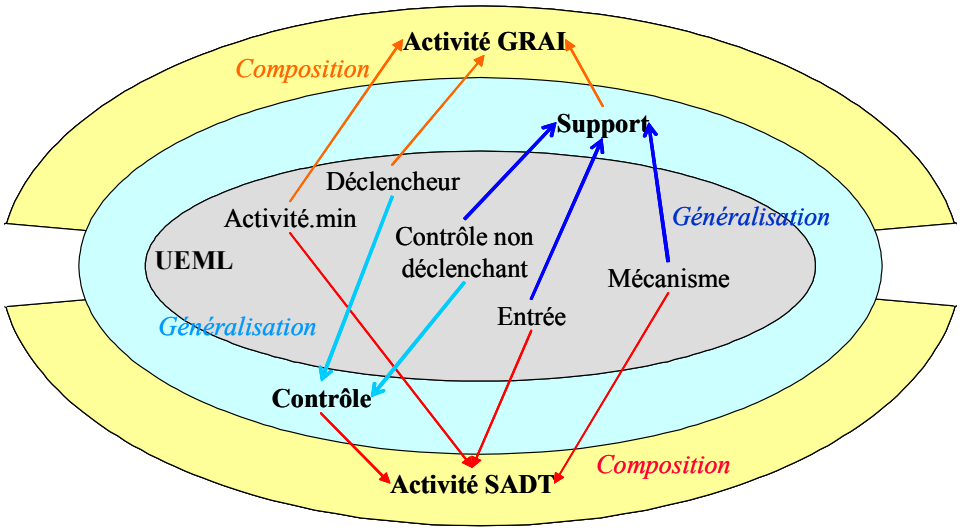


Figure 3.4-15 : Structure interne de UEML et recomposition

Plus précisément, on peut noter que, dans un premier temps les relations de généralisation permettent de reconstruire les « composants locaux » (i.e. composants appartenant à seulement un langage) à partir des composants élémentaires et, qu'en en second lieu, les relations de composition permettent d'obtenir le langage considéré. Ces relations de généralisation et de composition représentent *les règles de correspondances* entre UEML et les activités GRAI et SADT [Roque *et al.,* 2005] (Figure 3.4-16).

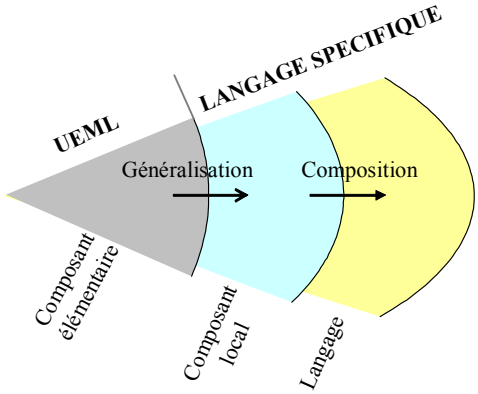


Figure 3.4-16 : Relations entre les composants élémentaires, locaux et les langages.

Une autre manière de représenter la structure de la Figure 3.4-15, est de la représenter sous une forme de liste, comme illustrée sur la Figure 3.4-17.

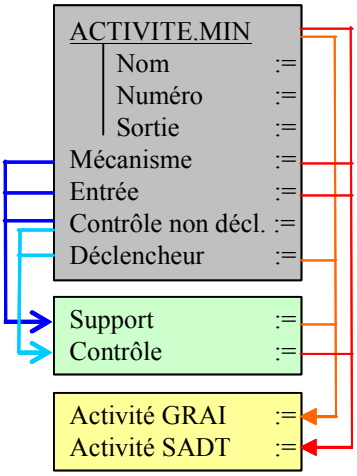


Figure 3.4-17 : Même structure sous forme de liste

3.4.3.1 Traduction SADT → GRAI

Les informations sont données dans le langage SADT, comme illustré sur la Figure 3.4-18.

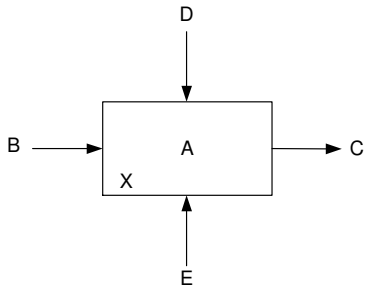


Figure 3.4-18 : Information dans le langage SADT

Traduction SADT → UEML :

Si nous projetons le modèle de l'activité SADT dans le langage UEML, nous obtenons la Figure 3.4-19 :

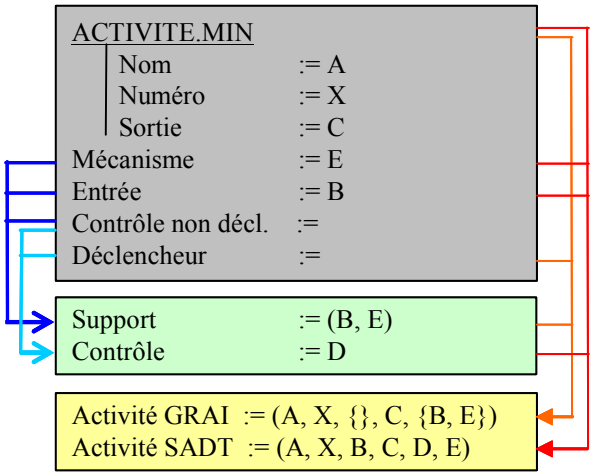


Figure 3.4-19 : Traduction de l'activité SADT en UEML

On peut noter que l'activité GRAI est automatiquement construite. Cependant, il y a une incertitude au sujet de la composition de l'activité GRAI parce le champ « déclencheur » est vide. Néanmoins, D est éligible à être déclencheur. Cette situation vient du fait que D a été seulement déclaré en tant que contrôle et non pas au niveau des spécialisations du contrôle (tel que le déclencheur).

Ainsi, si nous voulons projeter le modèle UEML sur l'activité GRAI, nous obtenons deux cas possibles (les deux spécialisations du contrôle). Il n'y a aucune information dans le modèle permettant de décider. Plus d'information est alors exigé de la part de l'analyste, afin de pouvoir compléter le modèle. On doit ainsi se poser la question suivante :

D est un Déclencheur ou un Contrôle non déclenchant ?

Traduction UEML → GRAI :

1er cas : D est un contrôle non déclenchant.

Si D est finalement un contrôle non déclenchant, il devient alors un support (par la deuxième relation de généralisation) et est ajouté à l'activité GRAI en tant que tel. La structure est mise à jour comme illustré sur la Figure 3.4-20 et, en conclusion, la traduction de l'activité SADT dans l'activité GRAI donne le résultat également représenté sur la Figure 3.4-20.

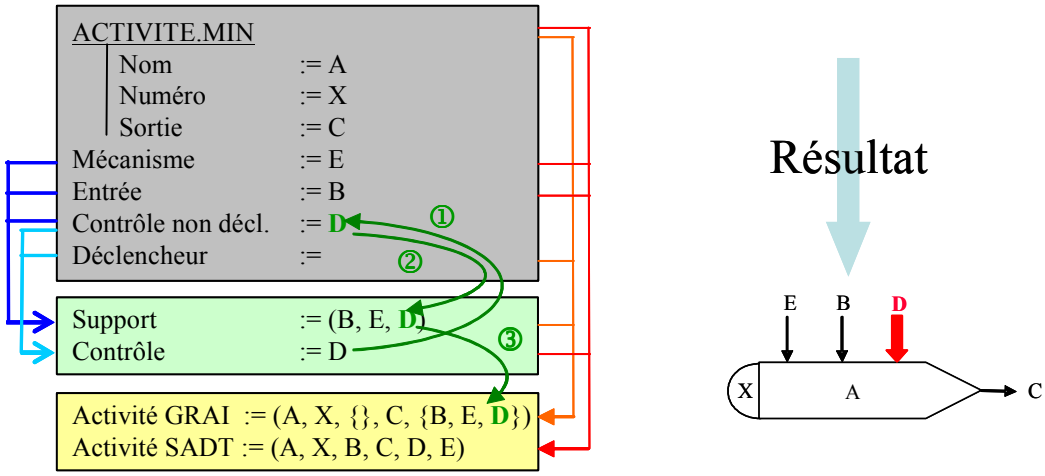


Figure 3.4-20 : Le résultat dans le cas où D est un contrôle non déclenchant

Le résultat est syntaxiquement correct car le composant déclencheur est optionnel pour une activité GRAI.

2ème cas : Le contrôle est un déclencheur dans le langage GRAI.

Si la réponse à la question est maintenant que D est un déclencheur alors la structure est mise à jour comme illustrée sur la Figure 3.4-21. Le résultat de l'activité GRAI correspondante est présenté également sur cette figure.

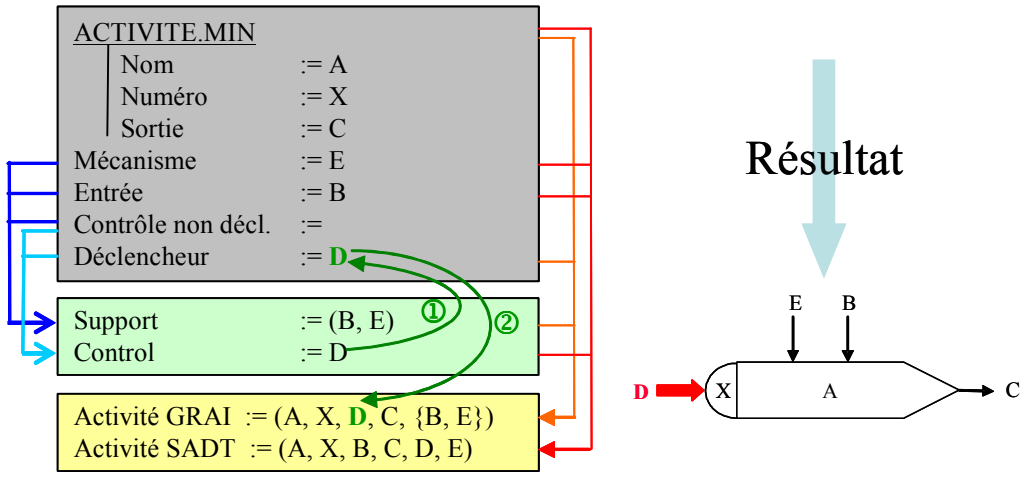


Figure 3.4-21 : Le résultat dans le cas où D est un déclencheur

3.4.3.2 Traduction GRAI → SADT

Les informations sont données dans le langage GRAI, comme illustré sur la Figure 3.4-22.

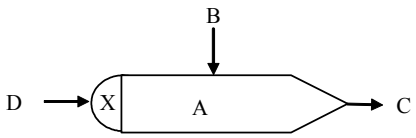


Figure 3.4-22 : Information dans le langage GRAI

Traduction GRAI → UEML :

Si nous projetons le modèle de l'activité GRAI dans le langage UEML, nous obtenons la Figure 3.4-23.

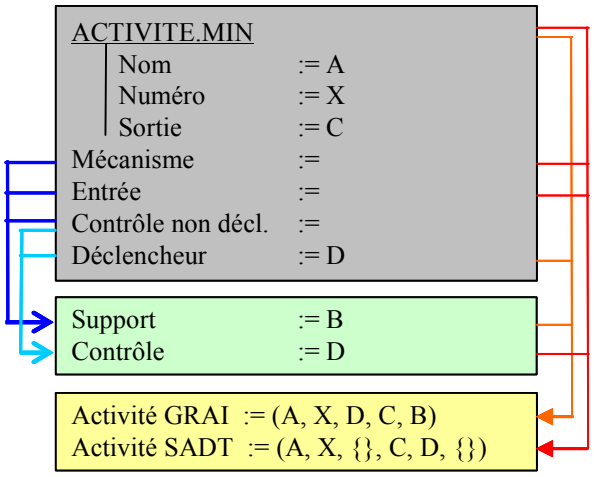


Figure 3.4-23 : Traduction de l'activité GRAI en UEML

Traduction UEML → SADT :

Si nous voulons maintenant traduire cette structure en activité SADT, nous sommes dans la même situation que précédemment : B est un support, mais nous ne savons pas quelle spécialisation de support reçoit B comme instance. B est éligible à être entrée, contrôle ou mécanisme. Supposons que l'analyste déclare que D est une entrée. La structure est alors mise à jour et la traduction dans l'activité SADT est présentée sur la Figure 3.4-24.

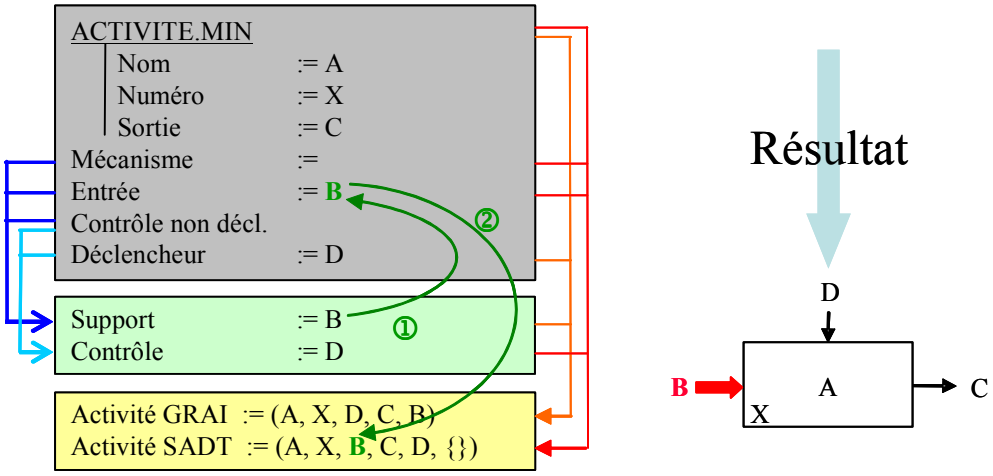


Figure 3.4-24 : Résultat dans le cas où B est une entrée

Le résultat est syntaxiquement correct car le composant de mécanisme est optionnel pour une activité SADT.

3.5 Conclusion

Nous avons dans ce chapitre, exposé nos positons et points de vue concernant l'élaboration d'un langage unifié de modélisation d'entreprise. Nous avons également présenté trois démarches différentes afin de construire le langage UEML. Nous avons ainsi insisté sur la nécessité de spécifier les fonctionnalités attendues pour UEML pour sa définition Enfin, nous avons traité un exemple très simple de traduction. Grâce à lui, on peut dégager un certain nombre de difficultés et points spécifiques à la traduction.

1. **Difficultés pour définir les composants élémentaires :**

   La définition et la recherche de composants peuvent être des opérations délicates. Nous allons voir dans le chapitre 4, les différentes situations qui peuvent se présenter lorsque l'on veut comparer deux composants, et comment on peut définir des composants élémentaires, d'une manière plus formelle.

2. **Certains composants peuvent être des sur-classes d'autres composants :**

   Nous avons vu qu'il existe une certaine hiérarchie entre les composants d'un point de vue relation de la généralisation (généralisation multiple ou non). Nous verrons par la

suite qu'il existe des relations plus complexes entre les composants. Par ailleurs, compte tenu de notre définition du composant élémentaire, toutes ces relations ne représentent que les règles de correspondance (règles de traduction) qui existent entre le langage UEML et les langages considérés.

3. **Il existe des composants propres à UEML (n'existant pas en tant que tel dans aucun langage opérationnel) :**

La recherche de composant élémentaire peut mener à la création de composants élémentaires, qui n'appartiennent à aucun des langages considérés (tel que contrôle non déclenchant). Ces composants sont internes au langage UEML, et n'apparaissent donc pas dans les langages en tant que tel. Ils permettent de « reconstruire » des composants plus complexes appartenant aux langages opérationnels.

4. **Renseignement partiel d'un modèle dans un langage opérationnel à partir d'un modèle dans un autre langage opérationnel : projection**

En effet, même si l'ensemble des informations provenant de modèles écris dans plusieurs langages différents sont stockés dans le noyau du langage UEML, il peut y avoir cependant une « perte d'informations » lorsque l'on projette certains composants d'un modèle UEML sur un langage, si celui-ci ne prend pas en compte ces composants là (Figure 3.5-1).

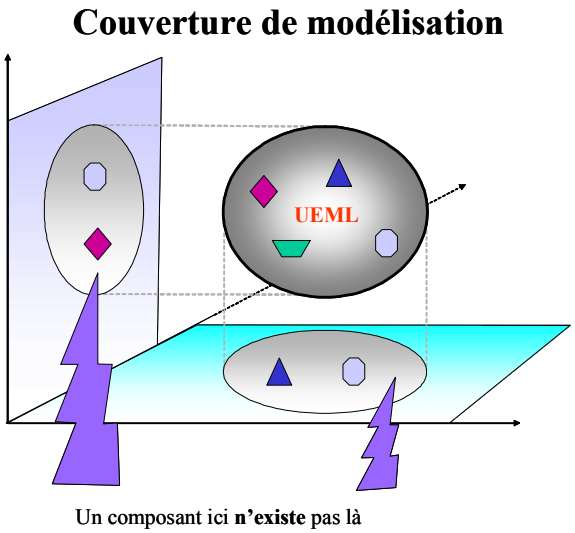


Figure 3.5-1 : Couverture de modélisation

Nous avons vu également, grâce à cet exemple, que dans certain cas la traduction est automatique et que dans d'autres, il est nécessaire que l'analyste fasse un choix. En effet, comme illustré sur la Figure 3.5-2, ce problème de choix apparaît lorsque l'on veut projeter un langage dans UEML.

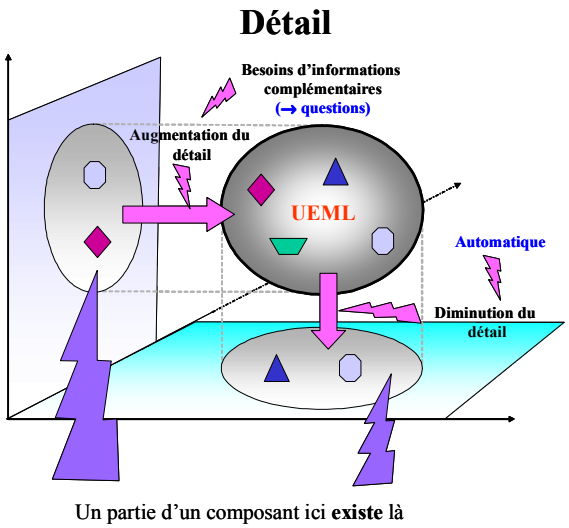


Figure 3.5-2 : Couverture et détail de UEML

Dans ce cas, il y a augmentation de la quantité de détail nécessaire car UEML est plus détaillé que tous les autres langages (donc besoin d'informations complémentaires). Dans le cas, où on projette un modèle UEML sur un langage, la traduction peut se faire de manière automatique, puisqu'il y a diminution du détail. UEML est donc à la fois plus vaste (car c'est une réunion) et plus détaillé également que tous les autres langages impliqués[3].

Enfin, nous voyons au travers de cet exemple, que des composants qui sont finalement assez similaires, peuvent être représentés d'un manière totalement différente au sein des langages opérationnels. Il nous semble ainsi important de parler de modèle et de modélisation. Nous allons pour cela dans le chapitre suivant, présenter ce que l'on entend par modèle et ceci au travers de la science de la sémiotique, qu'il nous semble important de prendre en compte pour traiter notre problématique.

[3] En général, on a tendance à penser que plus le domaine considéré est grand moins on perçoit de détail.

Chapitre IV

Méta-modélisation et écriture formelle

Chapitre IV
Méta-modélisation et écriture formelle

4.1 Introduction

Dans le chapitre précédent, nous avons traité un exemple de traduction de composants. Cette traduction a été faite d'une manière empirique. Dans ce chapitre, nous allons présenter les concepts relatifs à la méta-modélisation et nous montrerons également qu'il est important de considérer les aspects du triangle sémiotique et donc de considérer les concepts de signifiant, de signifié et de référent. Ainsi, nous dégagerons certaines situations que l'on rencontre lors de la comparaison des composants des langages. Nous clarifierons ces situations à l'aide d'une approche ensembliste. Ensuite, nous reprendrons notre exemple de traduction et nous le traiterons d'une manière plus formalisée. Nous montrerons aussi, qu'il est également important de se diriger vers l'utilisation de langages plus formels, afin d'aider à la méta-modélisation des langages. Ensuite, nous distinguerons un autre type de comparaison qui est celle des interactions (associations) qui peuvent exister entre les composants.

4.2 La modélisation et la méta-modélisation

4.2.1 Modèle et modélisation

Tout modèle (résultats d'un acte de modélisation) est décrit à l'aide d'un langage (graphique, mathématique etc.) de modélisation qui est *« une représentation de la connaissance à communiquer sans ambiguïté. Il permet de construire des modèles selon des composants associés »* [Doumeingts, 1990].

Les **composants** de modélisation devraient avoir une sémantique clairement définie, ils sont nécessaires pour effectuer une **interprétation** des **langages** de modélisation, tandis que ces derniers ont pour objet de permettre une **description** de ces composants.

Ainsi, un modèle est une vision conceptuelle (une représentation) d'un système. Ce modèle est constitué d'un ensemble de composants qui décrivent des entités du système. La correspondance entre ces composants et les objets du monde réel qu'ils représentent a été appelée « triangle sémiotique » par les linguistes. Un modèle est constitué d'un ensemble d'objets (les signifiants) qui, de par leur sens (les signifiés), désignent certains objets du monde réel.

En effet, F. de Saussure [Ferdinand de Saussure, 1983] décrit le « signe linguistique » comme une entité psychique comportant deux faces indissociables (une réalité bi-face) : *un signifiant* (les sons ou leur transcription écrite, la partie sensible) et *un signifié* (le concept, la partie abstraite). Ainsi, par exemple, les lettres imprimées sur cette page **ch- a-t /chat/** (signifiant) évoquent, pour celui qui comprend le français l'idée de «chat» (signifié), cet animal familier à poil doux (Figure 4.2-1).

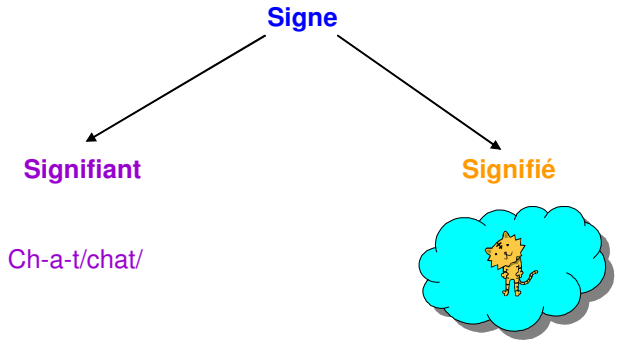


Figure 4.2-1 : Relation entre le signifiant et le signifié

En conséquence, le signifiant, le signifié et le référent constituent les sommets de ce triangle sémiotique. Ce dernier étant défini comme « l'entité physique du monde réel » identifiée par le couple « signifiant, signifié » (Figure 4.2-2).

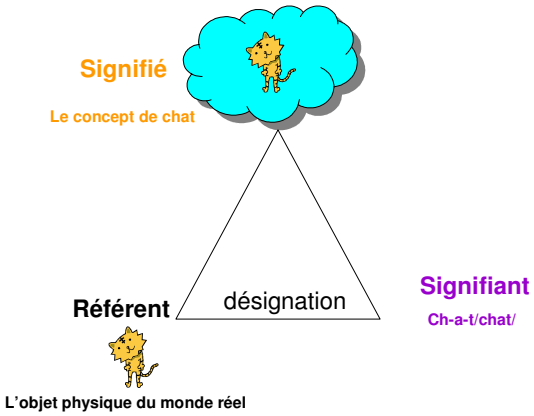


Figure 4.2-2 : Le triangle sémiotique

Ainsi, comme le précise [Morand, 1995], *« Description n'est pas concept : la carte routière représente mais n'exprime pas le trajet. Elle n'indique pas que l'autoroute comparée à la route départementale, peut constituer un trajet d'une durée plus courte bien qu'elle montre une distance plus longue entre deux points ».*

Dans les domaine des systèmes d'information, l'approche de FRISCO [Falkenberg et al., 1998] a pour but d'établir le lien entre la « réalité » et ses concepts de modélisation. Elle est basée sur le triangle sé*miotique.* Celui-ci a été prolongé en un tétraèdre en rajoutant la dimension « acteur ».

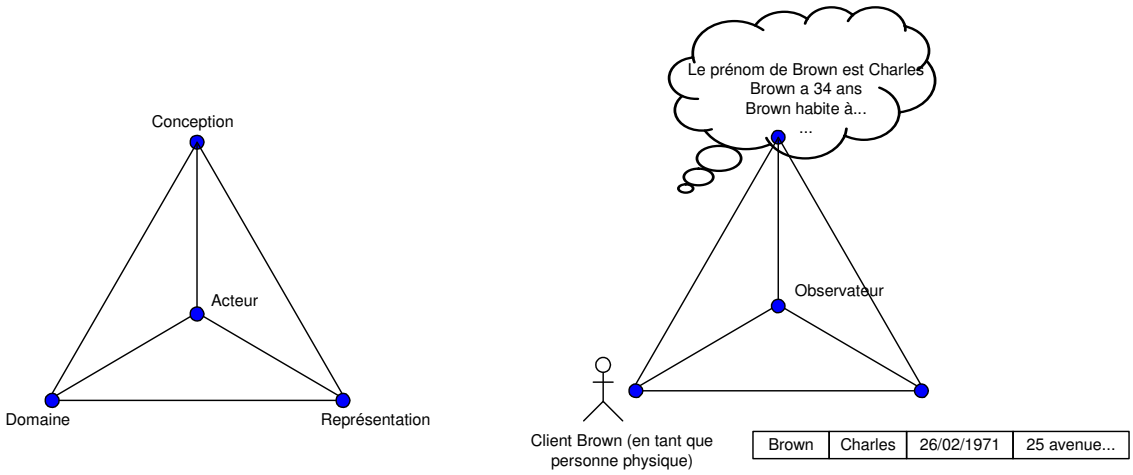


Figure 4.2-3 : Le tétraèdre FRISCO

Ainsi, les concepts dégagés par l'approche FRISCO sont :

- **Domaine** : le monde autour de l'analyste.
- **Acteur** : un acteur percevant et concevant l'univers, en utilisant ses sens.
- **Conception** : vision de l'analyste, quand il observe l'univers, en utilisant sens, et interprètent ce qu'il perçoit.
- **Représentation** : le résultat d'un analyste dénotant une conception, en utilisant un certain langage et moyen pour s'exprimer.

Ainsi, en modélisant une partie du monde, les analystes sont influencés par leurs intérêts particuliers pour l'univers observé.

4.2.2 Méta-modélisation

On parle de méta-modélisation dès lors que l'acte de modélisation s'applique sur des modèles. Un acte de méta-modélisation a les mêmes objectifs qu'un acte de modélisation avec pour seule différence l'objet de la modélisation.

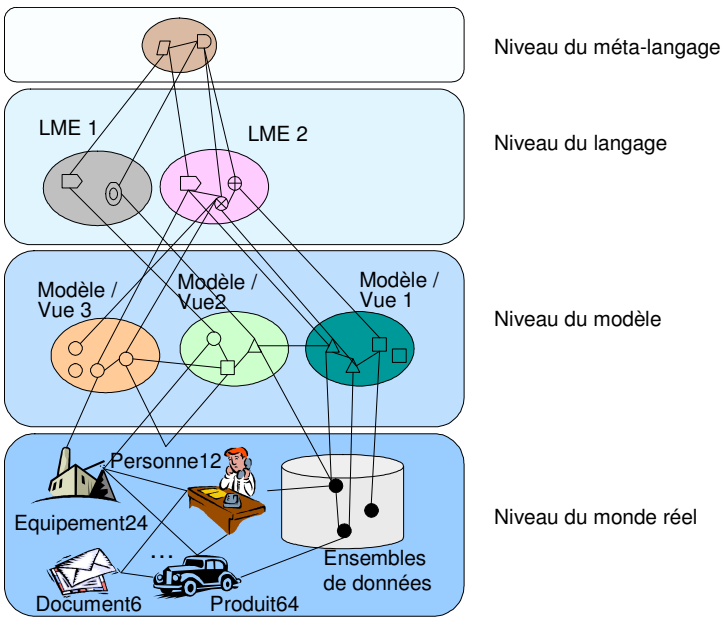


Figure 4.2-4 : Monde réel, modèle et méta-modèle [Berio *et al.*, 2004]

Ainsi, on peut voir sur la Figure 4.2-4, que l'on peut représenter l'acte de méta-modélisation en plusieurs niveaux. Le premier niveau, correspond aux données de base et aux objets du monde réel, par exemple les constituants physiques d'un système, les données d'une simulation etc. Ces données sont ensuite utilisées afin de construire un modèle des objets du monde réel (niveau 2). Ce modèle est écrit dans un langage (niveau 3) qui peut être décrit à l'aide d'un langage de méta-modélisation (niveau 4).

La méta-modélisation concerne l'analyse systématique d'un langage dans le but d'en spécifier clairement les composants, d'après [Oussalah, 1997] elle peut être utilisée comme :

- technique de dialogue, dans ce cas elle sert de moyen de comparaison et d'unification de langages,

- technique de réflexivité, elle permet alors à un langage de s'auto-représenter,

- technique d'ingénierie, pour formaliser des langages semi-formels.

Nous effectuons ici une méta-modélisation dans l'optique de définir un langage unifié de modélisation d'entreprise. Nous utilisons donc la méta-modélisation comme une **technique de dialogue**, en décrivant les composants des différents langages, l'objectif étant de *« dépasser les problèmes terminologiques et les conventions graphiques afin d'identifier leurs points communs et ceux divergents »* [Oussalah, 1997].

L'étude des intersections entre langages en vue de leur unification, est grandement facilitée par la méta-modélisation. L'expression des composants de modélisation dans un langage unique « doit » lever toute ambiguïté sur leur définition, et ainsi les liens sémantiques et syntaxiques des différents langages sont alors plus facilement mis à jour.

La Figure 4.2-5 illustre comment le triangle sémiotique peut s'appliquer au niveau de la méta-modélisation.

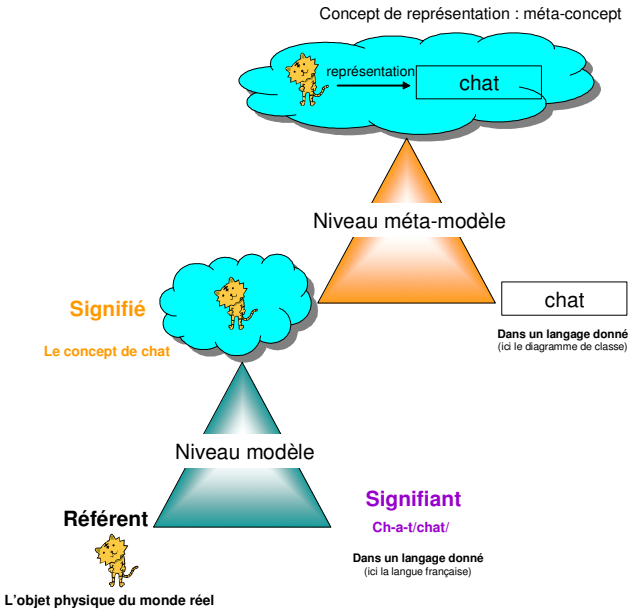


Figure 4.2-5 : Sémiotique et méta-modélisation

Ainsi, nous allons utiliser une approche par méta-modélisation pour décrire les capacités des langages de modélisation d'entreprise sélectionnés, ainsi que le langage UEML lui-même. Chaque langage doit être défini au travers d'un méta-modèle. Dans notre cas, le langage utilisé afin de décrire ces méta-modèles est donc unique et, de plus, est différent des autres langages.

Le langage utilisé afin de réaliser ces méta-modèles est le diagramme de classes du langage UML (Unified Modellling Langage) [OMG, 2003] car il semble suffisant pour traiter notre problématique qui est, dans un premier temps, de décrire les aspects syntaxiques *des langages.*

4.2.3 Méta-modèles concernant notre exemple

Si nous reprenons notre exemple concernant la traduction entre une activité GRAI et SADT nous pouvons, comme illustré sur la Figure 4.2-6, construire les méta-modèles simplifiés de ces activités. Ces méta-modèles sont simplifiés car ils ne font apparaître que les différents composants que l'on retrouve dans les deux types d'activités. Les liens entre ces composants ne sont pas représentés.

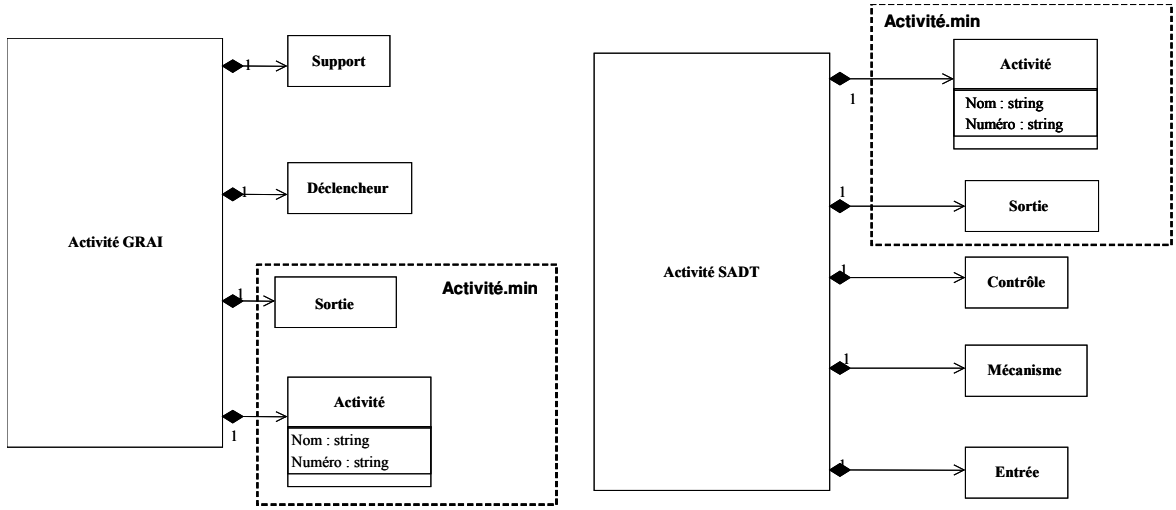


Figure 4.2-6 : Méta-modèle simplifié de l'activité SADT et GRAI

Remarque sur la méta-modélisation :

L'acte de modélisation (ou de méta-modélisation) n'est pas une activité que l'on pourrait qualifier de déterministe et il faudra donc faire attention à cet aspect lorsque l'on va vouloir comparer les composants de différents langages à l'aide de leur méta-modèles. De plus, le nom d'une classe (ou d'une association) est en général polysémique et ne permet donc pas de déterminer ce qu'elle dénote de manière univoque. C'est ainsi, qu'un même nom de classe peut avoir plusieurs sens différents. Or, le contexte donné par le diagramme de classe, ne permet pas toujours de choisir parmi ces sens, celui qui est le plus adéquat. Par ailleurs, sans méthode de représentation du sens, il est difficile de représenter correctement les classes (leur attribuer des propriétés et des associations). C'est pourquoi, il ne faut pas dissocier de chaque classe le sens dans lequel elle est utilisée (signifié) [cf paragraphe 4.2.1]. Nous reviendrons également sur ces aspects au paragraphe 4.3.7.7.

4.3 Comparaison de composants

Avant de procéder à la comparaison des composants des différents langages il nous semble important de préciser « ce qui doit être comparé ». Pour cela, nous allons utiliser le tétraèdre donné par FRISCO. En effet, dans notre idée et comme le précise [Parent *et al*, 2000] pour ce qui concerne les bases de données, « il faut dépasser les représentations, pour considérer en premier lieu ce qui est représenté plutôt que comment il est représenté ». Par exemple, nous voulons savoir si le composant d'activité, représenté dans le langage 1, est aussi représenté dans le langage 2, même si les deux composants possèdent des attributs complètement différents. Par conséquent, nous dirons que **deux composants appartenant à deux langages**

différents ont quelque chose en commun si les portions de monde réel qu'ils représentent ont des éléments communs (i.e. une intersection non vide), ou ont des éléments en relation entre eux par un lien d'intérêt pour des applications futures. Ainsi, si on regarde ce problème avec le tétraèdre donné par l'approche FRISCO, alors ceux que nous devons comparer entre eux sont les « domaines (les référents en sémiotique) » des composants. En effet, les langages de modélisation d'entreprises ont été développés en ayant une certaine vision de l'entreprise. Ainsi, les concepteurs du langages ont eu une « conception » de l'entreprise liée, par exemple, à leurs centres d'intérêts et à leurs discipline (e.g. théorie des systèmes, génie logiciel, etc.). Cette conception de l'entreprise est constituée d'un ensemble de concepts que nous appelons ici composants ayant certaines interactions entre eux. Enfin, ces composants sont en général représentés à l'aide d'un langage graphique, ou d'un langage mathématique. Ainsi, les différentes conceptions d'un même objet du monde réel ne doivent pas rentrer en compte, au moins dans un premier temps, dans la comparaison des composants des langages.

4.3.1 Taxonomie des conflits

Comme nous l'avons déjà abordé dans l'état de l'art de ce mémoire, des travaux ont déjà été effectués dans le domaine de l'intégration des bases de données. Dans [Parent *et al.*, 2000], nous pouvons trouver une taxonomie des conflits que l'on rencontre le plus souvent dans cette tâche d'intégration (Tableau 4.3-1).

Conflits de nom	Des composants différents possèdent le même nom (conflit d'homonymie) et des composants équivalents possèdent des noms différents (conflit de synonymie)
Classification	On a un conflit de classification lorsque les types en correspondance décrivent des ensembles différents, mais sémantiquement liés, d'éléments du monde réel. Un conflit de classification se traduit par l'utilisation d'une relation ensembliste autre que l'équivalence ($\subset$, $\cap$, $\neq$).
Fragmentation	Le même objet du monde réel est décrit au travers de différentes décompositions d'éléments plus élémentaires.
Description	Les composants ont des ensembles différents de propriétés et/ou leurs propriétés sont représentées de manière différente.
Structure	Un conflit structurel survient lorsque les éléments en correspondance sont décrits par des composants de niveaux de représentation différents ou soumis à des contraintes différentes : par exemple, une classe d'objets et un attribut, ou un type d'entité et un type d'association.
Hétérogénéité	Les modèles de données utilisés sont différents.

Tableau 4.3-1: Extrait de la taxonomie des conflits pour l'intégration des bases de données

Des solutions existent pour traiter de manière plus ou moins efficace ce genre de conflits. Pour plus de détails sur ce sujet, nous invitons le lecteur à se référer à [Parent *et al.*, 2000].

4.3.2 Comparaison de deux composants

Ainsi, en prenant en compte les enseignements obtenus grâce à notre exemple de traduction du chapitre précédent et compte tenu de notre étude du domaine des bases de données, nous pouvons finalement faire ressortir un certain nombre de cas que l'on peut rencontrer lorsque l'on veut comparer deux composants. Les différentes situations sont exposées ci-après. Nous comparons dans ce qui suit deux composants A et B appartenant respectivement à deux langages de modélisation différents et nous indiquons quel(s) composant(s) nous devons intégrer dans le langage UEML.

Cas 1 : Les composants n'ont pas de rapport

Les deux composants à comparer ne sont pas du tout de même nature et ne possèdent donc pas de points communs.

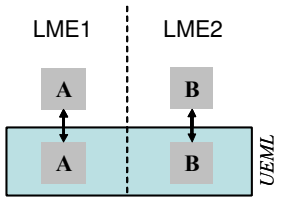


Figure 4.3-1 : Aucun rapport

Langage UEML :

Les deux composants A et B appartiennent tel quel au méta-modèle UEML (Figure 4.3-1).

Case 2 : Les composants sont équivalents (A ≡ B)

Les deux composants à comparer sont totalement équivalents.

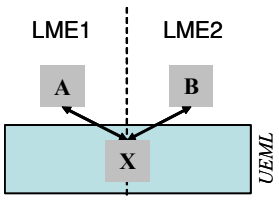


Figure 4.3-2 : Equivalence

Langage UEML :

Le composant X appartenant au méta-modèle UEML est créé (Figure 4.3-2). La règle de correspondance est dans ce cas :

$$X \equiv A \equiv B$$

Cas 3 : Le composant A est entièrement une partie du composant B

L'un des deux composants est une partie de l'autre composant.

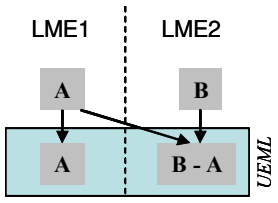


Figure 4.3-3 : Entièrement une partie

Langage UEML :

Les composants A et le composant X ≡ (B - A) appartiennent à UEML (Figure 4.3-3). Ces composants permettent de recomposer les composants A et B mais les règles de correspondances sont plus difficiles à écrire et elles seront détaillées dans les paragraphes suivants. Les flèches indiquées ici ne font que spécifier une certaine dépendance des composants entre eux.

Cas 4 : Les composants A et B possèdent une partie commune.

Il existe dans ce cas une partie commune aux deux composants à comparer.

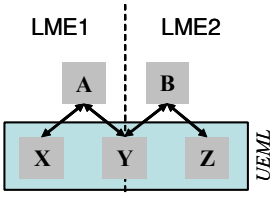


Figure 4.3-4 : Partie commune

Langage UEML :

Trois composants X, Y et Z (Y ≡ partie commune, X ≡ A - Y, Z ≡ B - Y) sont créés dans UEML (Figure 4.3-4). Ces composants permettent de recomposer les composants A et B mais dans ce cas également les règles de correspondances sont plus difficiles à écrire. Les doubles flèches indiquées ici ne font également que spécifier une certaine dépendance des composants entre eux

Il est notable à ce niveau que le cas 3 est une version simplifiée du cas 4. Cependant, dans un souci de clarté, nous continuerons dans la suite à distinguer ces deux cas.

4.3.3 Détail de la méta-modélisation et comparaison de composants

La quantité de détails nécessaires dans un méta-modèle est un autre problème qui se pose lors de la méta-modélisation des différents langages. En effet, la nécessité de réaliser le méta-modèle d'un langage avec un niveau de détail plus ou moins élevé ne se justifie que lorsque l'on veut comparer au moins deux langages. Nous considérons que les méta-modèles des langages sont réalisés par des experts de ces langages et donc qu'ils ne véhiculent pas d'erreurs par rapport aux langages.

Ainsi, dans ce cadre là, nous avons pour les cas 3 (A est entièrement une partie de B) et cas 4 (A et B possèdent une partie commune) du paragraphe 4.3.2 (Figure 4.3-5), deux situations différentes qui peuvent se présenter :

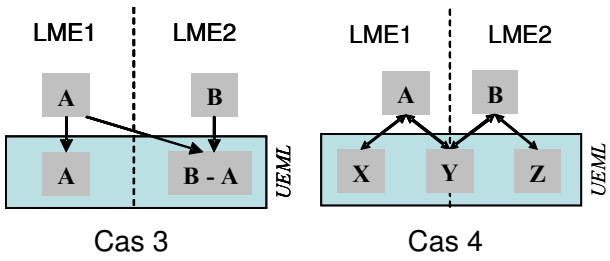


Figure 4.3-5 : Cas 3 et cas 4

1. Soit les deux composants ne sont pas assez détaillés dans les méta-modèles, mais des composants plus détaillés *sont tout de même définis* dans le « glossaire » du langage.

2. Soit des composants plus détaillés *ne sont pas définis* dans le « glossaire » des langages considérés, mais ces composants existent tout de même, c'est à dire que ces composants appartiennent à un domaine contenu dans la couverture de modélisations des langages.

4.3.3.1 Méta-modèles non assez détaillés

Ainsi, pour les cas 3 et 4, la question qu'il faut se poser est « existe t-il un ensemble de composants plus détaillés représentant B », B étant ici une vision agrégée de ces composants. Ainsi, nos deux cas n'apparaissent ici que parce que l'analyste n'a pas représenté les méta-modèles avec un niveau de détail suffisant, afin de permettre une comparaison correcte des composants des deux langages.

En effet, dans un premier temps, chaque méta-modèle est réalisé isolément sans se soucier réellement de ce problème de détail, ils sont réalisés avec un niveau de détail assez peu élevé. Le raffinement de ces méta-modèles ne vient qu'avec la comparaison d'au moins deux langages. La Figure 4.3-6 ci-dessous, illustre qu'une relation de type « est entièrement une partie de » peut n'être induite que par une méta-modélisation avec une niveau de détail trop faible.

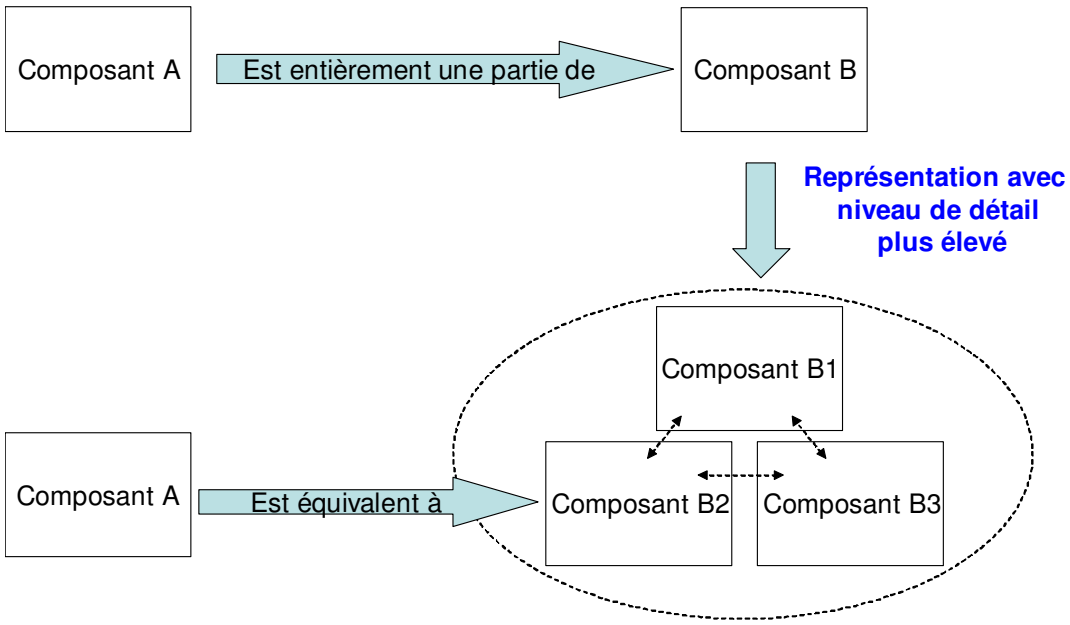


Figure 4.3-6 : Détails et méta-modélisation

En effet, le composant A est dans un premier temps une partie entière du composant B. Si on effectue une méta-modélisation avec un niveau de détail plus élevé, on s'aperçoit que le

composants B n'est en fait qu'une vision agrégée de trois composants B1, B2 et B3. Dans ce cas-ci, le composant A est finalement lié par une simple relation d'équivalence avec le composant B2.

Ainsi, avant de procéder à la comparaison des composants, il faut que les méta-modèles des langages soient « *homogènes* » d'un point de vue du niveau de détail représenté.

Il est tout de même important de noter que dans le cas de la Figure 4.3-6, le fait de représenter le composant B avec un niveau de détail supplémentaire ne fait en réalité que ***« déplacer »*** *le problème. En effet, d'autres relations peuvent apparaître entre les composants B1, B2 et B3. Le problème précédent était un problème relatif à deux langages et on l'a par la suite déplacé afin qu'il n'apparaisse qu'au sein d'un seul et même langage. On peut ainsi avoir des règles de correspondance plus simples entre les composants de différents langages.*

4.3.3.2 Non existence d'un composant dans le glossaire d'un langage

Une fois que les méta-modèles sont réalisés de manière homogène ou que les langages sont représentés avec un niveau de détail maximum[4], mais que, malgré cela, il existe toujours des relations de types « A est entièrement une partie de B » et « A et B possèdent une partie commune » entre deux langages, alors il est nécessaire de «décomposer les composants» en composants « plus petits ». Nous devons, en effet, définir le composant « B – A » pour le cas 3 et les composants « X, Y et Z » pour le cas 4. Cependant, afin de répondre à cette question, il nous faut tout d'abord répondre à l'interrogation suivante :

Que signifie réellement la relation « est une partie de » ?

En effet, la notion d'« ensemble » en mathématiques est très générale. Un « ensemble » peut désigner n'importe quelle collection. Les objets qui se trouvent dans une chambre forment un ensemble, fût-il hétéroclite, défini par la propriété « être un objet situé dans la chambre d'un individu à tel instant ». Cependant, peut-on dire qu'un individu est un ensemble, et que son bras lui appartient selon une relation ensembliste ? On peut répondre oui, en raison de la généralité de la notion d'ensemble; mais ce n'est pas nécessairement *pertinent*, car cela ne

[4] Compte tenu des composants définis dans le glossaire des langages.

correspond pas nécessairement à ce qu'on voulait *modéliser*. Lorsque nous pensons à notre bras gauche, nous ne le considérons pas en effet comme un élément d'un ensemble où il serait analogue à d'autres, mais comme un organe ou *composant* qui appartient à un organisme ou composé où il remplit une fonction spécifique, en relation avec d'autres composants. Les directions d'une entreprise composent celle-ci; ce vocabulaire peut servir à qualifier la relation qui existe entre un bras et un individu : c'est une « composition » ou une «agrégation» au sens UML et non une inclusion ensembliste [Volle, 2003].

Ainsi dans la suite, nous utiliserons le terme d' « ***ensemble*** » pour désigner des éléments qui ont en commun certaines propriétés et le terme de « ***composé*** » pour désigner un élément composé d'autres composants (ou organes).
De ce fait, nous pouvons dire que la relation « est une partie de » correspond, quand elle représente une inclusion ou un chevauchement au sens ensembliste du terme, à une relation d'abstraction ou de généralisation (Figure 4.3-7a). S'il s'agit d'une relation de type composant-composé, comme nous venons de le voir précédemment, alors nous sommes donc dans le cas d'une relation de type composition (ou d'agrégation suivant s'il y a ou non correspondance des cycles de vie) (Figure 4.3-7b).

Figure 4.3-7 : Généralisation (a) et composition (b)

Les deux figures précédentes, qui font référence à l'exemple de la traduction entre une activité GRAI et une activité SADT, illustre bien la différence entre ces deux types de relation. En effet, un contrôle non déclenchant et un déclencheur sont bel et bien des sous-ensembles du composant contrôle. Par contre, le composant support n'est pas un sous-ensemble du composant d'activité mais « fait partie du » composant d'activité au sens composant-composé.

4.3.4 Besoin d'une représentation mathématique - Représentation ensembliste d'une classe

L'objectif recherché par la réalisation des méta-modèles est de représenter l'ensemble des composants appartenant à un langage. Cependant, notre but final est de comparer les composants entre eux. Nous avons vu au paragraphe 4.3.2 quelques situations que l'on peut identifier lorsque l'on veut comparer deux composants. Au paragraphe 4.3.3.2 nous avons montré qu'il fallait décliner ces situations suivant deux points de vue distincts : le point de vue de la généralisation et celui de la composition. Par ailleurs, pour nous le langage UEML est une union des composants des langages, mais comme nous ne voulons pas qu'il y ait de redondance d'information dans le méta-modèle UEML, il est nécessaire de traiter les composants qui ont des points communs de manière globale.

Ainsi, afin d'être complet, nous ne devons pas nous limiter à la comparaison de seulement deux composants, mais étendre les résultats à un nombre « N_c » de composants. De ce fait, afin de pouvoir écrire une série d'équations concernant la comparaison de ces « Nc » composants, nous devons représenter de manière plus mathématique les éléments de modélisation contenus dans une classe d'un diagramme de classe UML.

Il est évident que les notions de classe et d'ensemble sont très proches. Les objets instances d'une classe partagent des caractéristiques générales, exprimées dans la classe sous forme d'attributs, d'opérations et de contraintes. Ainsi, il est facile de pouvoir représenter une classe de manière ensembliste et donc de pouvoir récrire un certains nombre d'équations. Cette mise en forme ensembliste ne se veut pas être originale, mais se veut être claire et simple. Notre but n'est ici que de clarifier les mécanismes d'échange de propriétés entre les classes d'un point de vue de la généralisation et de la composition. Nous utiliserons ensuite ces résultats afin de traiter la comparaison des composants d'une manière plus systématique et pour un nombre quelconque de composants.

Ainsi, la Figure 4.3-8 illustre un exemple concernant la relation qui existe entre la notion de classes et la notion d'ensembles. La classe A possède une seule propriété $P_1(A)$. Chaque objet instance de la classe A peuvent être vu comme les éléments d'un ensemble A.

Les éléments de l'ensemble A seront notés **: a** *comme les objets instances de la classe A. Les différents éléments de l'ensemble $P_1(A)$ correspondant à la propriété de la classe A seront notés $p_1(A)$.*

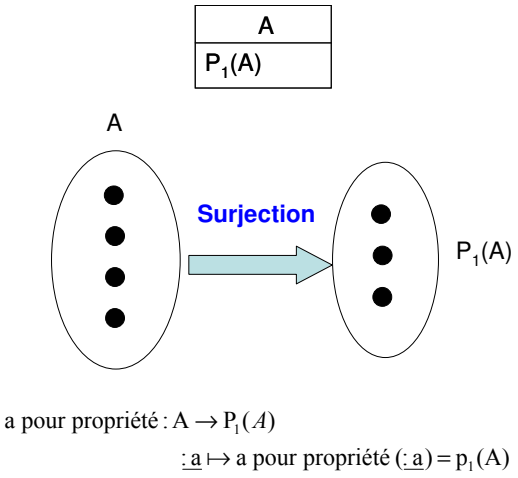


Figure 4.3-8 : Des ensembles aux classes (1)

Cet ensemble possède ainsi une propriété caractéristique $P_1(A)$ correspondant à la propriété de la classe A et valable pour tous les éléments de l'ensemble A. Cette propriété caractéristique peut être également vue comme un ensemble dont les éléments correspondent aux instances de cette propriété. Nous avons donc à relier deux ensembles : l'ensemble A dont les éléments « : *a* » correspondent aux objets instances de la classe A et l'ensemble $P_1(A)$ dont les éléments « $p_1(A)$ » correspondent aux instances de la propriété $P_1(A)$ de la classe A. Nous allons utiliser une fonction (qui est en fait une application) afin de relier ces deux ensembles. En effet, nous pouvons définir l'application **« a pour propriété »** de l'ensemble A vers l'ensemble $P_1(A)$ qui à tout élément de l'ensemble A fait correspondre un élément et un seul de l'ensemble $P_1(A)$. Cette application est une **surjection** car tout élément de $P_1(A)$ est l'image d'au moins un élément de l'ensemble A. Par contre, elle n'est pas une **injection** car certains éléments de P_1 (A) peuvent être l'image de plusieurs éléments de l'ensemble A.

Nous pouvons donc écrire l'application suivante :

$$\textbf{a pour propriété} : A \rightarrow P_1(A)$$
$$\textbf{: a} \mapsto \textbf{a pour propriété (: a)} = p_1(A)$$

Si on généralise ce qui vient d'être dit au cas où une classe A possède plusieurs propriétés, alors la propriété caractéristique d'un ensemble A, notée P (A), est le produit cartésien de « n » ensembles, chacun de ces ensembles correspondant à une propriété de l'ensemble A (et donc de la classe A). Ainsi, $P(A) = P_1(A) \times P_2(A) \times \ldots \times P_n(A)$. Dans le cas général, l'application définie précédemment est une application de l'ensemble A vers l'ensemble $P(A) = P_1(A) \times P_2(A) \times \ldots \times P_n(A)$.

$$\begin{aligned} &a\ pour\ propriété\ A \to P(A) = P_1(A) \times P_2(A) \times \ldots \times P_n(A) \\ &\underline{:a} \mapsto (p_1(A), p_2(A), \ldots, p_n(A)) = a\ pour\ propriété(\underline{:a}) \end{aligned}$$

Sur la Figure 4.3-9, cette application est illustrée dans le cas de trois propriétés.

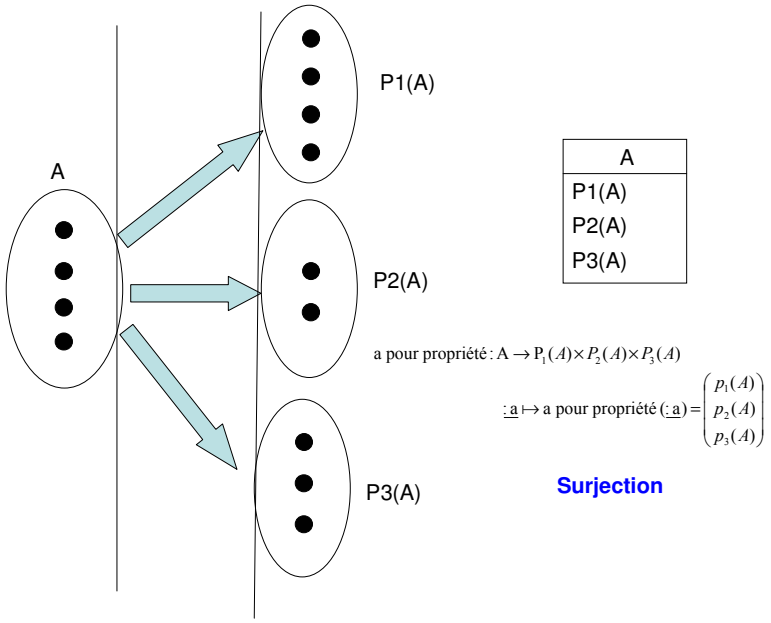


Figure 4.3-9 : Des ensembles aux classes (2)

4.3.4.1 Vue ensembliste de la généralisation et de la composition

4.3.4.1.1 Généralisation

Dans le cas de Figure 4.3-10, le composant B est une généralisation du composant A. Un composant A est un composant B plus spécialisé. D'un point de vue de la théorie des ensembles, tous les éléments de l'ensemble A sont des éléments de l'ensemble B. La propriété caractéristique de l'ensemble B est égale à $P(B) = P_1(B) \times P_2(B) \times \ldots \times P_n(B)$ dans le cas où la

classe B possède « n » propriétés. Pour l'ensemble A, il faut distinguer deux types propriétés caractéristiques : une spécifique à A, noté $P_{sp,}$ et qui provient des « m » propriétés de la classe A et qui vaut $P_{sp}(A) = P_1(A)$ x $P_2(A)$ x … x $P_m(A)$ et une autre qui est en en fait P(B) car les éléments de A sont aussi des éléments de B.

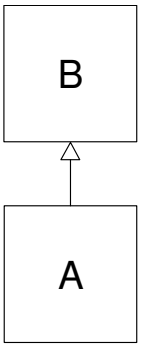


Figure 4.3-10 : Généralisation et vision ensembliste

On a donc pour l'ensemble B l'expression suivante :

$$\begin{aligned} \text{a pour propriété}_1 : \ & B \to P(B) = P_1(B) \times P_2(B) \times \ldots \times P_n(B) \\ & \underline{: b} \mapsto (p_1(B), p_2(B), \cdots, p_n(B)) = \text{a pour propriété}(\underline{: b}) \end{aligned}$$

Pour l'ensemble A, nous avons donc deux expressions pour représenter les propriétés

- Une liée au fait que $A \subset B$,

$$\begin{aligned} \text{a pour propriété}_3 : A \to & P(B) = P_1(B) \times P_2(B) \times \ldots \times P_n(B) \\ & \underline{: a} \mapsto (p_1(B), p_2(B), \cdots, p_n(B)) = \text{a pour propriété}(\underline{: a}) \end{aligned}$$

- une uniquement liée aux propriétés de A.

$$\begin{aligned} \text{a pour propriété}_3 : A \to & P_{sp}(A) = P_1(A) \times P_2(A) \times \ldots \times P_m(A) \\ & \underline{: a} \mapsto (p_1(A), p_2(A), \cdots, p_m(A)) = \text{a pour propriété}(\underline{: a}) \end{aligned}$$

Ainsi, on obtient finalement en regroupant les deux expressions :

$$\text{a pour propriété}_4 : A \rightarrow P(B)\times P_{sp}(A) = P_1(B)\times P_2(B)\times\ldots\times P_n(B)\times P_1(A)\times P_2(A)\times\ldots\times P_m(A)$$
$$a \mapsto (p_1(B), p_2(B), \cdots, p_n(B), p_1(A), p_2(A), \cdots, p_m(A)) = \text{a pour propriété}(:\underline{a})$$

La Figure 4.3-11 illustre un exemple.

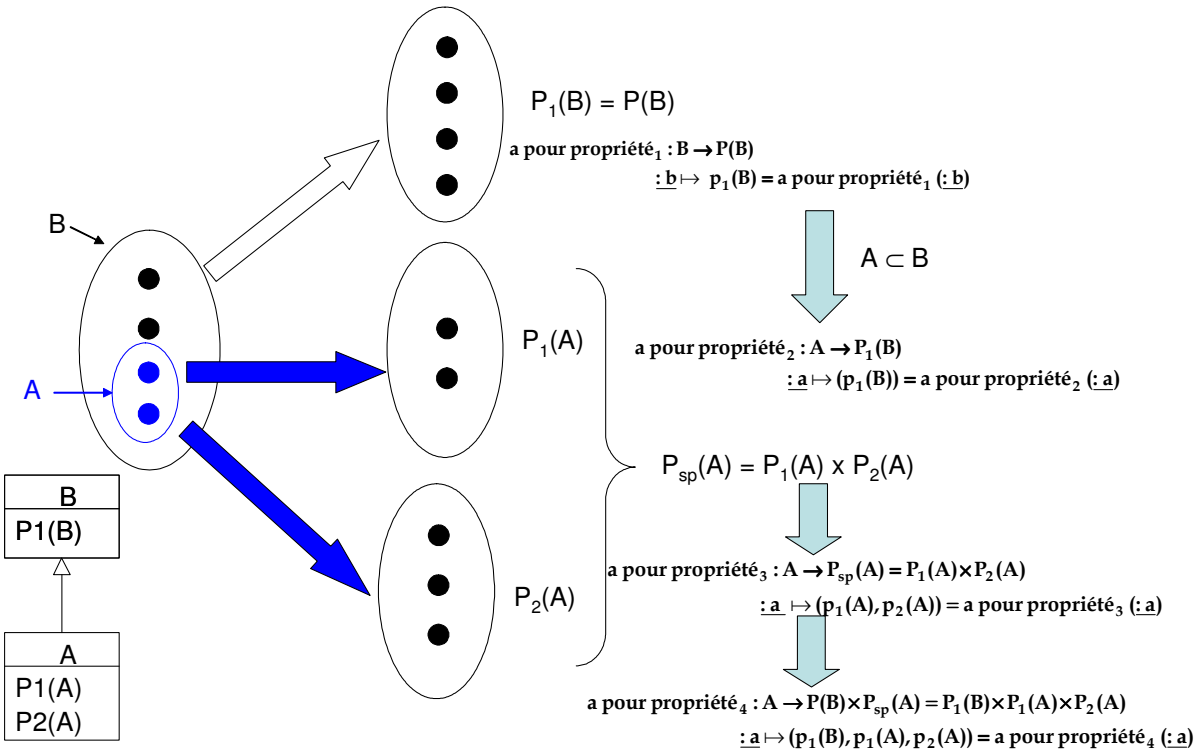


Figure 4.3-11 : Exemple de généralisation en ensembliste

4.3.4.1.2Composition

Nous allons ici présenter une vision ensembliste de la composition. Pour cela, nous sommes amenés à définir quelques applications, d'une manière tout à fait similaire, à ce que l'on a fait avec les propriétés, dans le cas de la généralisation.

Considérons une classe A et les classes X_n, qui représentent la totalité des « n » composants de classe A. Ainsi, nous allons définir une « application » surjective « a pour composant » qui va associer a un élément de l'ensemble A (correspondant à un objet instance de la classe A), un élément et un seul de chaque ensemble X_n (correspondant à un objets instance de la classe X_n) (Figure 4.3-12).

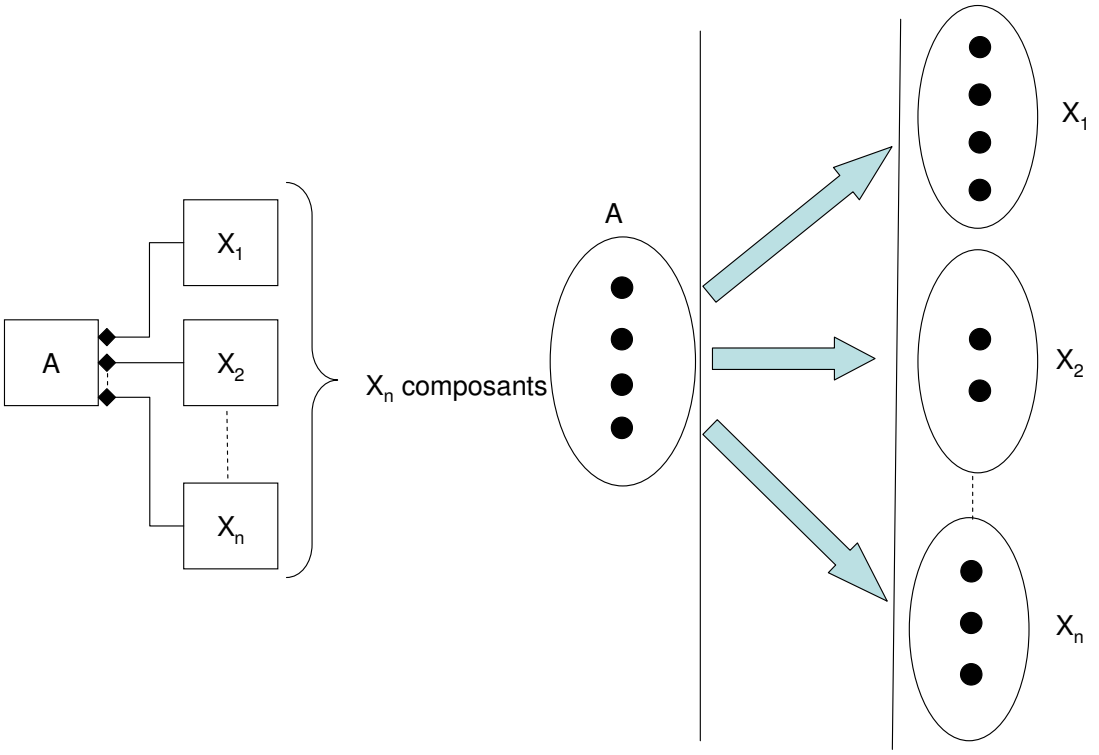
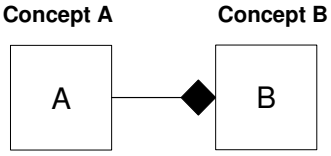


Figure 4.3-12 : Représentation ensembliste de la composition

Ainsi, on peut écrire l'expression qui donne les composants qui appartiennent à un composant donné.

$$\begin{aligned} &a\ pour\ composant\ :\ A \to X_1 \times X_2 \times \ldots \times X_n \\ &\qquad\qquad :\underline{a} \mapsto \left(x_1, x_2, \ldots, x_n\right) = \text{a pour composant } (:\underline{a}) \end{aligned}$$

Appliquons l'expression précédente, à l'exemple suivant, où le composant A est un composant du composant B.



$$\begin{aligned} &a\ pour\ composant\ :\ B \to A \\ &\qquad\qquad :\underline{b} \mapsto \text{ a pour composant } (:\underline{b}) = (:\underline{a}) \end{aligned}$$

Nous ne parlerons pas ici des relations que l'on peut définir entre les propriétés caractéristiques du composant et de ses composés. Nous verrons cela dans le paragraphe 4.3.5.2

4.3.5 Retour à la comparaison des composants

4.3.5.1 Comparaison d'un point de vue de la généralisation

En reprenant les cas 3 et 4 vus précédemment, et en les étudiant avec un point de vue de la généralisation, nous obtenons les résultats présentés dans les paragraphes suivants :

La notation utilisée est la suivante :

$\subset$: opérateur d'inclusion, $A \subset B$ signifie que l'ensemble A est inclus dans l'ensemble B.

$\cup$: opérateur d'union, $A \cup B$ est l'union des ensembles A et B

/ : opérateur de différence ensembliste (B/A = B –A d'un point de vue ensembliste).

${}^{C}A$: représente le complément de l'ensemble A dans l'ensemble E ($E = A \cup B$).

4.3.5.1.1 Cas 3 : A est entièrement une partie de B

D'un point de vue ensembliste cette situation se représente comme indiqué par la Figure 4.3-13.

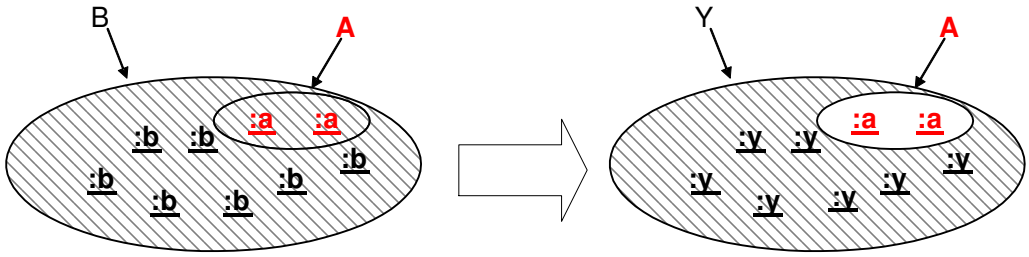


Figure 4.3-13 : A est entièrement une partie de B d'un point de vue de la généralisation

On a donc : $\mathbf{A \subset B, A \neq B}$ et $\mathbf{Y = B/A = {}^{c}A \cap B}$

On peut ainsi écrire que :

$$\mathbf{P(A) = P(B) \times P_{sp}(A)}$$

$$\mathbf{P(Y) = P(B) \times P_{sp}(Y)}$$

car les classes A et Y sont des sous-classes de la classe B. A ce niveau là, il n'est pas possible de déterminer la propriété caractéristique spécifique de Y, ainsi on en fait P(Y) = P(B).

Pour retrouver les composants originaux, il faut appliquer les règles de correspondances suivantes :

$$\mathbf{B = Y \cup A_{UEML}}$$

$$\mathbf{P(B) = P(A_{UEML}) \cap P(Y)}$$

Ce qui donne d'un point de vue d'un diagramme de classe, la Figure 4.3-14.

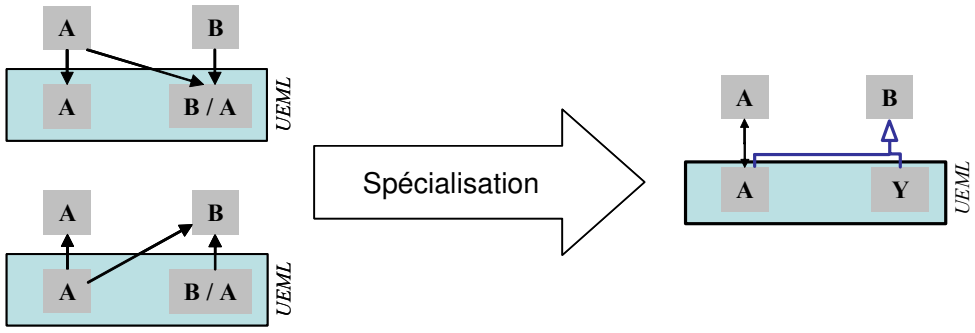


Figure 4.3-14 : Cas 3 d'un point de vue de la généralisation

Dan ce cas, les règles de correspondances sont :	**A_{UEML} est une sous-classe de B**
	Y est une sous classe de B

La généralisation multiple entre B, Y et A est complète et disjointe.

4.3.5.1.2Cas 4 : A et B ont une partie commune

Chevauchement partiel ($A \not\subset B$, $B \not\subset A$, $A \cap B \neq \varnothing$)

D'un point de vue ensembliste cette situation se représente comme indiqué par la Figure 4.3-15.

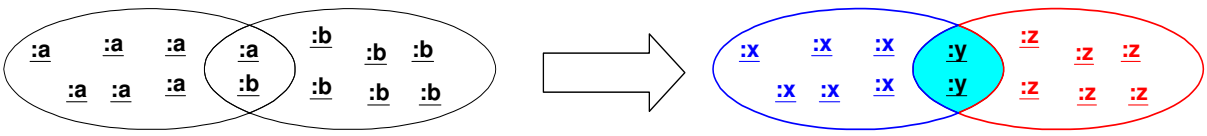


Figure 4.3-15 : A et B ont une partie commune d'un point de vue de la généralisation

On a : $\mathbf{Y \subset A, Y \subset B, Y = A \cap B}$

$\mathbf{Y \subset A, Y \subset B, Y = A \cap B}$ **donc** $\mathbf{P(Y) = P_{sp}(Y) \times P(A) \times P(B)}$

$\mathbf{X \subset A, X = A/Y}$ **donc** $\mathbf{P(X) = P_{sp}(X) \times P(A)}$

$\mathbf{Z \subset B, Z = B/Y}$ **donc** $\mathbf{P(Z) = P_{sp}(Z) \times P(B)}$

A ce niveau là, et comme dans le cas précédent, il n'est pas possible de déterminer les propriétés caractéristiques spécifiques de X, Y et Z. Ainsi, on aura seulement P(X) = P(A), P(Y) = P(A) x P(B), P(Z) = P(B).

Pour retrouver les composants originaux, il faut appliquer les règles suivantes :

$$\mathbf{A = Y \cup X}$$

$$\mathbf{B = Y \cup Z}$$

$$\mathbf{P(A) = P(X) \cap P(Y)}$$

$$\mathbf{P(Z) = P(Y) \cap P(Z)}$$

Ce qui donne d'un point de vue d'un diagramme de classe, la Figure 4.3-16.

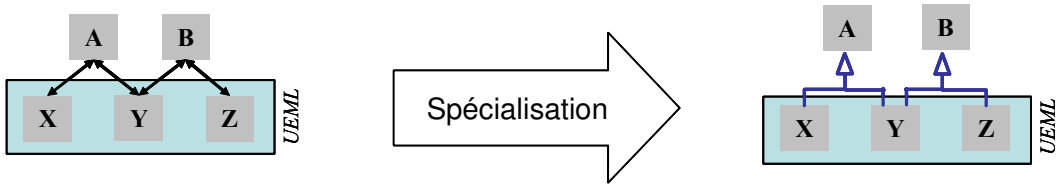


Figure 4.3-16 : Cas 4 d'un point de vue de la généralisation

Dan ce cas, les règles de correspondances sont :	**X est une sous-classe de A**
	Z est une sous classe de B
	Y est une sous-classe de A et B simultanément.

4.3.5.2 Comparaison d'un point de vue de l'agrégation - composition

Notations utilisées :

◇∩ = « intersection » composite

◇∪ = « union » composite

◇/ = « différence » composite

4.3.5.2.1 Cas 3 : A est entièrement une partie de B

Ce cas ci est illustré sur la Figure 4.3-17. Ainsi, nous pouvons écrire :

$$\text{a pour composant} : B \to A$$
$$:\underline{b} \mapsto \left(:\underline{a}, :\underline{y}\right) = \text{a pour composant} \left(:\underline{b}\right)$$

On a : **Y = (B ◇/ A)**

Ce n'est donc pas dans ce cas là une relation de type soustraction ensembliste mais bien une différence au sens structurel.

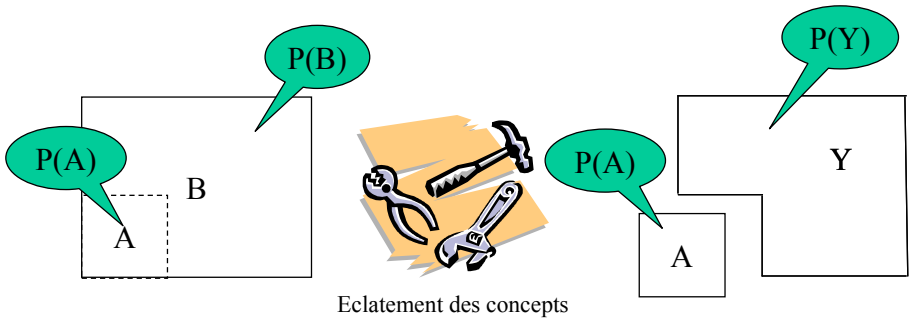


Figure 4.3-17 : A est entièrement une partie de B d'un point de vue de la composition

Afin d'exprimer les règles de correspondances qui existent entre les composants, il est nécessaire de définir quelques fonctions entre le composé et l'ensemble de ces composants. En effet, il est notable que la propriété caractéristique d'un composant est liée à celle de son composé et inversement. Ainsi, la propriété caractéristique P(A) du composant A est liée à la propriété caractéristique P(B) du composant B. Nous définissons donc la fonction « $\mathbf{f_a}$ » et

P(A) est égale à f_a [P(B)]. De la même manière nous définissons une fonction « $\mathbf{f_y}$ » qui donne la propriété caractéristique du composant Y en fonction de celle de A et de B.

$$\mathbf{Y = (B \diamondsuit A)}$$
$$\mathbf{P(A) = f_a\ [\ P(B)]}$$
$$\mathbf{P\ (Y) = f_y\ [\ P(A)\ ,\ P(B)\ ]}$$

Afin d'être capable de retrouver la propriété caractéristique du composant B en fonction de celles de ses composants A et Y, nous devons définir la nouvelle fonction « $\mathbf{f_b}$ ».

Ainsi, pour retrouver les composants originaux, il faut appliquer les règles suivantes :

$$\mathbf{B = Y \diamondsuit A^{UEML}}$$
$$\mathbf{P\ (B) = f_b\ [\ P(A_{UEML})\ ,\ P(Y)\ ]}$$

4.3.5.2.2Cas 4 : A et B ont une partie commune

Ce cas-ci est illustré sur la Figure 4.3-18. Ainsi, nous pouvons écrire :

$$\text{a pour composant} : A \to Y \times X$$
$$:\underline{a} \mapsto \left(:\underline{y}\ ,:\underline{x}\right) = \text{a pour composant}\ (:\underline{\text{a}})$$

$$\text{a pour composant} : B \to Y \times X$$
$$:\underline{b} \mapsto \left(:\underline{y}\ ,:\underline{z}\right) = \text{a pour composant}\ (:\underline{\text{b}})$$

Soit :

$$\mathbf{Y=A \diamondsuit B}$$
$$\mathbf{Z = (B \diamondsuit Y)}$$
$$\mathbf{X = (A \diamondsuit Y)}$$

Pour les mêmes raisons que dans le cas précédent, nous devons définir trois fonctions, afin d'écrire les règles de correspondances entre les différents composants. Nous définissons les fonctions « $\mathbf{f_x}$ », « $\mathbf{f_y}$ » et « $\mathbf{f_z}$ ».

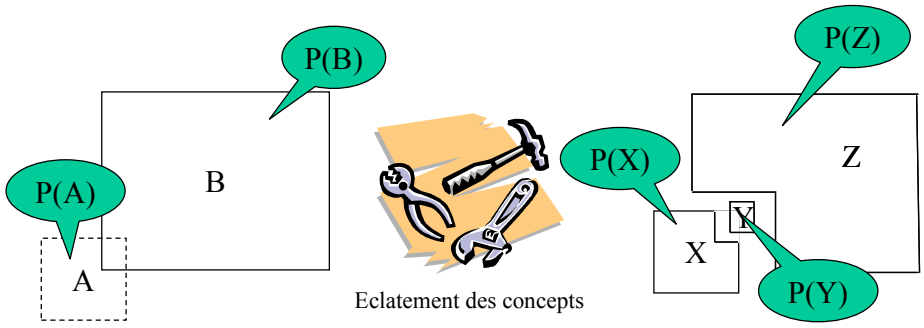


Figure 4.3-18 : A et B ont une partie commune B d'un point de vue de la composition

$$P(X) = f_x\ [\ P(A), P(B)]$$
$$P(Y) = f_y\ [P(A), P(B)]$$
$$P(Z) = f_z\ [P(A), P(B)]$$

Nous définissons également les fonctions « f_A » et « f_B » afin d'être capable de retrouver la propriété caractéristique des composant A et B en fonction de celles de leurs composants. Ainsi, pour retrouver les composants originaux il faut appliquer les règles suivantes :

$$A = Y \diamondsuit X$$
$$B = Y \diamondsuit Z$$
$$P(A) = f_A\ [P(Y), P(X)]$$
$$P(B) = f_B\ [\ P(Y), P(Z)]$$

4.3.5.2.3Définition formelle des fonctions

A ce stade, nous avons défini des fonctions afin de pouvoir écrire des règles de correspondances entre les composants. Il est, cependant, important de noter que la définition formelle de ces fonctions, n'est pas une chose facile. Considérons, en effet, l'exemple simple illustré sur la Figure 4.3-19.

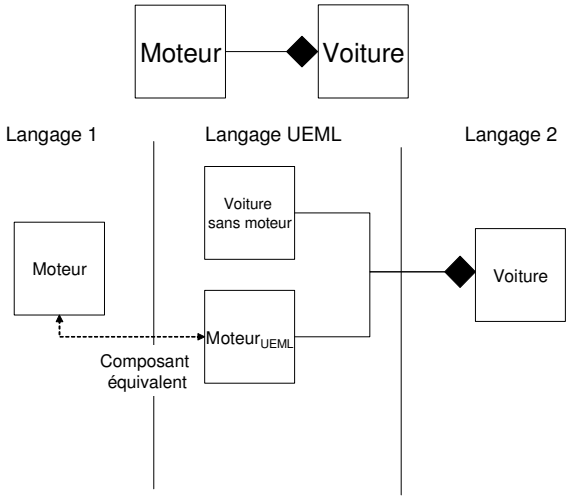


Figure 4.3-19 : Comparaison de deux composants d'un point de vue de la composition UML

Nous devons comparer deux composants qui appartiennent chacun à deux langages différents. Le composant de « Moteur » et celui de « Voiture ». Nous sommes dans le cas 3, d'un point de vue composition. Nous décomposons donc le composant de « voiture » en deux sous-composants, qui sont le « moteur » (qui appartient également au langage 1) et le composant « voiture sans moteur ». Ces deux sous-composants appartiennent au langage UEML.

On a donc :

$$\text{Voiture sans moteur} = (\text{Voiture} \ \diamondsuit \ \text{Moteur})$$

a pour composant: $Voiture \rightarrow Moteur \times Voiture\ sans\ moteur$

$$:\underline{Voiture} \mapsto \left(:\underline{Moteur}, :\underline{Voiture\ sans\ moteur}\right) = \text{a pour composant}(\underline{:\text{Voiture}})$$

Comme **$\text{Moteur}_{\text{UEML}} \equiv \text{Moteur}$** alors nous avons également **:**

a pour composant: $Voiture \rightarrow Moteur_{UEML} \times Voiture\ sans\ moteur$

$$:\underline{Voiture} \mapsto \left(:\underline{Moteur}_{UEML}, :\underline{Voiture\ sans\ moteur}\right) = \text{a pour composant}(\underline{:\text{Voiture}})$$

Les relations de correspondance entre les propriétés caractéristiques des composants sont représentées ci-dessous :

$$P(Moteur_{UEML}) = f_{Moteur}\ [\ P(Voiture)] \text{ et/ou } P(Moteur_{UEML}) = P(Moteur)$$

$$P(Moteur_{UEML}) = f_{Moteur}\ [\ P(Voiture)]$$

$$P\ (Voiture\ sans\ moteur) = f_{\ Voiture\ sans\ moteur}\ [\ P(Voiture)\ ,\ P(Moteur)\ ]$$

La fonction $f_{Voiture}$ ci-après permet de retrouver la propriété caractéristique du composant « Voiture » en fonction de celles des composants « Voiture sans moteur » et « Moteur ».

$$P\ (Voiture) = f_{\ Voiture}\ [\ P(Voiture\ sans\ moteur)\ ,\ P(Moteur_{UEML})\ ]$$

On a également :

$$Voiture = Voiture\ sans\ moteur \diamondsuit Moteur_{UEML}$$

4.3.5.2.4Conclusion

Cet exemple permet de mettre en évidence la difficulté de définir de manière formelle les différentes fonctions précédentes. Il est, en effet, très difficile de pouvoir définir, pour le composant « voiture sans moteur », tout un ensemble de propriétés qui seraient extraites du composant « voiture » de manière cohérente.

Notre but ici, n'est pas de proposer une définition formelle de ces fonctions. Il est cependant important de prendre en considération ce qui vient d'être dit. Afin d'utiliser le langage UEML comme langage « pivot », entre plusieurs langages, il est donc nécessaire de pouvoir écrire toutes les règles de correspondance entre ces langages et le langage UEML. Ainsi, nous venons de voir que dans le cas d'une relation de type composant-composé, cette tâche n'est pas simple. Nous avons simplement donné, ici, quelques éléments de réflexion.

4.3.6 Synthèse – Généralisation à N_c composants

Nous allons dans ce paragraphe généraliser l'ensemble des éléments que nous avons présentés concernant la comparaison de deux composants, à un nombre quelconque de composants.

Si on considère que l'on a un nombre N_c de composants (**C_i**) à comparer, on peut alors définir $N_{CE} = 2^{N_C} - 1$ « composants élémentaires (sous-composants) » (**CE_i**).

Les équations présentées sont des équations utilisables dans le cas général. Ainsi, elles ne présument pas que les intersections entre les composants d'un même langage sont nulles (ensemble vide).

4.3.6.1 Point de vue de la généralisation

Soit l'ensemble **E** représentant l'union ensembliste des N_c composants. On a alors :

$$\mathbf{E} = \bigcup_{\mathbf{i=1}}^{\mathbf{N_c}} (\mathbf{C_i})$$

Détermination des composants élémentaires.

On peut ainsi définir les $N_{CE} = 2^{N_C} - 1$ composants élémentaires par les équations suivantes :

$$\forall i \in [1, N_{CE}],\ CE_i = \bigcap_{j=1}^{N_C} (C_j)'$$

$$\forall i \in [1, N_{CE}],\ P(CE_i) = P_{sp}(CE_i) \underset{j=1}{\overset{N_c}{X}} P(C_j)'$$

où **(C_j)'** est soit égal à **(C_j)** soit égal à son complémentaire ensembliste, c'est à dire à **$[E - (C_j)]$** noté **$^c(C_j)$**. Toutes les combinaisons différentes donnant (CE_i) doivent être effectuées.

Remarques importantes :

SI	(C_j)' est égale à ^cC_j
ALORS	il est nécessaire de supprimer le terme P(^cC_j) de l'équation de P(CE_i) car il n'a pas de sens car l'information est déjà donnée par les autres termes.
SI	$\forall i \in [1, N_{CE}],\ CE_i = \bigcap_{j=1}^{N_C} (C_j)' = \varnothing$
ALORS	l'expression donnant P(CE_i) n'a pas de sens et ne doit pas être considérée.

On peut également noter que l'intersection, entre tous les complémentaires des composants est nulle.

$$\forall i \in [1, N_{CE}], \bigcap_{j=1}^{N_C} {}^{C}(C_j) = \varnothing$$

Règles de correspondance - Recomposition des composants originaux.

Afin de retrouver les composants originaux, on utilise les équations suivantes :

$$\forall i \in [1, N_C], C_i = \bigcup_{j=1}^{N_{CE}} \left[(CE_{j/i}) \right]$$

$$\forall i \in [1, N_C], P(C_i) = \bigcup_{j=1}^{N_{CE}} \left[P(CE_{j/i}) \right]$$

où $CE_{j/i}$ est égale à CE_j si C_i apparaît dans l'équation de CE_j et à l'ensemble vide sinon.

Remarque importante :

SI	$P(CE_j)$ n'a pas de sens
ALORS	il ne doit pas apparaître dans l'équation.

Exemple :

Nous allons ici traiter un exemple de recherche des composants élémentaires, dans le cas de la comparaison de trois composants. Pour $N_c = 3$ composants on a $N_{CE} = 2^3-1 = 7$ composants élémentaires.

La Figure 4.3-20 illustre le découpage des composants originaux en composants élémentaires. On a donc :

$$\mathbf{CE_1} = C_1 \cap C_2 \cap C_3$$

$$\mathbf{P(CE_1)} = P_{sp}(CE1) \times P(C_1) \times P(C_2) \times P(C_3)$$

$$\mathbf{CE_2} = C_1 \cap C_2 \cap {}^{c}C_3$$

$$\mathbf{P(CE_2)} = P_{sp}(CE_2) \times P(C_1) \times P(C_2) \times \cancel{P({}^{c}C_3)} = P_{sp}(CE_2) \times P(C_1) \times P(C_2)$$

$$\mathbf{CE_3} = C_1 \cap {}^{c}C_2 \cap C_3$$

$$\mathbf{P(CE_3)} = P_{sp}(CE_3) \times P(C_1) \times \cancel{P(^{c}C_2)} \times P(C_3) = P_{sp}(CE_3) \times P(C_1) \times P(C_3)$$

$$\mathbf{CE_4} = C_1 \cap {}^{c}C_2 \cap {}^{c}C_3$$

$$\mathbf{P(CE_4)} = P_{sp}(CE_4) \times P(C_1) \times \cancel{P(^{C}C_2)} \times \cancel{P(^{C}C_3)} = P_{sp}(CE_4) \times P(C_1)$$

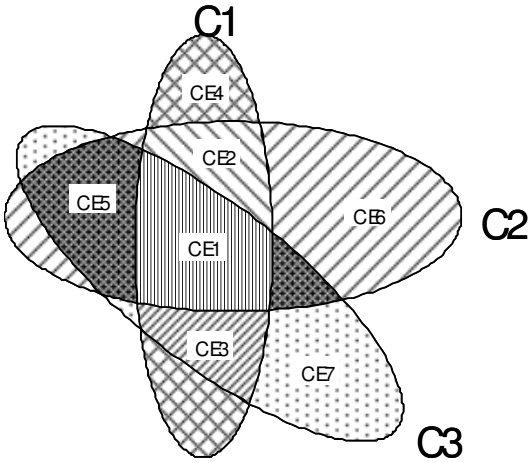


Figure 4.3-20 : Composants et composants élémentaires

$$\mathbf{CE_5} = {}^{c}C_1 \cap C_2 \cap C_3$$

$$\mathbf{P(CE_5)} = P_{sp}(CE_5) \times \cancel{P(^{C}C_1)} \times P(C_2) \times P(C_3) = P_{sp}(CE_5) \times P(C_2) \times P(C_3)$$

$$\mathbf{CE_6} = {}^{c}C_1 \cap C_2 \cap {}^{c}C_3$$

$$\mathbf{P(CE_6)} = P_{sp}(CE_6) \times \cancel{P(^{C}C_1)} \times P(C_2) \times \cancel{P(^{C}C_3)} = P_{sp}(CE_6) \times P(C_2)$$

$$\mathbf{CE_7} = {}^{c}C_1 \cap {}^{c}C_2 \cap C_3$$

$$\mathbf{P(CE_7)} = P_{sp}(CE_7) \times \cancel{P(^{C}C_1)} \times \cancel{P(^{C}C_2)} \times P(C_3) = P_{sp}(CE_7) \times P(C_3)$$

$$^{c}C_1 \cap {}^{c}C_2 \cap {}^{c}C_3 = \varnothing$$

Pour retrouver les composants originaux il est nécessaire d'utiliser les expressions suivantes :

$$\mathbf{C_1} = CE_1 \cup CE_2 \cup CE_3 \cup CE_4$$

$$\mathbf{P(C_1)} = P(CE_1) \cap P(CE_2) \cap P(CE_3) \cap P(CE_4)$$

$$\mathbf{C_2} = CE_1 \cup CE_2 \cup CE_5 \cup CE_6$$

$$\mathbf{P(C_2)} = P(CE_1) \cap P(CE_2) \cap P(CE_5) \cap P(CE_6)$$

$$\mathbf{C_3} = CE_1 \cup CE_3 \cup CE_5 \cup CE_7$$

$$\mathbf{P(C_3)} = P(CE_1) \cap P(CE_3) \cap P(CE_5) \cap P(CE_7)$$

4.3.6.2 Point de vue de l'agrégation - composition

Soit l'ensemble **E** représentant l'union composite des N_c composants. On a :

$$\mathbf{E} = \diamondsuit_{\mathbf{i=1}} \mathbf{(C_i)}$$

Détermination des sous-composants.

On peut ainsi définir les $N_{CE} = 2^{N_C} - 1$ composants élémentaires par l'équation :

$$\forall \mathbf{i \in [1, N_{CE}]} \quad \mathbf{CE_i} = \diamondsuit_{\mathbf{j=1}}^{\mathbf{N_c}} \mathbf{(C_j)'}$$

avec

$$P(CE_i) = f_{CE_i} \ [P(C_j)']_{j=1}^{N_C}$$

où $\mathbf{(C_j)'}$ est soit égal à $\mathbf{(C_j)}$ soit égal à son complémentaire composite c'est à dire à $\mathbf{[E \diamondsuit (C_j)]}$ noté ${}^{c\diamondsuit}(C_j)$.

Remarques importantes :

SI	$(C_j)'$ est égale à ${}^{c}C_j$
ALORS	il est nécessaire de supprimer le terme $P({}^{c}C_j)$ de l'équation de $P(CE_i)$ car il n'a pas de sens car l'information est déjà donnée par les autres termes.
SI	$\forall \mathbf{i \in [1, N_{CE}]} \quad \mathbf{CE_i} = \diamondsuit_{\mathbf{j=1}}^{\mathbf{N_c}} \mathbf{(C_j)'}$
ALORS	l'expression donnant $P(EC_i)$ n'as pas de sens et ne doit pas être considérée.

Règles de correspondances - Recomposition des composants originaux

Afin de retrouver les composants originaux on utilise les équations suivantes :

$$\forall \in [1,N_C],\ C_i = \bigcup_{j=1}^{N_{CE}} [(CE_{j/i})]$$

$$\forall i \in [1, N_C],\ P[C_i] = f_{\_C_i}\left[\left[P(CE_{j/i})\right]_{j=1}^{N_{CE}}\right]$$

Remarque importante :

SI	$P(CE_j) = \varnothing$
ALORS	il ne doit pas apparaître dans l'équation.

4.3.7 Retour à l'exemple – Comparaison plus formelle

Les méta-modèles présentés précédemment concernant l'activité SADT et l'activité GRAI permettent d'avoir une bonne vision des composants de chaque langage considéré. Chaque composant de modélisation peut maintenant être comparé aux autres. Nous utilisons les équations présentées dans le paragraphe précédent. Ainsi, une première comparaison permet de mettre en évidence les expressions suivantes :

Nom GRAI ≡ Nom SADT	Numéro GRAI ≡ Numéro SADT
Sortie GRAI ≡ Sortie SADT	Déclencheur ⊂ Contrôle
Support ∩ Contrôle ≠ ∅	Mécanisme ⊂ Support
Entrée ⊂ Support	Déclencheur ∩ Entrée = ∅

A ce niveau, il est possible d'utiliser les équations du paragraphe précédent avec Nc = 5. Ainsi, nous devons considérer les composants support, contrôle, déclencheur, entrée et mécanisme. Dans ce cas, nous devons résoudre $2^5 - 1 = 31$ équations, donnant 31 composants élémentaires. Cependant, beaucoup d'entre eux sont en fait égaux à l'ensemble vide. Lorsque le calcul est fait à l'aide d'une application informatique cela ne pose pas réellement de problème. Toutefois, dans le cadre de cet exemple nous allons séparer la comparaison en deux sous-problèmes :

1. *comparaison des composants support, entrée et mécanisme,*
2. *comparaison des composants déclencheur, contrôle et support.*

4.3.7.1 Comparaison des composants Support, Entrée et Mécanisme

4.3.7.1.1 Détermination des composants élémentaires

Nous considérons que $P_{sp}(CE_j) = \varnothing$ car nous ne prenons pas en compte les attributs supplémentaires que peuvent posséder certaines classes par rapport à la méta-classe dont elles sont une spécialisation. Ainsi, on peut déterminer les composants élémentaires obtenus lors de cette comparaison, ainsi que leur propriété caractéristique respective, comme illustré sur le Tableau 4.3-2 :

Composants élémentaires	Propriétés caractéristiques
$CE_1 = SUPPORT_1 \cap ENTREE_1 \cap MECANISME_1 = \varnothing$	$P(CE_1)$ n'a pas de sens
$CE_2 = SUPPORT_1 \cap ENTREE_1 \cap {}^{c}MECANISME_1 = ENTREE_{UEML} = ENTREE_1$	$P(ENTREE_{UEML}) = P(ENTREE_1)$ et/ou $P(ENTREE_{UEML}) = P(SUPPORT_1)$
$CE_3 = SUPPORT_1 \cap {}^{c}ENTREE_1 \cap MECANISME_1 = MECANISME_{UEML} = MECANISME_1$	$P(MECANISME_{UEML}) = P(MECANISME_1)$ et/ou $P(MECANISME_{UEML}) = P(SUPPORT_1)$
$CE_4 = SUPPORT_1 \cap {}^{c}ENTREE_1 \cap {}^{c}MECANISME_1 = \varnothing$	$P(CE_4)$ n'a pas de sens
$CE_5 = {}^{c}SUPPORT_1 \cap ENTREE_1 \cap MECANISME_1 = \varnothing$	$P(CE_5)$ n'a pas de sens
$CE_6 = {}^{c}SUPPORT_1 \cap ENTREE_1 \cap {}^{c}MECANISME_1 = \varnothing$	$P(CE_6)$ n'a pas de sens
$CE_7 = {}^{c}SUPPORT_1 \cap {}^{c}ENTREE_1 \cap MECANISME_1 = \varnothing$	$P(CE_7)$ n'a pas de sens

Tableau 4.3-2 : Comparaison des composants Support, Entrée et Mécanisme

4.3.7.1.2 Règles de correspondances

Par la suite, on peut écrire les règles de correspondances entre les composants élémentaires identifiés précédemment et les composants des langages originaux (activité SADT et GRAI). On procède de la même manière en ce qui concerne les propriétés caractéristiques. Ces résultats sont illustrés sur le Tableau 4.3-3.

Composants	Propriétés caractéristiques
$SUPPORT_1 = \varnothing \cup ENTREE_{UEML} \cup MECANISME_{UEML} \cup \varnothing$ $= ENTREE_{UEML} \cup MECANISME_{UEML}$	$P(SUPPORT_1)$ $= P(ENTREE_{UEML}) \cap P(MECANISME_{UEML})$
$ENTREE_1 = \varnothing \cup ENTREE_{UEML} \cup \varnothing \cup \varnothing$ $= ENTREE_{UEML}$	$P(ENTREE_1) = P(ENTREE_{UEML})$
$MECANISME_1 = \varnothing \cup MECANISME_{UEML} \cup \varnothing \cup \varnothing$ $= MECANISME_{UEML}$	$P(MECANISME_1) = P(MECANISME_{UEML})$

Tableau 4.3-3 : Comparaison des composants Support, Entrée et Mécanisme – règles de correspondances

Nous pouvons ainsi construire une partie du méta-modèle de notre langage UEML, comme illustré sur la Figure 4.3-21.

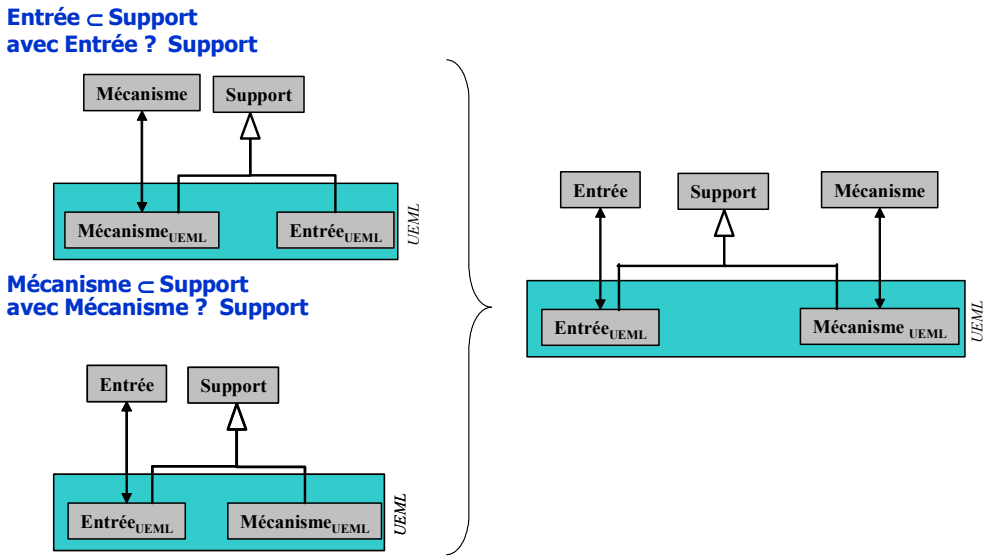


Figure 4.3-21 : Comparaison des composants support, entrée et mécanisme

4.3.7.2 Comparaison des composants Déclencheur, Contrôle et Support

4.3.7.2.1 Détermination des composants élémentaires

On peut également déterminer les composants élémentaires obtenus lors de cette comparaison, ainsi que leur propriété caractéristique respective, comme illustré sur le Tableau 4.3-4.

Composants élémentaires	Propriétés caractéristiques
$CE_1 = SUPPORT_2 \cap CONTROLE_2 \cap DECLENCHEUR_2 = \varnothing$	$P(CE_1)$ n'a pas de sens
$CE_2 = SUPPORT_2 \cap CONTROLE_2 \cap {}^{c}DECLENCHEUR_2$ = CONTROLE NON DECLENCHANT	P(CONTROLE NON DECLENCHANT) = $P(SUPPORT_2)$ x $P(CONTROLE_2)$
$CE_3 = SUPPORT_2 \cap {}^{c}CONTROLE_2 \cap DECLENCHEUR_2 = \varnothing$	$P(CE_3)$ n'a pas de sens
$CE_4 = SUPPORT_2 \cap {}^{c}CONTROLE_2 \cap {}^{c}DECLENCHEUR_2 = \varnothing$	$P(CE_4)$ n'a pas de sens
$CE_5 = {}^{c}SUPPORT_2 \cap CONTROLE_2 \cap DECLENCHEUR_2 = DECLENCHEUR_{UEML} = DECLENCHEUR_2$	$P(DECLENCHEUR_{UEML}) = P(DECLENCHEUR_2)$ et/ou $P(DECLENCHEUR_{UEML}) = P(CONTROLE_2)$
$CE_6 = {}^{c}SUPPORT_2 \cap CONTROLE_2 \cap {}^{c}DECLENCHEUR_2 = \varnothing$	$P(CE_6)$ n'a pas de sens
$CE_7 = {}^{c}SUPPORT_2 \cap {}^{c}CONTROLE_2 \cap DECLENCHEUR_2 = \varnothing$	$P(CE_7)$ n'a pas de sens

Tableau 4.3-4 : Comparaison des composants Déclencheur, Contrôle et Support

4.3.7.2.2Règles de correspondances

Les règles de correspondances sont illustrées sur le Tableau 4.3-5.

Composants	Propriétés caractéristiques
$SUPPORT_2 = \varnothing \cup$ CONTROLE NON DECLENCHANT $\cup \varnothing \cup \varnothing$ = CONTROLE NON DECLENCHANT	$P(SUPPORT_2)$ = P(CONTROLE NON DECLENCHANT)
$CONTROLE_2 = \varnothing \cup$ CONTROLE NON DECLENCHANT $\cup DECLENCHEUR_{UEML} \cup \varnothing$ = CONTROLE NON DECLENCHANT $\cup DECLENCHEUR_{UEML}$	$P(CONTROLE_2)$ = P(CONTROLE NON DECLENCHANT) $\cap$ $P(DECLENCHEUR_{UEML})$
$DECLENCHEUR_2 = \varnothing \cup \varnothing \cup DECLENCHEUR_{UEML} \cup \varnothing$ = $DECLENCHEUR_{UEML}$	$P(DECLENCHEUR_2)$ = $P(DECLENCHEUR_{UEML})$

Tableau 4.3-5 : Comparaison des composants Déclencheur, Contrôle et Support – règles de correspondances

Nous pouvons ainsi construire une autre partie du méta-modèle de notre langage UEML, comme illustré sur la Figure 4.3-22.

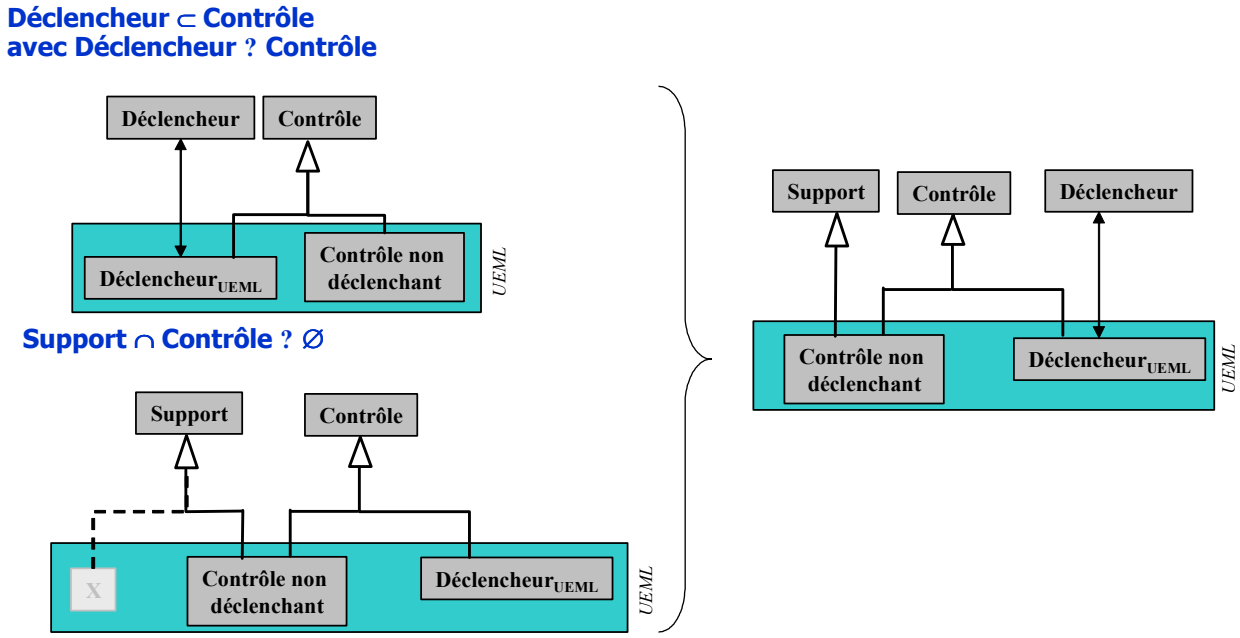


Figure 4.3-22 : Comparaison des composants Déclencheur, Contrôle et Support

4.3.7.3 Comparaison des composants Déclencheur, Contrôle, Support, entrée et mécanisme

4.3.7.3.1 Détermination des composants élémentaires

Afin de déterminer le résultat concernant notre problème originel nous devons combiner les deux résultats précédents. On a donc ainsi quatre composants élémentaires (Tableau 4.3-6) :

Composants élémentaires	Propriétés caractéristiques
CONTROLE NON DECLENCHANT	P(SUPPORT) x P(CONTROLE)
$DECLENCHEUR_{UEML}$	P(CONTROLE) et/ou P(DECLENCHEUR)
$ENTREE_{UEML}$	P(ENTREE) et/ou P(SUPPORT)
$MECANISME_{UEML}$	P(MECANISME) et/ou P(SUPPORT)

Tableau 4.3-6 : Comparaison des composants Déclencheur, Contrôle, Support, entrée et mécanisme

4.3.7.3.2Règles de correspondances

On obtient les règles de correspondances contenues dans le Tableau 4.3-7.

Composants	Propriétés caractéristiques
SUPPORT = $SUPPORT_1 \cup SUPPORT_2$ = $ENTREE_{UEML} \cup MECANISME_{UEML} \cup$ CONTROLE NON DECLENCHANT	P(SUPPORT) = P($ENTREE_{UEML}$) ∩ P($MECANISME_{UEML}$) ∩ P(CONTROLE NON DECLENCHANT)
ENTREE = $ENTREE_1$ = $ENTREE_{UEML}$	P(ENTREE) = P($ENTREE_1$) = P($ENTREE_{UEML}$)
MECANISME = $MECANISME_1$ = $MECANISME_{UEML}$	P(MECANISME) = P($MECANISME_1$) = P($MECANISME_{UEML}$)
CONTROLE = $CONTROLE_2$ = CONTROLE NON DECLENCHANT ∪ $DECLENCHEUR_{UEML}$	P(CONTROLE) = P($CONTROLE_2$) = P(CONTROLE NON DECLENCHANT) ∩ P($DECLENCHEUR_{UEML}$)
DECLENCHEUR = $DECLENCHEUR_2$ = $DECLENCHEUR_{UEML}$	P(DECLENCHEUR) = P($DECLENCHEUR_2$) = P($DECLENCHEUR_{UEML}$)

Tableau 4.3-7 : Comparaison des composants Déclencheur, Contrôle, Support, entrée et mécanisme – règles de correspondances

4.3.7.4 Comparaison des composants Nom, Numéro et Sortie

Nom GRAI = Nom SADT ≡ $\mathbf{Nom_{UEML}}$

Numéro GRAI = Numéro SADT ≡ $\mathbf{Numéro_{UEML}}$

Sortie GRAI = Sortie SADT ≡ $\mathbf{Sortie_{UEML}}$

Nous pouvons ainsi construire la dernière partie du méta-modèle de notre langage UEML, comme illustré sur la Figure 4.3-23.

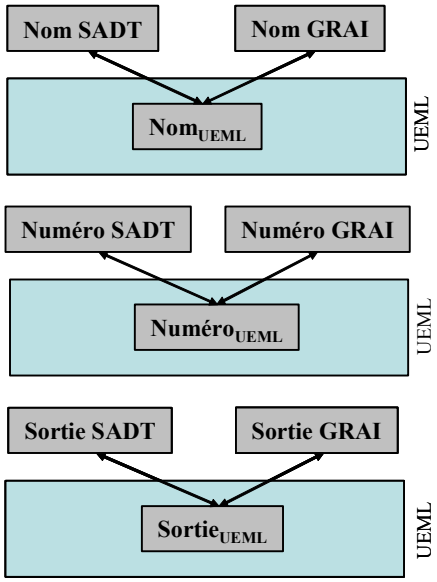


Figure 4.3-23 : Comparaison des composants Nom, Numéro et Sortie

4.3.7.5 Méta-modèle du langage UEML dans le cadre de notre exemple

Nous pouvons finalement construire le méta-modèle simplifié du langage UEML concernant notre exemple (Figure 4.3-24).

Comme UEML est une réunion, alors il contiendra tous les composants élémentaires identifiés précédemment.

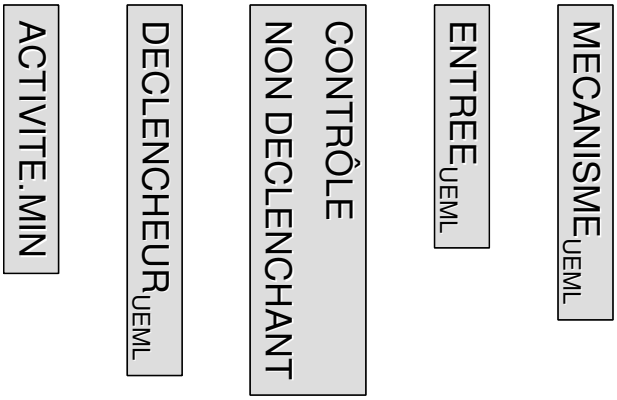


Figure 4.3-24 : Version simplifiée du méta-modèle UEML

Nous n'avons pas considéré dans cet exemple les relations qui peuvent exister entre les composants.

4.3.7.6 Règles de correspondances

Le diagramme de classe de la Figure 4.3-25, montre les règles de correspondances qui existent entre les composants du méta-modèle du langage UEML avec ceux des activités GRAI et SADT. Ces règles de correspondances sont de type généralisation.

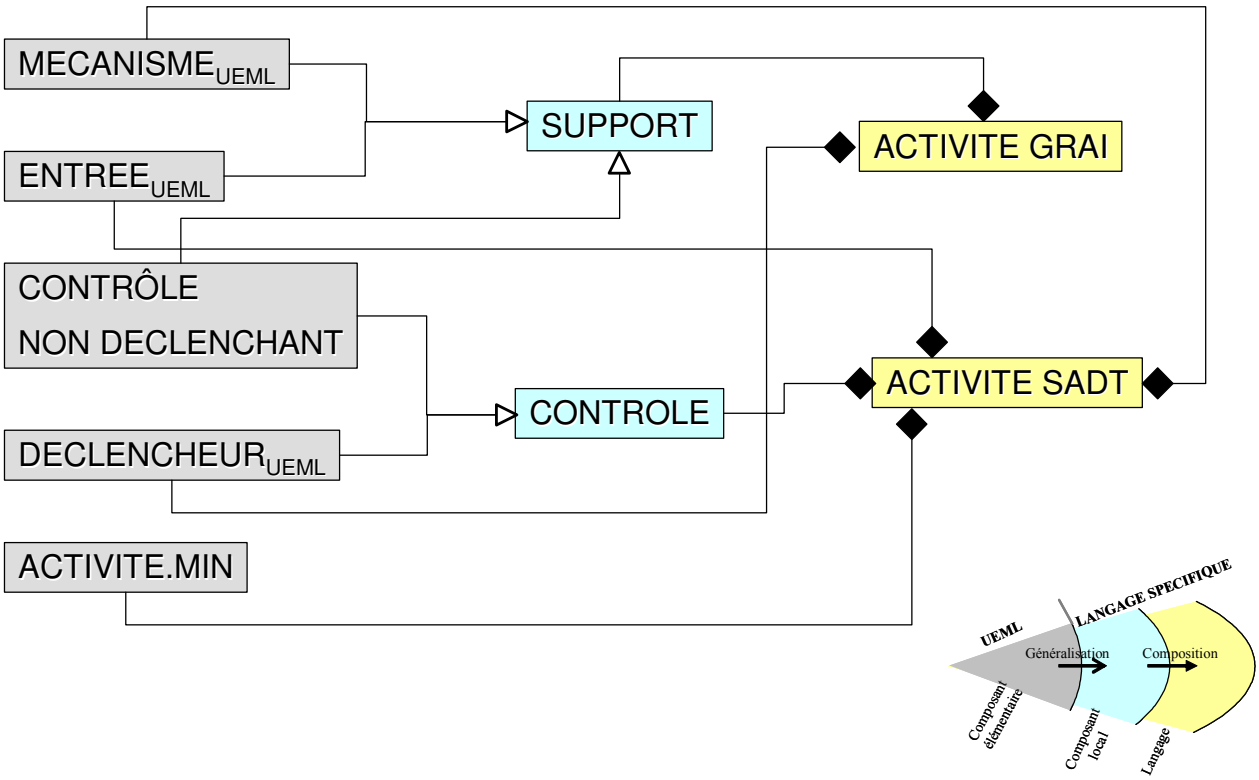


Figure 4.3-25 : Règles de correspondances

4.3.7.7 Elément de modélisation et élément de visualisation

Dans notre approche, nous distinguons (comme défini dans le méta-modèle du langage UML) deux éléments différents : les éléments de modélisation et ceux de visualisation (respectivement EM et EV). Un élément de modélisation est un composant employé pour modéliser un objet réel, il a une vraie signification dans un modèle. Il correspond au concept de signifié évoqué au paragraphe 4.2.1. Un élément de visualisation correspond à la représentation graphique d'un ou plusieurs éléments de modélisation. Cet élément est à rapprocher du concept de signifiant, également évoqué paragraphe 4.2.1. Le même élément de modélisation peut avoir plusieurs éléments de visualisation différents. Néanmoins, quelques éléments de visualisation peuvent ne pas correspondre à la représentation d'un élément de modélisation (même si ce cas n'apparaît pas souvent). Ils sont seulement utilisés pour avoir

une meilleure représentation graphique du modèle. On peut, par exemple, considérer les opérateurs de renvoi des réseaux GRAI, que nous avons vu dans le paragraphe 4.2.3 du chapitre 3 de ce mémoire. En effet, ces opérateurs de renvoi sont utilisés lorsqu'il est nécessaire d'indiquer l'origine ou la destination d'une entité lorsque cette origine ou cette destination est extérieure au schéma. On utilise ces opérateurs lorsqu'une entité circule entre le système étudié et son environnement, lorsqu'une entité, circule d'un centre de décision à un autre ou lorsque le réseau est trop grand pour rentrer sur une page. Ces opérateurs n'ont donc qu'une signification purement graphique.

Ainsi, nous proposons de considérer deux aspects différents de chaque composant :

- la signification proprement dite du composant, correspondant à l'élément de modélisation,

- la représentation graphique du composant, correspondant à l'élément de visualisation.

Un exemple, de ces deux aspects est illustré sur la Figure 4.3-26. Nous considérons dans cet exemple le composant activité GRAI.

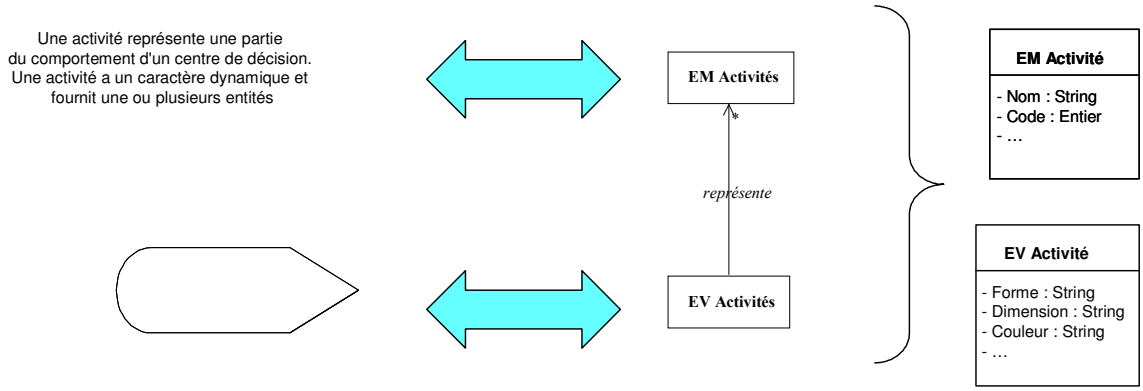


Figure 4.3-26 : Elément de modélisation et de visualisation

La Figure 4.3-27 illustre comment les deux aspects différents d'un composant sont employés dans le processus de traduction. La traduction du composant correspond à la traduction de la signification du composant (aspects sémantiques et syntaxiques). La traduction graphique correspond à la traduction de la représentation graphique du composant. Les éléments de visualisation des langages sont liés aux éléments de visualisation de UEML. La traduction des éléments de visualisation suit le même principe que la traduction des éléments de

modélisation. Chaque élément de modélisation d'UEML est lié à un ou plusieurs éléments de visualisation d'UEML. En fait, les traductions des éléments de modélisation et de visualisation sont effectuées en même temps. Chaque composant d'un langage étant considérés comme un couple < élément de modélisation, élément(s) de visualisation >.
Cependant, comme UEML n'est pas destiné à être un langage opérationnel (au moins dans un premier temps), on pourrait également considérer que la représentation graphique d'un composant n'est pas vraiment nécessaire. Cependant, dans notre approche nous prenons tout de même compte de cet aspect.

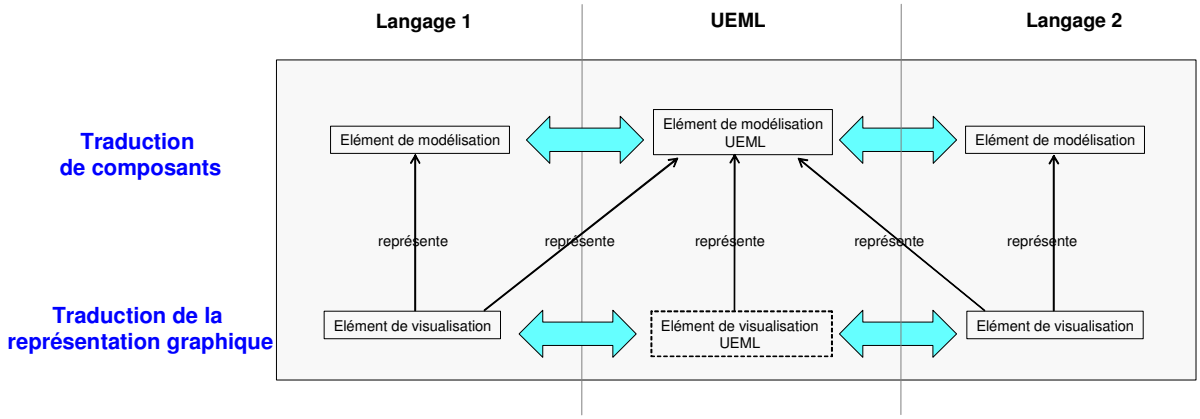


Figure 4.3-27 : Traduction du composant et de sa représentation graphique

C'est pourquoi, afin d'avoir une bonne séparation entre les éléments de visualisation et ceux de modélisation, nous proposons d'utiliser trois différents diagrammes de classes. Un premier diagramme permet de représenter uniquement les éléments de modélisation, ainsi que les liens qui existent entre ces éléments. Le second diagramme permet de représenter la même chose mais d'un point de vue des éléments de visualisation. Enfin, le dernier diagramme est utilisé afin de représenter les liens entre les éléments de visualisation et ceux de modélisation. Concernant, l'activité GRAI nous obtenons, les deux méta-modèles supplémentaires suivants :

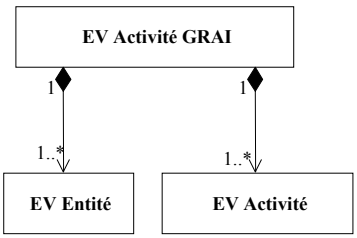


Figure 4.3-28 : Eléments de visualisation de l'activité GRAI

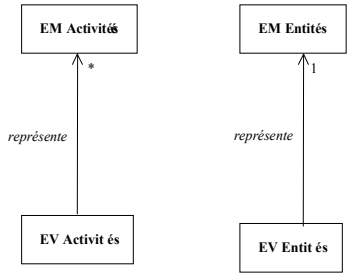


Figure 4.3-29 : Liens entre les éléments de visualisation et de modélisation de l'activité GRAI

4.3.8 Prise en compte des associations

4.3.8.1 Principes

Jusqu'à présent, dans la comparaison des composants des langages, nous avons considéré que les langages n'étaient constitués que d'un ensemble de composants indépendants. Nous n'avons, en effet, pas tenu compte des liens qui existent entre ces composants (associations), indiquant l'ensemble des interactions entre eux. Par exemple, la Figure 4.3-30, représente le méta-modèle des réseaux GRAI, prenant en compte ses associations.

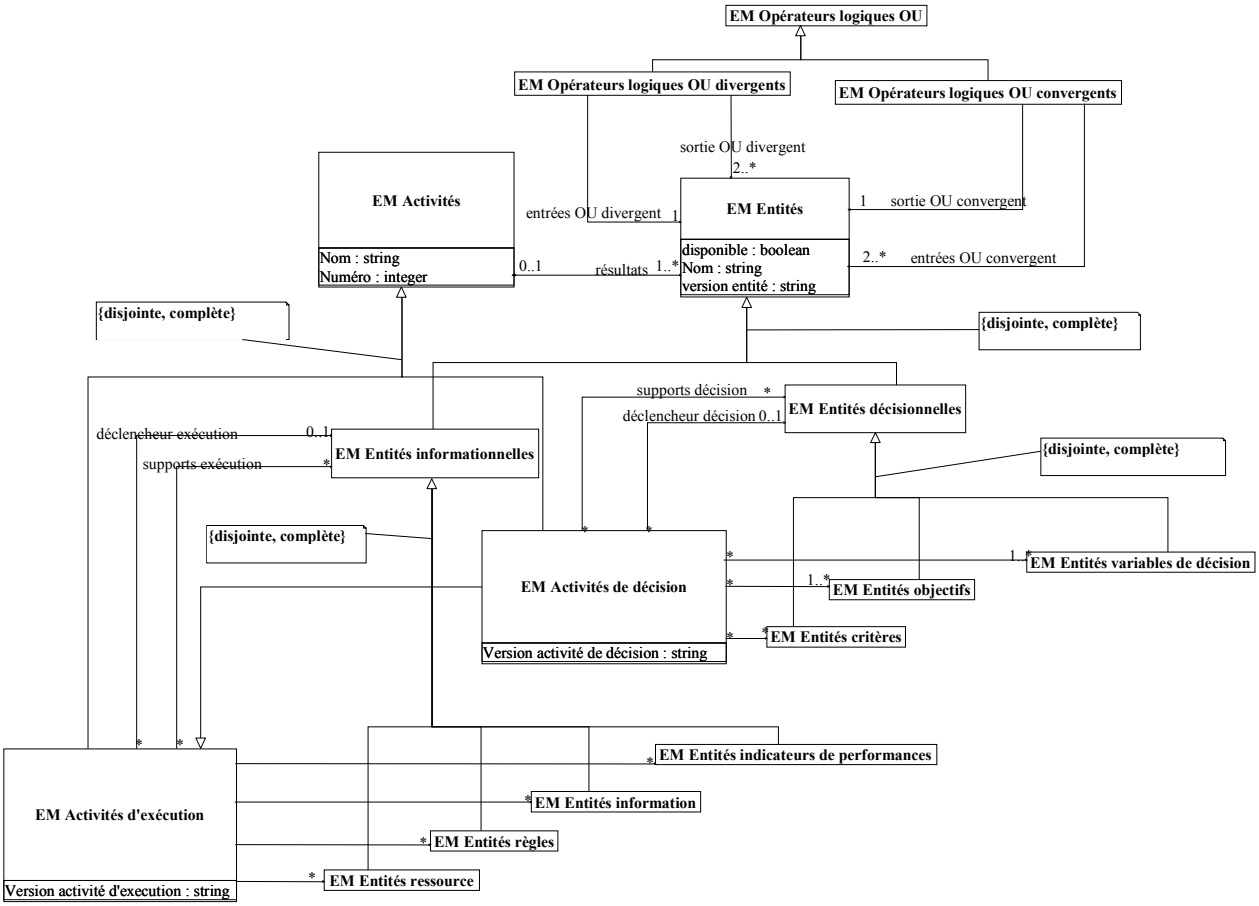


Figure 4.3-30 : Méta-modèle des réseaux GRAI

La Figure 4.3-31 illustre le lien qui existe entre une activité et une entité. Une activité peut générer une ou plusieurs entités et une entité ne peut être générée que par au plus une activité.

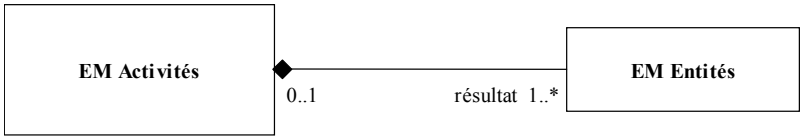


Figure 4.3-31 : Partie du méta-modèle des réseaux GRAI

La prise en compte de ces liens complique quelque peu le problème de la comparaison des composants. Nous n'allons, ici, considérer la comparaison de deux langages (LME1 et LME2) possédant deux composants chacun (respectivement X1, Y1 et X2, Y2) qui sont reliés par une association (Figure 4.3-32).

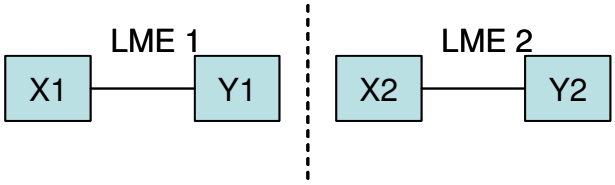


Figure 4.3-32 : Prise en compte des associations

Notation :

Nous noterons X — Y l'association liant la classe X à la classe Y.

Nous allons traiter la comparaison d'un point de vue de la généralisation. Les résultats pouvant être étendus facilement au cas de la composition.

Une association entre deux classes permet d'indiquer qu'il existe un « lien » entre deux populations d'objets. La nature du lien qui existe entre les classes est indiquée par le nom de l'association. Ainsi, dans notre exemple si on a X1 ≡ X2 et Y1 ≡ Y2 alors ceci n'indique pas que les associations X1 — Y1 et X2 — Y2 soient également équivalentes[5]. Les mêmes populations d'objets peuvent être utilisées de manière différente dans les deux langages.

[5] Nous ne nous posons pas, dans ce mémoire, le problème de savoir comment on identifie le fait que deux associations sont équivalentes.

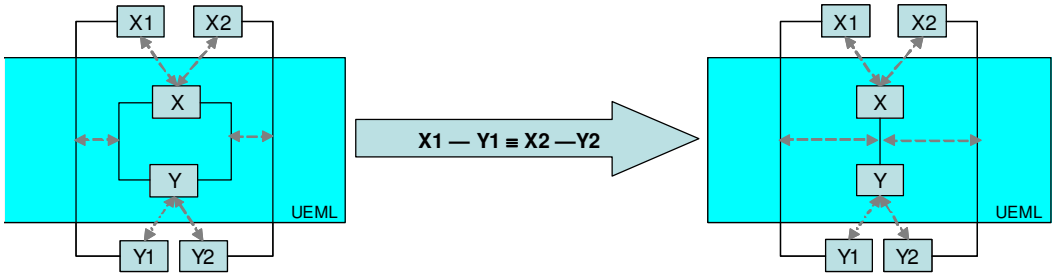


Figure 4.3-33 : Cas d'un égalité entre deux associations

Ainsi, afin de traiter le problème des associations, il faut le séparer en deux étapes. Dans un premier temps on décompose les composants en composants élémentaires en étudiant leur intersections. Cette étape nous amène par la suite à déduire des associations X1 — Y1 et X2 — Y2 celles qui vont exister entre les composants élémentaires qui vont former UEML.
Considérons la Figure 4.3-33, dans la cas où X1 ≡ X2 et Y1 ≡ Y2, alors on crée dans UEML deux composants X et Y tel que X ≡ X1 ≡ X2 et Y ≡ Y1 ≡ Y2. Ensuite, les deux associations entre X et Y peuvent se réduire à une seule si on a une équivalence entre les deux.

Nous illustrons sur la Figure 4.3-34, un autre cas où X1 ∩ X2 ≠ ∅ et Y1 ≠ Y2. Nous décomposons X1 et X2 en trois composants élémentaires A, B et C. Ainsi, nous devons également décomposer l'association X1 — Y1 en deux associations A — Y1 et B — Y1. De même, nous décomposons l'association X2 — Y2 en C — Y2 et B — Y2. Ces associations sont des sous-ensembles des associations d'où elles sont issues. Ainsi nous avons :

A — Y1 ⊂ X1 — Y1 **B — Y1 ⊂ X1 — Y1** **A — Y1 ∪ B — Y1 ≡ X1 — Y1**	**C— Y2 ⊂ X2 — Y2** **B — Y2 ⊂ X2 — Y2** **C — Y2 ∪ B — Y2 ≡ X2 — Y2**

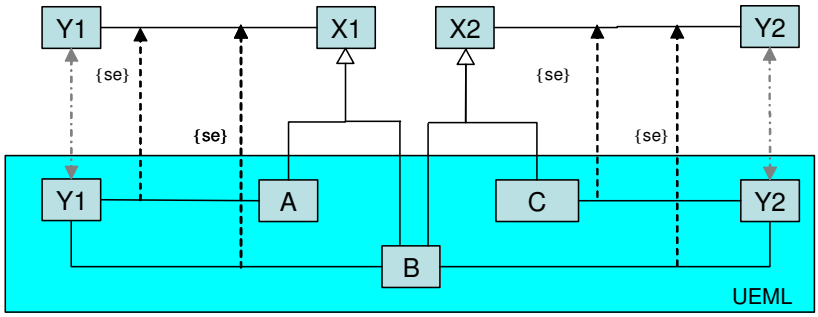


Figure 4.3-34 : Décomposition d'association

La Figure 4.3-35 illustre un cas plus général. Toutes les associations intégrées dans le langage UEML doivent être comparées aux autres afin de détecter les équivalences ou non.

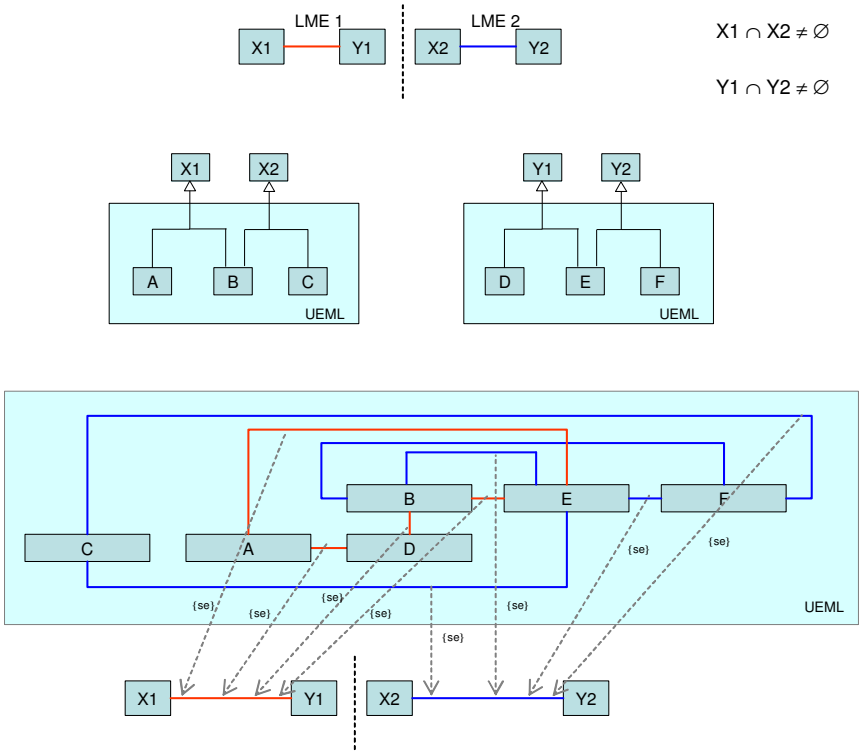


Figure 4.3-35 : Comparaisons d'association, cas plus général

4.3.8.2 Conclusion

Ainsi, nous pouvons distinguer un ensemble de cas que l'on va retrouver dans cette comparaison, en fonction des quatre cas différents que nous avons dégagés dans le paragraphe 4.3.2. Cet ensemble est présenté sur le Tableau 4.3-8.

Comparaison des composants			Associations
1	**Y1 ≠ Y2**	X1 ≠ X2	Ne présente pas d'intérêt
2	**Y1 ≠ Y2**	X1 = X2	On a deux associations : X — Y1 et X — Y2 avec X ≡ X1 ≡ X2.
3	**Y1 ≠ Y2**	X1 ∩ X2 ≠ ∅	Décomposition de X1 — Y1 en A — Y1 et B — Y1. Décomposition de X2 — Y2 en 2 — Y2 et B — Y2.
4	**Y1 = Y2**	X1 ≠ X2	Voir cas 2.
5	**Y1 = Y2**	X1 = X2	Vérifier si les associations sont équivalentes ou non.
6	**Y1 = Y2**	X1 ∩ X2 ≠ ∅	Vérifier si les associations sont équivalentes ou non.
7	**Y1 ∩ Y2 ≠ ∅**	X1 ≠ X2	Voir cas 3.
8	**Y1 ∩ Y2 ≠ ∅**	X1 = X2	Voir cas 6.
9	**Y1 ∩ Y2 ≠ ∅**	X1 ∩ X2 ≠ ∅	Mélange des cas 3 et 7

Tableau 4.3-8 : Comparaison d'associations

Nous n'avons, ici, présenté que des cas très simples de comparaisons. Des cas plus complexes apparaissent dans un cas réel, où plusieurs composants possédant plusieurs associations entre eux, doivent être comparés. De plus, nous n'avons pas tenu compte du cas où les composants X1 et Y1 par exemple sont tous les deux éléments de X2. Ce genre de comparaison, bien que suivant le même principe que celui que nous avons énoncé ici, est plus délicat à traiter.

Par ailleurs, le problème des relations de type composant-composé (composition) n'a pas du tout été abordé ici. Les résultats présentés semblent cependant généralisables à ce type de relations.

De même, nous n'avons pas parlé ici du problème des cardinalités des associations. D'une manière générale lorsque l'on désire remplacer deux associations par une seule dans le méta-modèle UEML, il est nécessaire de prendre les cardinalités qui contraignent le moins l'association (le minimum entre les cardinalités des deux associations), afin de pouvoir représenter tous les cas dans UEML. La restriction au niveau des cardinalités pour un langage donné sera incluse dans les règles de correspondances.

4.4 Ecriture formelle

Les résultats présentés précédemment, nous ont permis de mettre en évidence certaines difficultés lors de la comparaison des composants. Par ailleurs, la comparaison des méta-modèles individuels reste une tâche délicate car il existe une grande latitude dans l'utilisation d'un langage comme les diagrammes de classes. De plus, l'acte de méta-modélisation n'est pas une tâche déterministe, comme nous l'avons évoqué au paragraphe 4.2.3. Ces modèles ne comportent pas ou très peu d'éléments concernant la sémantique du langage et l'interprétation du méta-modèle repose complètement sur la compréhension de celui qui le lit. Par exemple, il est compréhensible qu'un déclencheur déclenche une activité parce qu'il s'appelle *déclencheur*.

4.4.1 Utilisation du langage OCL (Object Constraints Language)

Ainsi, l'utilisation de langages dits « formels » peut être utile afin d'établir des modèles plus précis et moins ambigu, en y ajoutant quelques contraintes mathématiques. En effet, les comparaisons effectuées entre les composants sont essentiellement faites sur la base d'une

comparaison des aspects syntaxiques. Cependant, afin d'améliorer les méta-modèles des langages, l'utilisation d'un langage tel que OCL [OMG, 2003], [Warmer *et al.*, 1999], par exemple, peut permettre d'indiquer des contraintes plus formelles et donner ainsi une meilleure description du langage.

4.4.1.1 Présentation du langage OCL (Object Constraint Language)

Les diagrammes obtenus tels que les modèles de classes ne sont pas suffisamment précis du fait de la notation graphique. Cette notation peut induire des ambiguïtés dans l'interprétation et la spécification des objets. Il faut alors nécessairement définir des contraintes sur les objets. Cela peut être effectué par des langages formels (la méthode B ou le langage Z par exemple), toutefois ces langages sont basés sur des définitions mathématiques et sont difficilement exploitables (des travaux ont été réalisés dans [Panetto *et al.*, 2002] concernant la formalisation de langages de modélisation d'entreprises en utilisant le langage UML et la méthode B). L'utilisation d'un langage naturel présente l'avantage d'être facilement compréhensible mais ne résout que partiellement les problèmes du fait de son manque de précision.
Ainsi, IBM a développé OCL[6], un langage dit « formel » qui cherche à tirer avantage des langages formels et des langages naturels. Le langage OCL est utilisé pour exprimer de manière précise des contraintes qui ne peuvent être spécifiées dans les diagrammes UML. Nous présentons dans cette partie les concepts de base du langage OCL.

OCL est un langage formel d'expressions typées, il peut être utilisé pour :

- spécifier des invariants de classes ou de types sur les diagrammes de classes,
- spécifier des pré ou post conditions sur les opérations et méthodes,
- spécifier des contraintes sur les opérations,
- naviguer,
- décrire des conditions de garde.

Le mot clé **context** définit le contexte de l'expression OCL. Les mots clés **inv**, **pre** et **post** désignent respectivement les stéréotypes « invariant », « precondition » et « postcondition ».

[6] Les informations sur le manuel de référence OCL sont disponibles sur le site de l'OMG, http://www.omg.org

Chaque contrainte OCL est écrite dans le contexte d'une instance ou d'un type spécifique, le mot clé **self** est utilisé pour faire référence à l'instance contextuelle.
Prenons l'exemple du diagramme de classes, Figure 4.3-36, adapté du manuel de référence OCL.

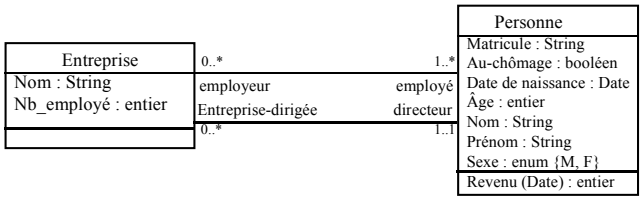


Figure 4.3-36 : Exemple de diagramme de classes

Une expression OCL peut se rapporter aux types, classes, interfaces, associations. Il en est de même pour tous les attributs, opérations, rôles et méthodes. La valeur d'une propriété (attribut, opération, rôle et méthode) d'un objet, définie dans un diagramme de classe est spécifiée par un point suivi du nom de la propriété.

```
context untype inv :
self.propriété
```

Invariant et Post (Pre)condition

Invariant, le nombre d'employés doit être supérieur à 10.

```
Context Entreprise inv :
Self.Nb_employé > 10
```

Postcondition sur le revenu des employés, l'opération revenu prend comme paramètre la date.

```
Context Personne :: revenu (d : date) : entier
Post : result = 5000
```

Il est possible de récupérer des valeurs antérieures dans les pré et post conditions. Pour faire référence à une valeur avant le début de l'opération on utilise le symbole '@'.

Par exemple lors de la mise à jour de l'âge d'une personne :

```
Context Personne :: jour-anniversaire()
Post : âge = âge@pre + 1
```

Navigation

La navigation est effectuée grâce aux rôles. L'expression objet.nom_rôle retourne un ensemble d'objets.

Exemple : Assurer que le nombre d'employés n'est pas nul

```
Context Entreprise
Inv : self.employé-> notEmpty ()
```

On peut accéder à un ensemble d'objets satisfaisant un certain nombre de critères par les opérateurs **select, reject** et **collect**. Par exemple, sélectionner l'ensemble des employés ayant plus de 50 ans.

```
Context Entreprise inv :
Self.employé->select(âge>50)
```

Lorsqu'une contrainte s'applique à tous les éléments on utilise l'opération **forAll**.

```
Context Entreprise inv :
Self.Employé->forAll(e1,e2 : Person | e1<>e2 implies
e1.matricule <> e2.matricule)
```

4.4.1.2 Exemple d'utilisation concernant l'activité GRAI

Ainsi, pour une activité GRAI, par exemple, il est possible d'indiquer avec ce type de langage, une partie de son comportement dynamique. En effet, on peut rajouter une contrainte liée à la possibilité pour une activité d' être effectuée. Le méta-modèle simplifié de l'activité de GRAI doit être modifié, en y ajoutant quelques attributs, comme illustré sur la Figure 4.3-37.

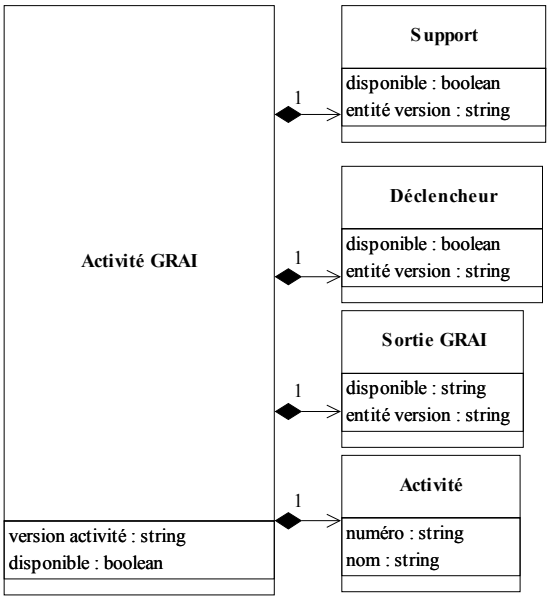


Figure 4.3-37 : Méta-modèle simplifié de l'activité GRAI pour OCL

La contrainte écrite ci-dessous en OCL indique, par exemple, qu'une activité GRAI est prête à être exécutée (disponible), seulement si tous ses supports et ses déclencheurs sont disponibles et avec le bon numéro de version. En effet, chaque entité nécessaire à l'activité est mise à jour à la fin de chaque période de la planification [Vallespir et al., 2002]. Pour cette raison, un numéro de version des entités doit être ajouté pour être sûr que c'est la bonne information et que l'activité n'a pas commencé avec des informations correspondantes à la version précédente[7].

context Activité GRAI inv :

self . disponible = 1 **implies** (self . support -> **forAll** (S : Support | S . entité version = self . version activité)
and (self . support -> **forAll** (S : Support | S . disponible = true))

and (self . déclencheur -> **forAll** (T : Déclencheur | T . entité version = self . version activité)
and (self . déclencheur -> **forAll** (T : Déclencheur | T . disponible = true)))

[7] D'autres contraintes doivent être définies afin d'indiquer, par exemple, que l'attribut « activité version » est incrémenté dès que l'activité est terminée. Pour qu'elle soit de nouveau exécutée, il est nécessaire que ses entités soient mises à jour.

Finalement, beaucoup de contraintes peuvent être ajoutées au méta-modèle afin d'avoir une vision plus complète des aspects importants à prendre en compte pour l'élaboration du méta-modèle du langage UEML. Avec ces contraintes, certains aspects sémantiques peuvent être spécifiés et des représentations trop ambiguës peuvent être évitées.

4.4.2 Formalisation des langages actuels

Cependant, même si l'on obtient une représentation totalement formelle des méta-modèles, cela n'est pas suffisant. En effet, comme nous l'avons déjà évoqué très peu de langages de modélisation d'entreprise possèdent une définition formelle de leur sémantique. Ainsi, ce manque engendre qu'un même composant, peut être utilisé dans un langage pour représenter des situations différentes. Il en résulte qu'un modèle réalisé dans un langage peut être ambigu et que si un autre analyste reprend le modèle, il ne comprendra pas forcément ce que le premier analyste a voulu modéliser. Par exemple, dans la majorité des langages actuels de modélisation par processus, l'utilisation des opérateurs logiques ne permet pas de lever toutes les ambiguïtés d'interprétation des modèles construits. La nécessité de lever les ambiguïtés subsistantes se voit par ailleurs renforcée par le désir (issu de pratiques industrielles de la modélisation des processus) de pouvoir réaliser une évaluation de performances sur la base des modèles obtenus. En effet, formaliser l'organisation des entreprises est une nécessité pour obtenir une meilleure efficacité et leur permettre d'évoluer avec le marché et l'environnement socio-technique. C'est ainsi que la modélisation des processus fait aujourd'hui partie des outils les plus émergents du domaine. Cependant, l'enrichissement continu des composants portés par ces méthodes, associé à la multiplicité croissante des langages utilisés est à l'origine d'ambiguïtés dans la construction et la compréhension des modèles. Ce constat ne remet généralement pas en cause la puissance de description de l'outil, mais caractérise plutôt une insuffisance dans la proposition de règles ou propriétés conduisant à une interprétation unique des modèles [Deschamps *et al.*, 2004].

Il est donc nécessaire de lever toutes les ambiguïtés contenues dans les langages de modélisation d'entreprise que l'on désire intégrer dans le langage UEML, avant de faire la recherche des composants élémentaires. En effet, réaliser le méta-modèle d'un langage d'une manière formelle alors que celui-ci n'est pas défini de façon formelle, ne présente pas grand intérêt.

Par ailleurs, beaucoup de langages de modélisation d'entreprise possèdent plusieurs vues de modélisation. Chaque vue permet de se focaliser sur un certain nombre d'aspects du système étudié. Cependant, généralement le domaine de modélisation couvert par chaque vue possède des intersections. Bien que ce phénomène soit normal, il en résulte tout de même un problème de cohérence globale au sein du langage. Il est en effet souvent difficile de savoir si telle ou telle information dans une vue est utilisée dans une autre vue et si oui comment, sous qu'elle forme et à quel endroit elle apparaît. Il est indispensable de traiter ce problème sur chaque langage que l'on désire intégrer dans UEML. Il faut procéder pour ainsi dire à l'élaboration d'un *« UEML interne »* de chaque langage en étudiant les intersections des composants appartenant à chaque vue. Si cette étape n'est pas réalisée le développement d'un langage unifié de modélisation d'entreprise est voué à l'échec, car les règles de correspondances intégrées ne seraient pas complètes.

4.5 Conclusion

Nous avons, dans ce chapitre, exposé un ensemble de principes relatifs à la comparaison des composants. Nous avons ainsi distingué le cas de l'inclusion de type ensembliste de celle de type composant-composé. Ainsi, les principes que nous avons vus au paragraphe 4.3 du chapitre 3, peuvent être étendus en prenant en compte ces aspects. Les composants locaux à un langage étant donc, soient liés par une relation de composition, soient par une relation de généralisation (Figure 4.5-1).

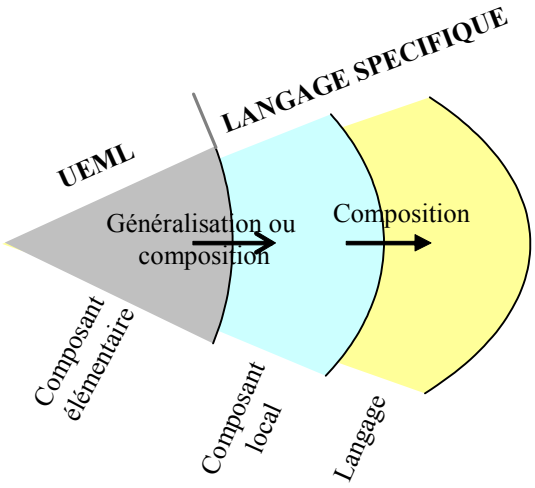


Figure 4.5-1 : Relations entre les composants élémentaires, locaux et les langages (2)

Nous avons illustré ces principes au travers d'un exemple très simple. Nous avons également montré que le développement d'un langage unifié nécessite, d'une part, une représentation formelle des méta-modèles, mais d'autre part, il faut que les composants des langages soient définis de manière plus formelle.

Cependant, l'exemple présenté ici, est un exemple très simple sur beaucoup de points. D'une part, seulement une partie (l'activité) des langages considérés (les réseaux GRAI et SADT) a été traitée. D'autre part, l'analyse de chaque langage est exprimée dans une liste de composants. De plus, les liaisons (associations) dont nous en avons dit seulement quelques mots, n'ont pas été considérées dans cet exemple.

Néanmoins, le cas pragmatique de la traduction présenté ici permet d'identifier certaines difficultés liées à UEML. Travailler sur ces aspects peut constituer la première étape vers la définition d'UEML. Mais il est évident que les étapes suivantes, traitant un problème de taille significative (langages complets, en considérant plus de deux langages et tenant compte des aspects syntaxiques et sémantiques) sont beaucoup plus difficiles et demandent davantage d'efforts. Les Méta-modèles établis et les travaux effectués dans le cadre du réseau thématique UEML montrent ces difficultés dans un cas réel, en prenant en compte trois langages de modélisation d'entreprise (la vue processus GRAI, IEM et EEML). Les aspects sémantiques et syntaxiques y sont traités, dans une certaine mesure. Les résultats de ces travaux sont présentés dans le chapitre suivant de ce mémoire.

Chapitre V

Exemple d'élaboration d'un UEML

Chapitre V

Exemple d'élaboration d'un UEML

5.1 Introduction

5.1.1 Objectifs

Le réseau thématique UEML (Unified Enterprise Modelling Language) UEML [IST - 2001 – 34229] a été lancé pour contribuer à la solution des problèmes engendrés par la multiplicité des Langages de Modélisation d'Entreprise. Il a débuté en mars 2002 et s'est terminé en juin 2003. L'objectif poursuivi à long terme était de définir un noyau d'UEML qui servirait de langage pivot à plusieurs Environnement Logiciel de Modélisation d'Entreprise. Ce langage fournira à la communauté de la modélisation d'entreprise :

- Un langage commun et éventuellement graphique, qui pourra être utilisé avec la plupart des Environnements Logiciel de Modélisation d'Entreprise commerciaux,

- des mécanismes standardisés permettant le partage et l'échange de modèles entre projets, de manière à surmonter les limites des outils et à réduire la dépendance envers ceux-ci.

A cet effet, ce réseau thématique avait pour but de contribuer à l'émergence d'un ensemble d'institution visant à :

- Créer un consensus au niveau Européen, dans le cadre des efforts de standardisation en cours, sur un langage de modélisation commun devant faciliter l'interopérabilité,

- construire un démonstrateur d'UEML pour promouvoir, tester, valider et améliorer les constructeurs du langage proposés,

- préparer le lancement d'un projet pour définir, implémenter et promouvoir un langage complet.

On peut finalement résumer les résultats attendus du projet par les six points suivants :

Résultats d'un point de vue de la recherche proprement dite :

1. Cadre d'évaluation des approches de modélisation d'entreprise,
2. besoins en composants de modélisation d'entreprise,
3. premier ensemble de composants de modélisation d'entreprise.

Résultats concernant la dissémination :

4. Portail de démonstration UEML
5. réseau UEML d'environ 80 membres,
6. proposition de développements ultérieurs.

La suite des travaux se déroule actuellement dans le cadre du Réseau d'Excellence INTEROP (Interoperability Research for Networked Enterprises Applications and Software ; 6ème PCRD, IST 508011).

5.1.2 Structuration

Le projet était structuré en cinq groupes (Work Package) de tâches dépendantes (Figure 5.1-1).

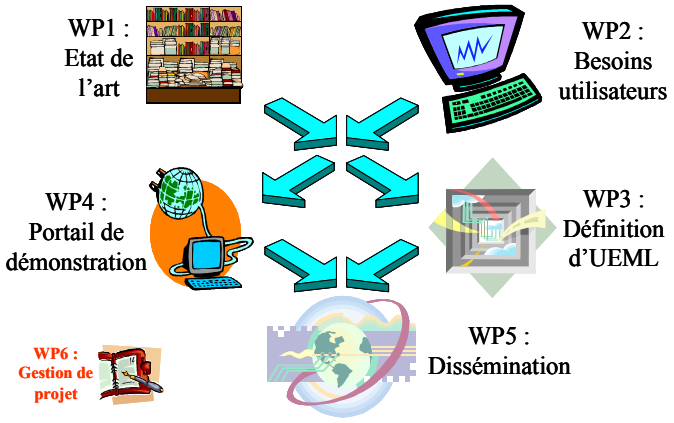


Figure 5.1-1 : Structuration du projet en terme de Work package

5.2 Elaboration d'un UEML

Les données de ce chapitre proviennent d'une part de l'ensemble des délivrables réalisés dans le Work package 3 du projet UEML, auquel nous avons participé, et d'autre part de [Berio et al., 2004].

5.2.1 Introduction

Le réseau thématique UEML étant un projet pilote, il avait pour objectif de démontrer la faisabilité de la définition d'un UEML. Par conséquent, ses résultats en terme d'objets de méta-modélisation se limitent à une première version d'UEML (UEML 1.0), définie sur la base de trois langages crées par trois des partenaires du projet (Tableau 5.2-1).

Langage de Modélisation d'Entreprise	Environnement Logiciel de Modélisation d'Entreprise	Fournisseur
IEM	MOOGO	Fraunhofer, IPK, Berlin
EEML	METIS	Computas, Lysaker
GRAI	eMAGIM	Graisoft SA, Bordeaux

Tableau 5.2-1 : Langage de modélisation d'entreprise retenus pour le projet UEML

5.2.2 Démarche adoptée

Comme nous l'avons dit précédemment, les langages de modélisation d'entreprise ne se sont pas tous définis de manière formelle. Ainsi, il est difficile de comparer entre eux les composants de chaque langage. De ce fait, lors du réseau thématique UEML, il a été décidé d'utiliser un scénario (une situation concrète de référence à modéliser), dans le but de faciliter ces comparaisons. Ce scénario a été à l'origine défini dans le langage IEM. Ensuite, il a été modélisé dans chacun des trois langages considérés. Cette tâche a été réalisée par des experts de modélisation de chacun des langages.

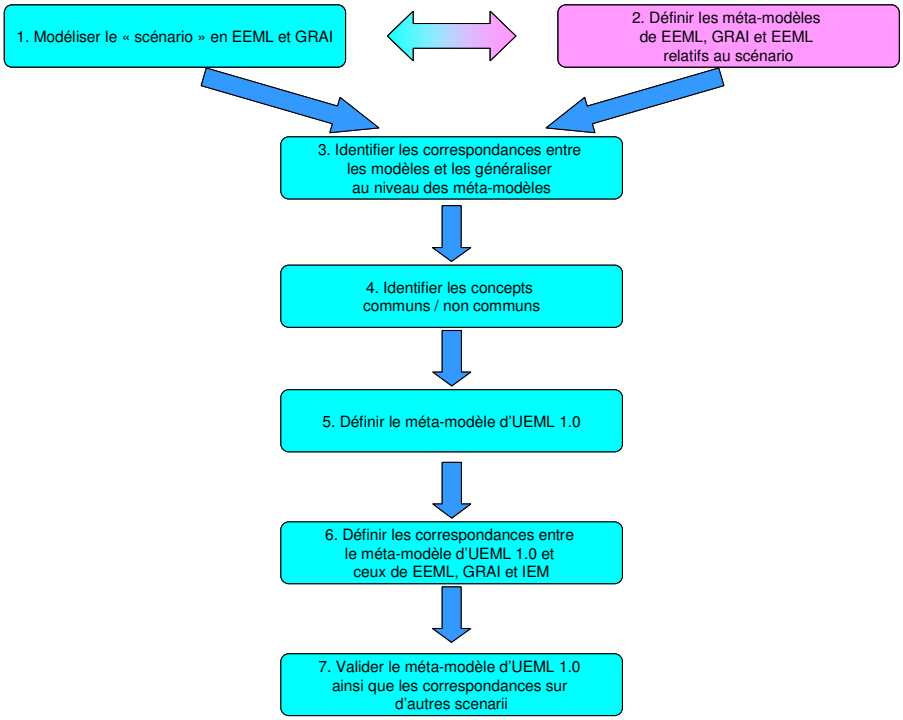


Figure 5.2-1 : Démarche adoptée pour UEML 1.0

Ensuite, chaque langage a été représenté par un méta-modèle, réalisé en utilisant le diagramme de classe UML. Les méta-modèles ne représentent que les composants des langages relatifs au scénario, ceux non nécessaires pour représenter le scénario n'apparaissant pas dans le méta-modèle. A ce niveau là, un couple auteur/vérificateur a été défini pour chaque langage, afin de valider les méta-modèles ainsi réalisés. Une recherche des correspondances entre les différents modèles du scénario réalisés dans les trois langages considérés a été effectuée. Les correspondances entre les modèles ont été recherchées en prenant en compte des paires de modèles, c'est-à-dire en comparant les langages GRAI et IEM, EEML et IEM, puis EEML et GRAI. Ces correspondances ont ensuite été généralisées au niveau des méta-modèles. Les

composants communs à ces langages ont été recherchés à partir de ces méta-modèles. A partir de ces composants communs, le méta-modèle d'UEML 1.0 a pu être défini et représenté à l'aide des diagrammes de classe UML. Par la suite, les correspondances entre ce méta-modèle et les méta-modèles des langages EEML, GRAI et IEM on été définies. Enfin, une vérification finale est faite sur les correspondances entre les UEMLs et les langages considérés (par exemple sur la base de nouveaux scénarios). La démarche présentée précédemment est résumée sur la Figure 5.2-1.

5.3 Présentation des langages

5.3.1 IEM (Integrated Enterprise Model)

Le langage IEM a été développé par l'IPK du Fraunhofer Institut de Berlin [Spur *et al*., 1996]. Il s'agit d'un langage de modélisation d'entreprise orienté objet tentant d'intégrer les points de vue fonctionnel et informationnel. Il décrit également le concept de ressource et de contrôle.

5.3.1.1 Composants du langages IEM

Le langage IEM repose sur trois classes génériques d'objets d'entreprise : le produit, l'ordre et la ressource. La notion centrale d'activité (action en IEM) est basée sur la définition de l'activité de IDEF0 à laquelle il ajoute une définition sémantique des entrées/sorties sous forme d'objets des types précédents. Il s'agit en quelque sorte d'un IDEF0/IDEF1 orienté objet auquel on ajoute des caractéristiques de workflow, afin de décrire les processus d'entreprise comme des chaînes d'activités.

Notamment, le langage IEM distingue trois types de processus :

- Les processus relatifs aux ordres (représentent les contrôles, au sens de l'actigramme SADT, dans une entreprise) (Figure 5.3-1),

Figure 5.3-1 : Représentation graphique des ordres

- les processus relatifs aux produits (tous les objets qui sont transformés par les activités d'une entreprise) (Figure 5.3-2),

Figure 5.3-2 : Représentation graphique des produits

- les processus relatifs aux ressources (le ressource est un moyen qui est exigé pour exécuter un processus) (Figure 5.3-3),

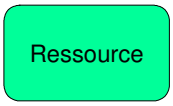


Figure 5.3-3 : Représentation graphique des ressources

- Les informations précédentes sont utilisées ou générées par une « **action** », qui désigne une activité ou un processus pour le langage IEM (Figure 5.3-4).

Figure 5.3-4 : Représentation graphique d'une action

Séquentialité :

Si deux actions sont reliées par une flèche (Figure 5.3-5) alors l'action suivante doit attendre que l'action précédente soit terminée pour commencer.

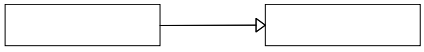


Figure 5.3-5 : Représentation graphique de la séquentialité

Séparation :

Une fois que l'action est terminée le processus peut continuer par l'exécution de l'une ou l'autre (ou les deux) des actions suivantes. Ce composant représente un « choix » dans un processus où plusieurs situations peuvent se présenter (Figure 5.3-6).

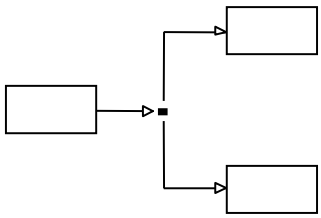


Figure 5.3-6 : Représentation de la séparation

Décision :

Une décision doit être prise pour savoir laquelle des deux suivantes (voir les deux) sera exécutée(s) une fois que l'action précédente sera terminée. Ce composant représente un « choix » comme dans le cas précédent, mais on suppose ici, que le « choix » est sujet à une prise de décision (Figure 5.3-7).

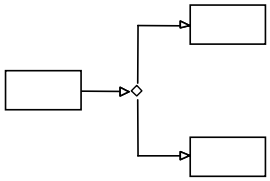


Figure 5.3-7 : Représentation de la décision

Jonction :

Ce composant permet de représenter un rendez-vous de flux dans un processus (Figure 5.3-8).

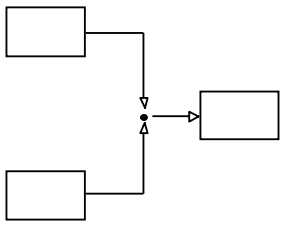


Figure 5.3-8 : Représentation de la jonction

5.3.1.2 Illustration d'un modèle réalisé en IEM

La Figure 5.3-9 représente un exemple de processus réalisé avec le langage IEM.

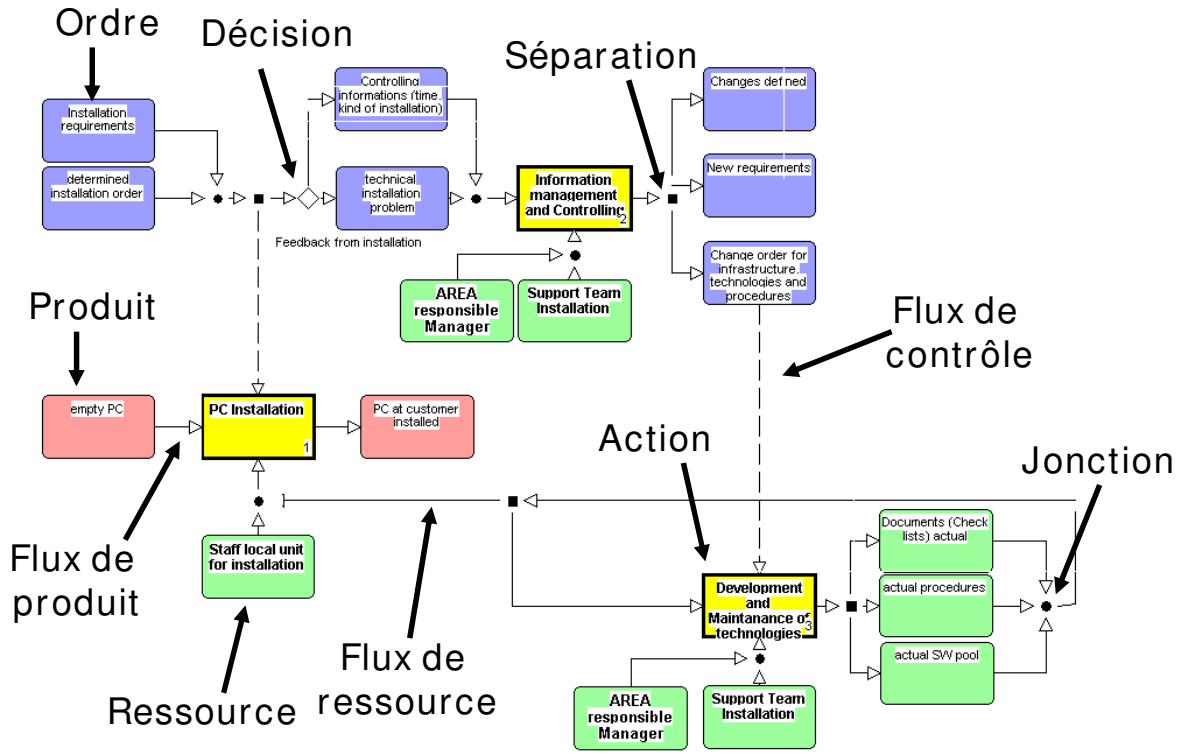


Figure 5.3-9 : Illustration d'un modèle réalisé en IEM

5.3.2 EEML (Extended Entreprise Modelling Language)

EEML a été développé dans le projet européen EXTERNAL [IST-1999-10091], comme prolongation d'APM (Action Port Model).

Les composants principaux contenus dans EEML sont :

- Tâche (et sous-tâches) avec un port d'entrée et de sortie,
- Point de décision (ET, OU, non spécifié),
- Rôles (Personne-rôle, Organisation-rôle, Outil-rôle, Objet-rôle),
- Ressources (personnes, organismes et groupes de personnes, outils (manuel et logiciel), objets (matériel et information),

5.3.2.1 Composants du langage EEML

L'objet général du langage EEML représente ce qui est commun à tous les termes et relations dans EEML. Les propriétés communes incluent *le nom* et *la description.*

Le composant principal est la **tâche** (Figure 5.3-10). Une tâche représente une partie du travail exécuté par un processus. Un rapport de décomposition est défini sur la tâche, fournissant la possibilité de décomposer une tâche en tâches plus petites. Une tâche ne peut être la décomposition que d'une tâche, et ne peut pas être la décomposition d'elle-même.

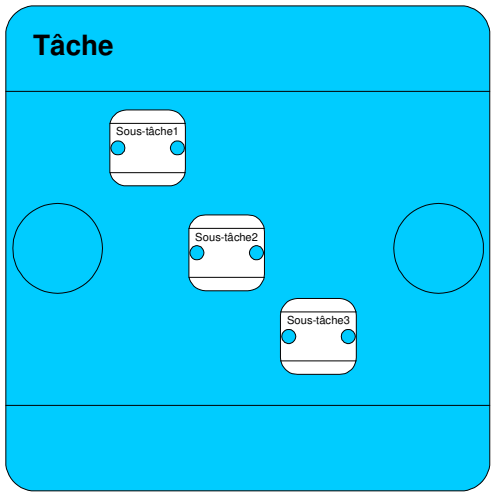


Figure 5.3-10 : Représentation graphique d'une tâche dans le langage EEML

Les composants « **rôles** » représentent les ressources développées, utilisées ou dépensées dans la tâche. Une ressource peut jouer plusieurs rôles différents suivant la tâche qui l'utilise.

Les objets de type « point de décision abstrait » incluent le port d'entrée et le port de sortie de la tâche. Un port d'entrée et un port de sortie sont obligatoires pour toutes les tâches, et sont employés pour représenter les conditions afin de commencer et de finir la tâche. D'autres points de décision peuvent également être inclus dans une tâche.

Port d'entrée

Le composant port d'entrée représente un composant obligatoire de toute tâche, indiquant les flux qui ont besoin d'être présents avant de commencer la tâche. Une tâche possède un et seulement un port d'entrée. Si des relations logiques plus avancées doivent être modélisés, celles-ci doivent l'être en utilisant des points de décision supplémentaires dans la tâche (voir plus loin).

Port de sortie

Le composant port de sortie représente un composant obligatoire de toute tâche, indiquant les flux qui doivent être envoyés avant de finir la tâche. Une tâche possède un et seulement un port de sortie. Si des relations logiques plus avancées doivent être modélisés, alors on doit utiliser des points de décision supplémentaires dans la tâche.

Le port d'entrée et le port de sortie sont sémantiquement inséparables de la tâche à laquelle ils appartiennent, par conséquent aucun d'eux ne peut être explicitement créé par l'utilisateur. De même, ni l'un ni l'autre ne peut être supprimé séparément.

Point de décision

Le terme de point de décision est employé pour représenter les points de décision dans un processus qui ne sont ni un port d'entrée ni un port de sortie.
Un point de décision représente une décision manuelle effectuée au sein du processus. La décision est en partie prise sur la base des flux entrant au point de décision. Le comportement du point de décision est en grande partie défini par la propriété *Relation logique* qui décrit comment manipuler des flux multiples venant ou sortant du point de décision.

La *relation logique* peut prendre une des trois valeurs suivantes :

- **ET :** Tous les flux d'entrée doivent être activés pour que le point de décision soit activé. Alors tous les flux sortants sont activés.

- **OU exclusif :** Dès qu'un des flux d'entrée est activé, le point de décision est activé. Alors exactement un des flux sortants est activé.

- **Non spécifié:** relatif au comportement des flux entrants et sortants multiples. La décision est prise par autre chose que les flux entrants (le plus souvent c'est l'utilisateur).

5.3.2.2 Illustration d'un modèle réalisé en EEML

La Figure 5.3-11 représente un exemple de processus réalisé avec le langage IEM.

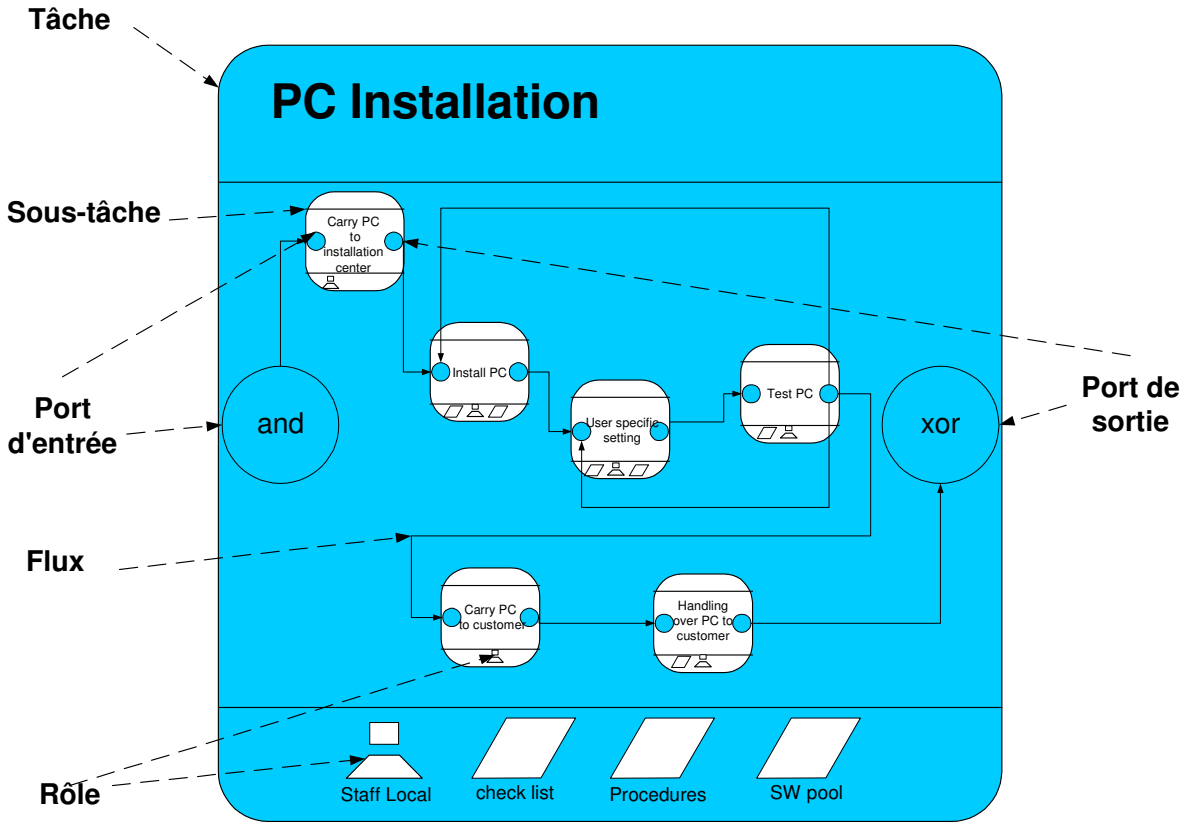


Figure 5.3-11 : Illustration d'un modèle réalisé en EEML

5.3.3 L'actigramme étendu : GRAI

L'actigramme étendu [Doumeingts *et al*., 2001] est un langage de modélisation par processus qui est fortement basé sur l'actigramme défini dans SADT, auquel quelques composants ont été ajoutés (tels que les opérateurs logiques et le concept de séquence).

5.3.3.1 Composants de l'actigramme étendu

Les activités étendues sont reliées par des liens de séquencement sur lesquels sont attachés des flux de produits ou d'informations. Les ressources (humaines ou matérielles), sont les moyens permettant d'effectuer la transformation indiquée par l'activité étendue.

Les activités sont soumises à des contrôles qui cadrent ou contraignent leur mise en œuvre ou leur exécution et sont déclenchées par des déclencheurs (périodiques ou événementiels).

Activités Etendues

Une activité étendue définit la transformation d'une entité d'entrée en une entité de sortie. Elle est représentée par un rectangle entourant un verbe décrivant la transformation effectuée par l'activité étendue.

Les principaux attributs de l'activité étendue sont :

- Son nom et son code,
- son périmètre,
- sa mission,
- son responsable,
- ses règles de fonctionnement,
- son coût (valeur min., max. ou moyenne),
- son délai de réalisation (valeur min., max. ou moyenne).

Les Ressources

Les ressources sont les moyens permettant d'effectuer la transformation des entités d'entrée en entités de sortie. Les ressources sont attachées à une activité étendue par un lien sur le côté inférieur du rectangle. Elles sont représentées par une icône et un nom. Les ressources disponibles (humaines et matérielles) sont gérées par types de ressources.

Les attributs caractérisant une ressource sont :

- Le type de ressources (humaine, matérielle, financière, information),
- le nom,
- la fonction,
- la localisation,
- la compétence ou la caractéristique technique,
- le coût unitaire.

Liens de séquencement

Les liens de séquencement relient deux activités étendues entre elles. Le lien s'attache sur le côté droit du rectangle représentant la première activité étendue et sur le côté gauche de celui représentant l'activité étendue suivante. La flèche en fin de lien détermine l'activité étendue suivante.

Les Flux de produits ou d'informations

Les flux de produits ou d'informations sont des échanges d'entités entre deux activités étendues. Ils sont assimilés aux liens de séquencement. En fait, l'existence d'un flux de produits ou d'informations se manifeste par l'affichage d'une étiquette sur le lien de séquencement et de la nature du flux de produits ou d'informations.
De plus, un flux de produits ou d'informations peut provenir de l'extérieur du processus modélisé ou être expédié vers l'extérieur. Dans ce cas, la provenance ou la destination sera identifiée par un connecteur.

Les attributs des flux de produits ou d'informations sont :

- Le nom,
- Le support,
- Sa nature (produit ou information).

Les Contrôles

Les contrôles sont des flux d'informations particuliers qui peuvent faciliter ou contraindre la transformation d'une activité étendue. Ils sont connectés sur le haut de l'activité étendue. Ils peuvent provenir d'une autre activité étendue (en sortie) ou de l'extérieur du processus (dans ce cas, la provenance sera identifiée par un connecteur).
En tant que flux d'informations, ils possèdent les mêmes attributs que ceux ci.

Les Déclencheurs

Les déclencheurs sont des contrôles particuliers qui déclenchent l'exécution d'une activité étendue. Ils sont soit à l'entrée soit sur la partie supérieure de l'activité étendue et se

différencient des contrôles par un trait plus épais. Ils peuvent provenir d'une autre activité étendue (en sortie) ou de l'extérieur du processus (dans ce cas, la provenance sera identifiée par un connecteur). Si le déclencheur est un événement périodique, il n'est pas représenté. En tant que contrôles particuliers, les déclencheurs en possèdent les attributs.

Les opérateurs logiques

- Le **ET synchrone**, positionné en entrée de plusieurs activités étendues, indique que ces activités seront lancées en même temps et effectuées en parallèle. Il est représenté par une double barre dont les extrémités sont bornées.

- Le **ET asynchrone,** positionné en entrée de plusieurs activités étendues, indique que ces activités seront effectuées en parallèles mais qu'elles pourront être lancées à des moments différents. Il est représenté par une double barre non bornée.

- Le **OU exclusif** indique un aiguillage dans le déroulement du processus. Plusieurs activités étendues sont proposées à la suite d'un OU mais une seule sera choisie et la suite du processus se fera à partir de cette activité. Il sera représenté par une barre simple.

Les Connecteurs

Les connecteurs sont utilisés pour représenter l'extérieur du processus modélisé. Ainsi, on pourra représenter les échanges d'informations ou de produits avec l'extérieur du processus modélisé, que ce soit en entrée, en contrôle ou en sortie. Trois sortes de connecteurs sont proposées :

- Le **connecteur processus** référence un processus modélisé de l'étude. Il est représenté par une forme en « diamant »,

- Le **connecteur externe** représente une entité à l'extérieur du domaine de l'étude, c'est à dire non modélisable sous forme de processus. Il est représenté par une ellipse en trait simple,

- Le **connecteur interne** représente une entité à l'intérieur du domaine étudié, mais non modélisable sous forme de processus. Il est représenté par une ellipse en trait épais.

5.3.3.2 Illustration d'un modèle réalisé en actigramme étendu

La Figure 5.3-12 représente un exemple de processus réalisé avec le langage IEM.

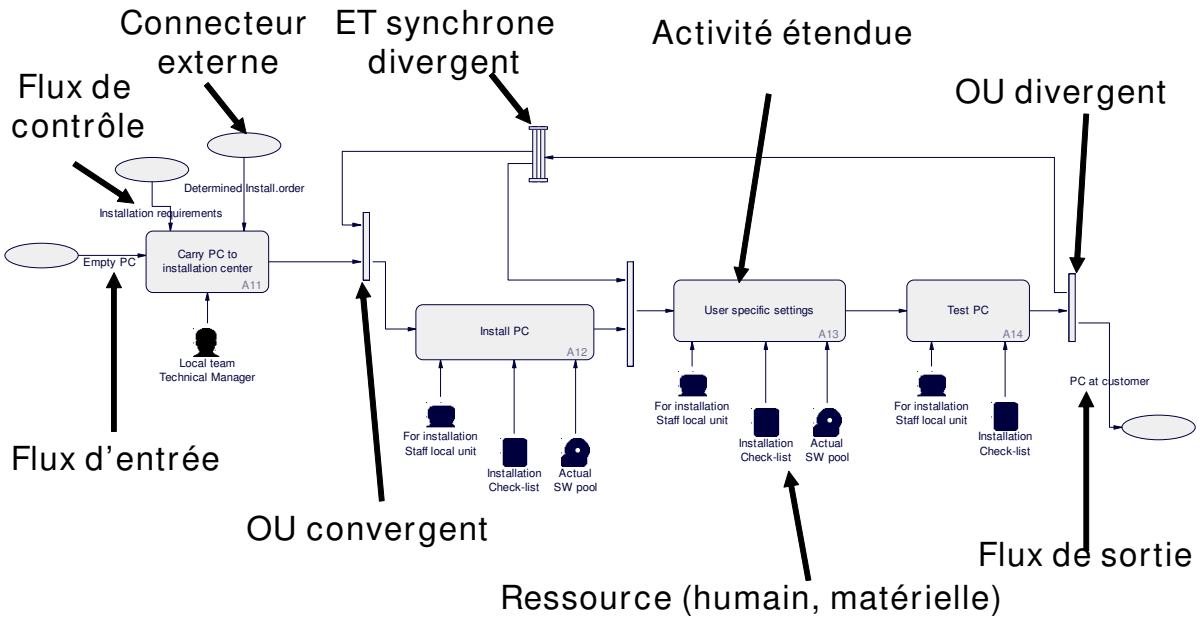


Figure 5.3-12 : Illustration d'un modèle réalisé en actigramme étendu

5.4 Modélisation du scénario

Le scénario a été modélisé dans les langages GRAI et EEML, le scénario étant à l'origine modélisé en IEM. Nous présentons ici un extrait des modèles du scénario en se focalisant sur la décomposition d'une même activité, l'activité « PC installation » du scénario. Ces différents modèles sont présentés dans les paragraphes suivants.

- Modèle en IEM : Figure 5.4-1,
- modèle en EEML : Figure 5.4-2,
- modèle en actigramme étendu : Figure 5.4-3.

5.4.1 IEM

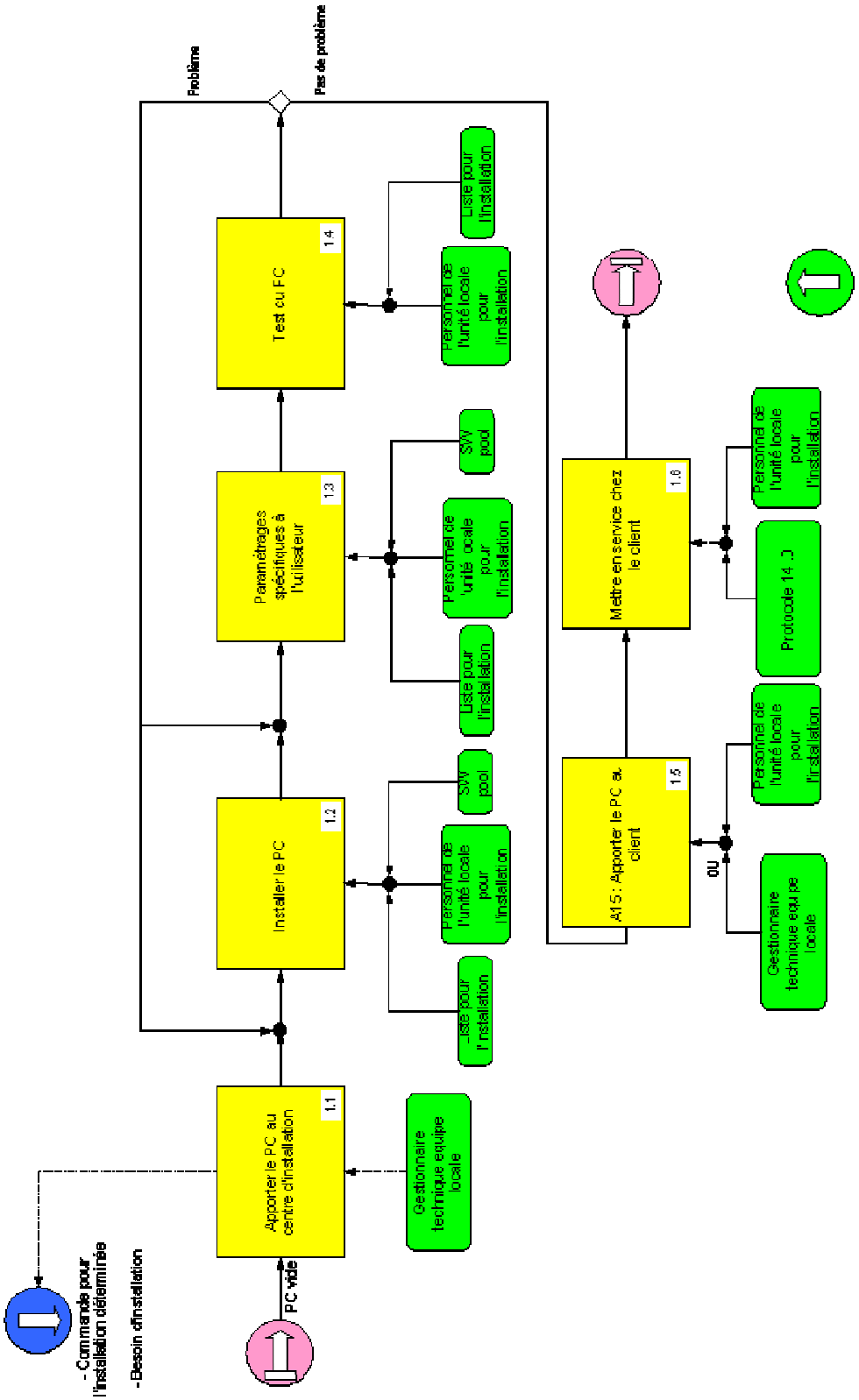


Figure 5.4-1 : Modèle de « PC installation » en IEM

5.4.2 EEML

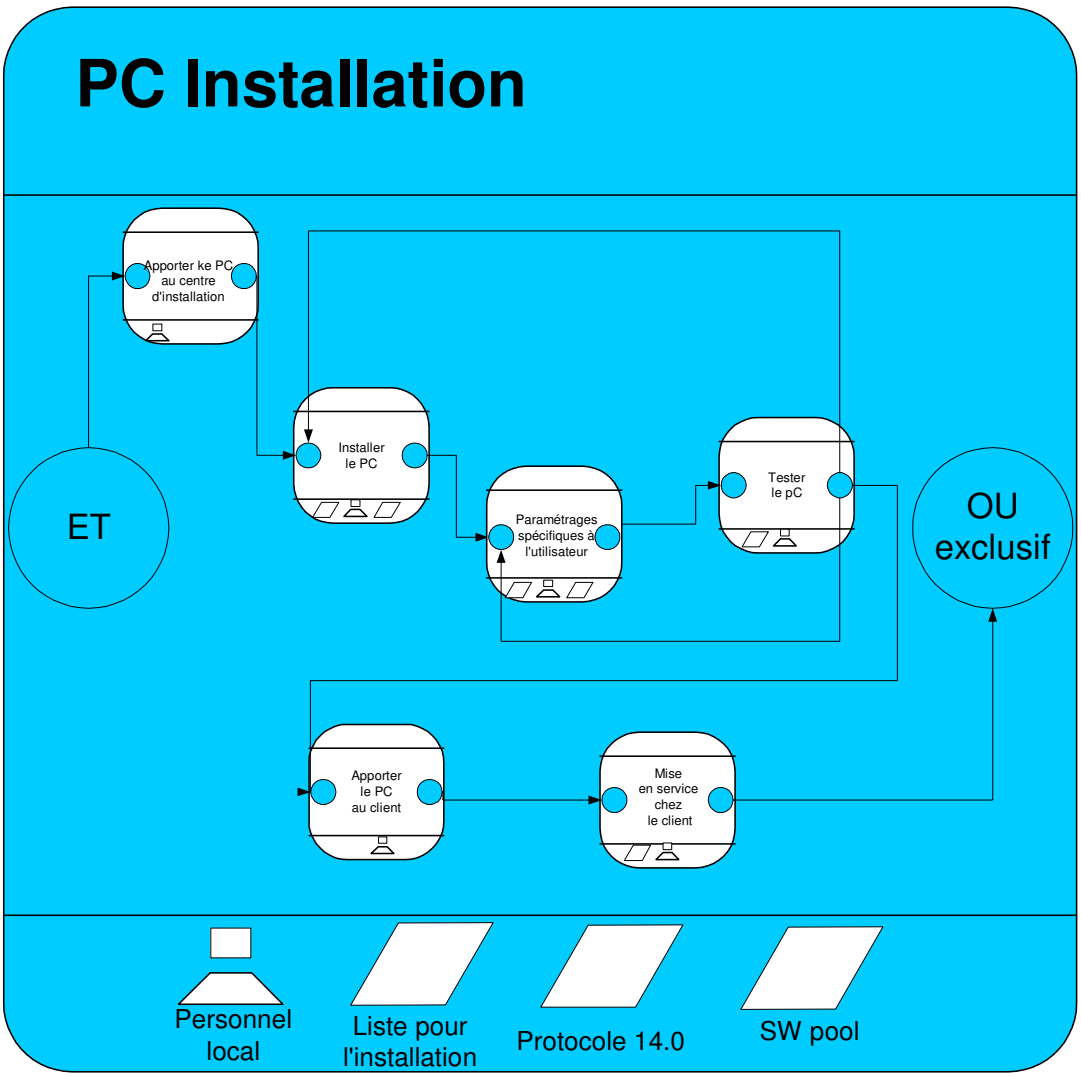


Figure 5.4-2 : Modèle de « PC installation » en EEML

5.4.3 GRAI : L'actigramme étendu

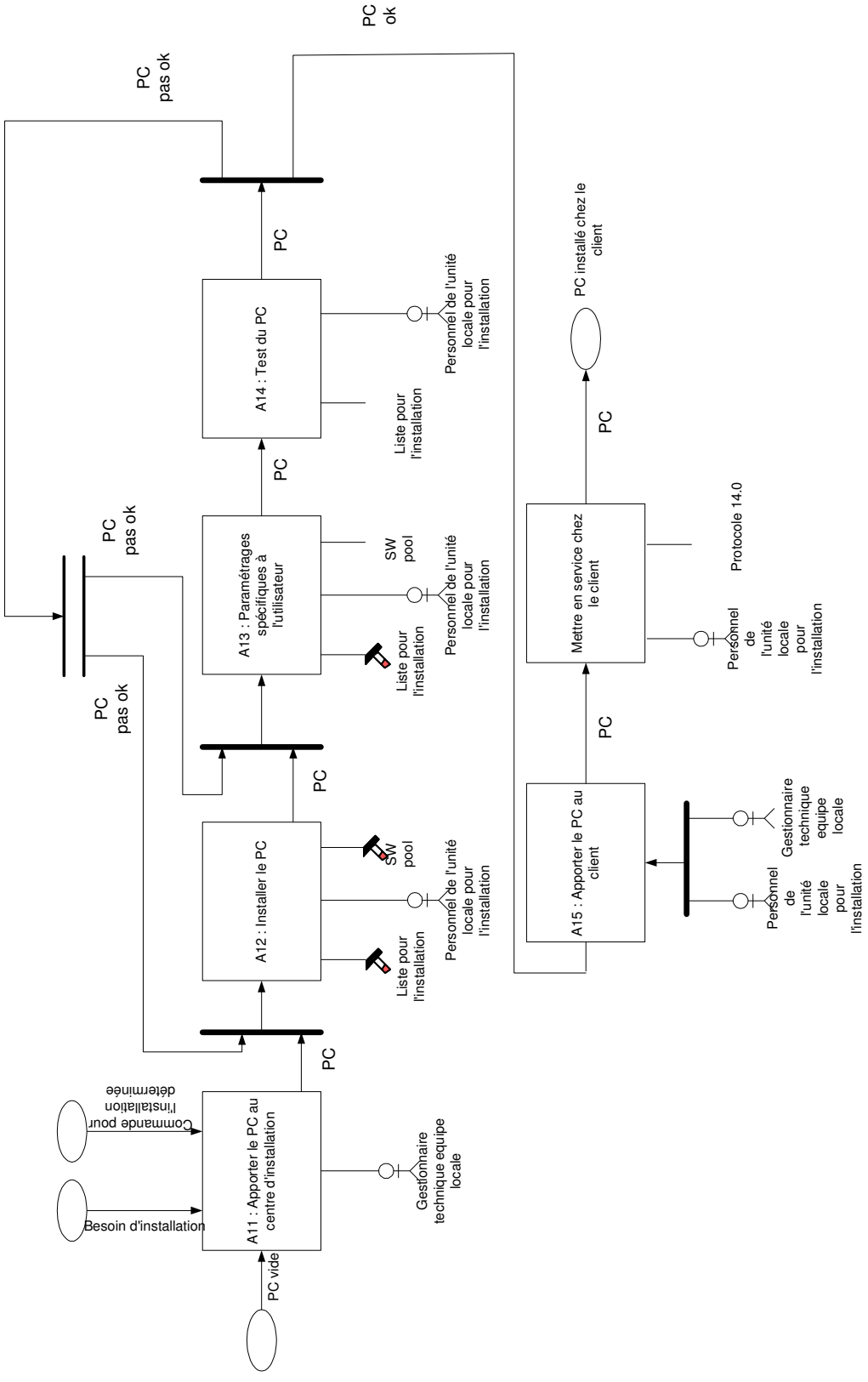


Figure 5.4-3 : Modèle de « PC installation » en actigramme étendu

5.5 Méta-modèle des langages

Chaque langage a donc été représenté par un méta-modèle. Ce méta-modèle a été construit à l'aide du diagramme de classe UML. Pour chaque langage un couple auteur/vérificateur a été défini afin de d'éliminer les erreurs d'interprétation, l'auteur étant un expert du langage. Ces différents méta-modèles sont présentés dans les paragraphes suivants.

- Méta-modèle d'IEM : Figure 5.5-1,
- méta-modèle d'EEML : Figure 5.5-2,
- méta-modèle de l'actigramme étendu : Figure 5.5-3.

5.5.1 IEM

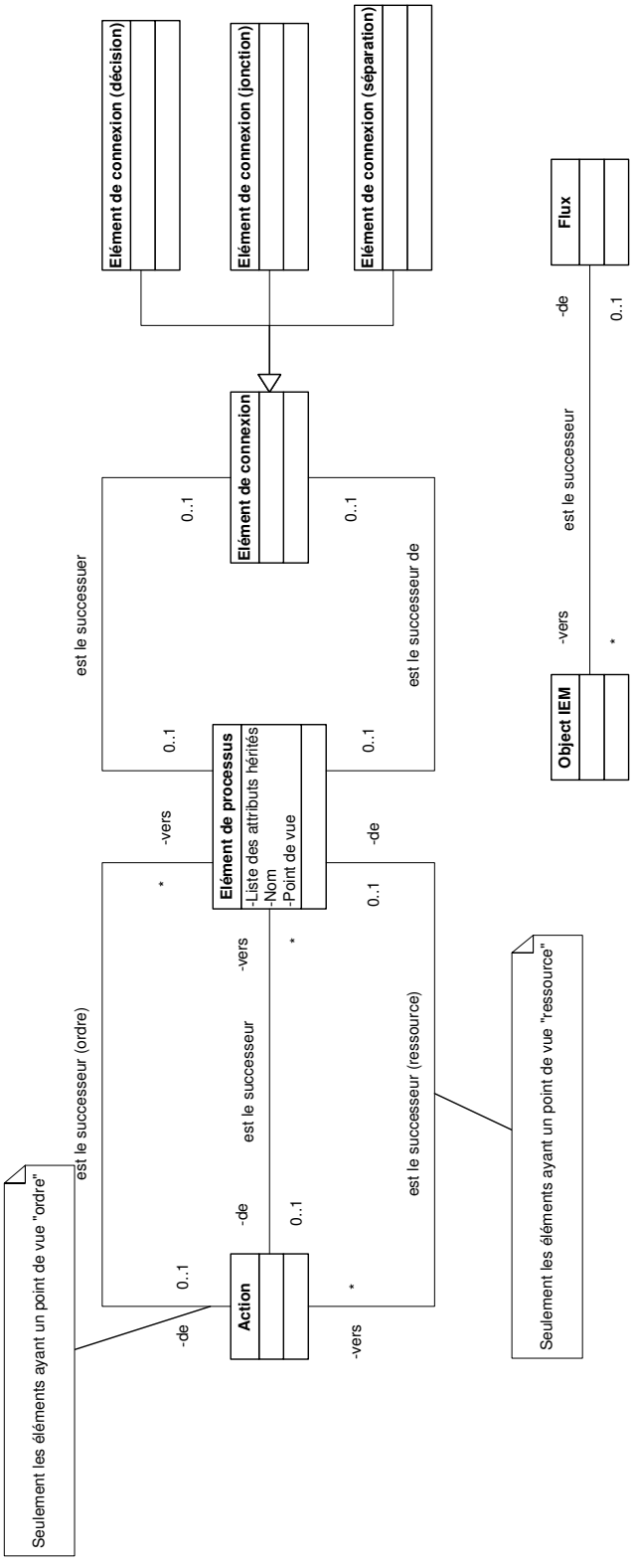


Figure 5.5-1 : Méta-modèle du langage IEM

5.5.2 EEML

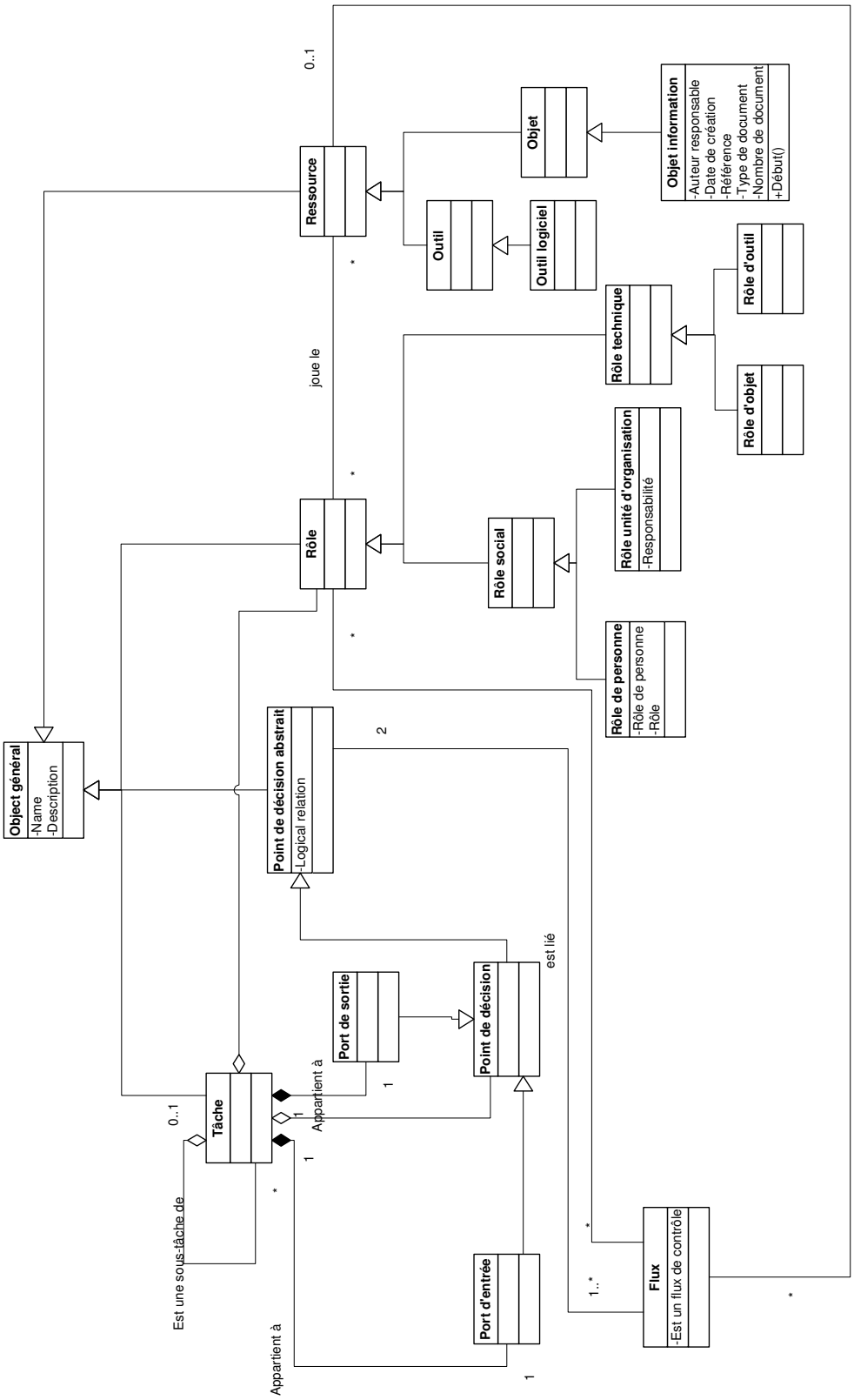


Figure 5.5-2 : Méta-modèle du langage EEML

5.5.3 GRAI : L'actigramme étendue

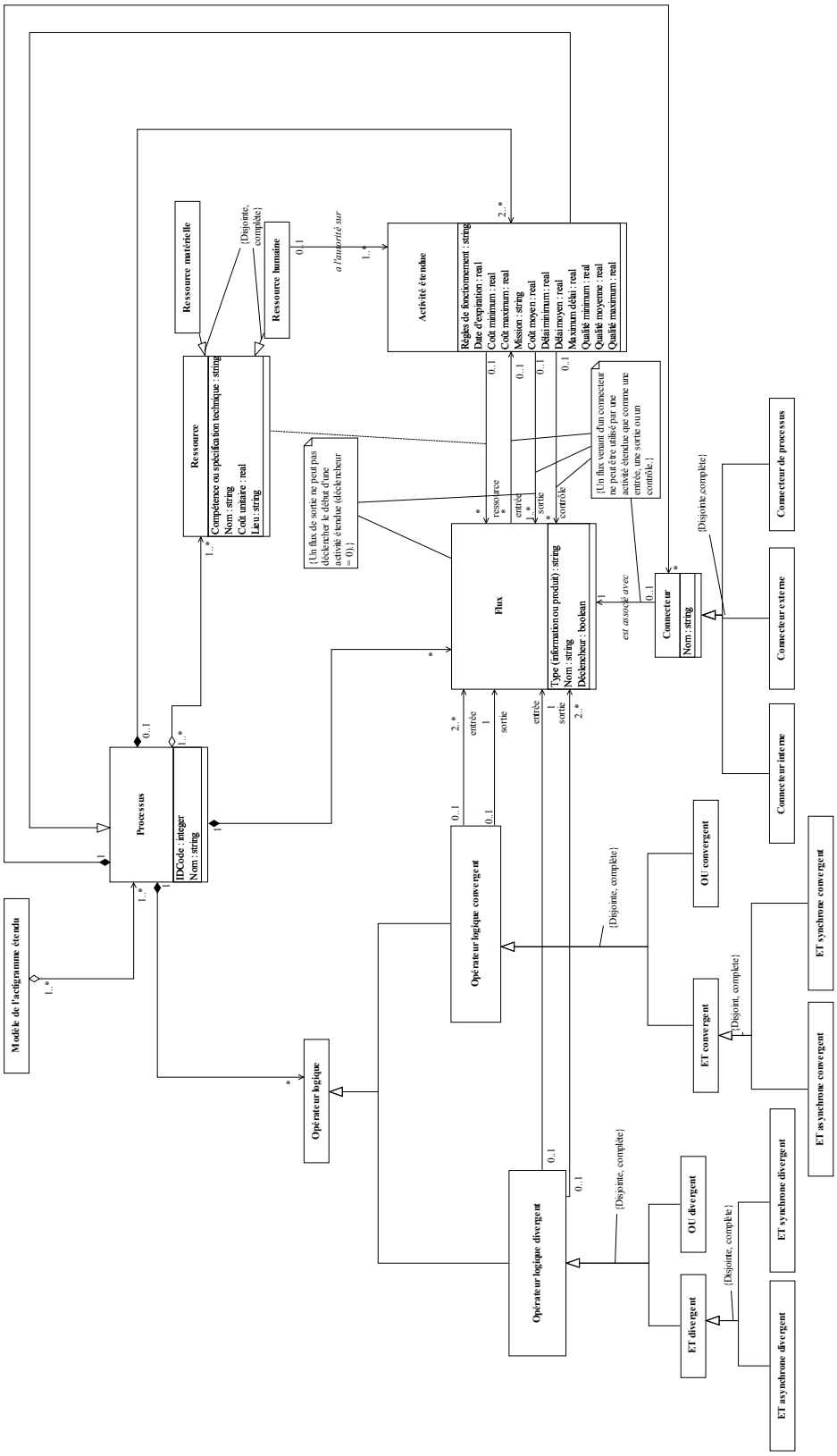


Figure 5.5-3 : Méta-modèle du langage actigramme étendu

5.6 Détermination des composants communs et non communs

Les correspondances sémantiques entre deux composants C1 et C2 peuvent être caractérisées de la façon suivante :

- **C1 = C2**

  C1 est équivalent à C2

- **C1 ⊇ C2**

  C1 est toujours utilisé où C2 est utilisé

- **C1 ∩ C2 ≠ ∅**

  C1 est quelque fois utilisé où C2 est utilisé, mais pas toujours

Ainsi, nous retrouvons les mêmes cas que ceux identifiés dans le chapitre 4.

Comme nous l'avons déjà dit, il est important de faire la distinction entre la comparaison qui s'effectue au niveau du langage et au niveau du modèle. En effet, comme indiqué sur la Figure 5.6-1, il est nécessaire de séparer ces deux niveaux.

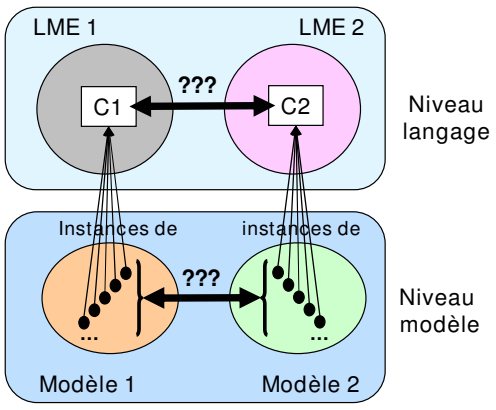


Figure 5.6-1 : Comparaison au niveau du modèle et au niveau du langage [BERIO *et al.* 2004]

Dans cette approche, on commence par comparer l'utilisation des composants en comparant les modèles du scénario élaborés dans chaque langage considéré. A ce niveau là, il n'est pas possible de conclure qu'il y ait une équivalence entre deux composants au niveau des langages (et donc des méta-modèles) à partir d'une équivalence des composants au niveau des

modèles (c'est à dire que pour un scénario donné deux composants appartenant à deux langages sont utilisés pour représenter la même situation du scénario). Afin de pouvoir affirmer une telle chose, il faudrait qu'il y ait égalité sur une infinité de scénarios différents. On peut cependant supposer, d'une manière raisonnable, qu'il y a effectivement égalité si cette équivalence se vérifie sur un nombre significatif (non infini) de scénario. Dans le cadre de ce réseau thématique, les premières comparaisons ont été réalisées sur un seul scénario. Cependant, il faudrait vérifier ces correspondances avec d'autres scénarios afin de pouvoir généraliser. Ainsi, pour définir le schéma global d'UEML, les composants ont été classifiés de la manière suivante:

- Les composants **équivalents** entre tous les méta-modèles, sont qualifiés de **composants communs,**

- Les composants **équivalents** à au moins deux méta-modèles seulement, sont qualifiés de composants **non communs,**

- des composants qui n'**appartiennent qu'à un seul langage** sont également qualifiés de **composants non communs**,

- Les composants qui ne sont pas **significatifs** ne sont pas **gardés.**

- Lorsque qu'il y a **chevauchement** on a besoin d'approfondir l'analyse : l*es composants se chevauchant* peuvent être réduits *à des composant équivalents* en contraignant (spécialisation) leur utilisation. Cette spécialisation peut se diviser en deux cas :

    - il est possible de définir explicitement des contraintes sur les classes originelles du méta-modèle ce qui caractérise avec précision les composants équivalents,

    - la spécialisation est liée à certaines décisions qui doivent être prises par l'analyste et dont les réponses ne sont pas explicites dans le scénario.

La comparaison des composants ne tient pas compte ici, des relations composant-composé que nous avons mis en évidence dans le chapitre précédent. Le chevauchement des composants n'est en effet traité que d'un point de vue de la généralisation (spécialisation).

5.7 Equivalence entres les composants

Ainsi, on peut avec les règles définies dans le paragraphe précédent, identifier les correspondances entre les méta-modèles. La recherche des correspondances est effectuée en prenant en compte des paires de modèles, c'est-à-dire en comparant les langages GRAI et IEM, EEML et IEM puis EEML et GRAI. Ces correspondances ont ensuite été généralisées au niveau des méta-modèles. Les composants communs à ces langages ont été recherchés à partir de ces méta-modèles.

5.7.1 GRAI et IEM[8]

Dans le Tableau 5.7-1, sont indiquées les relations que l'on peut identifier entre les composants de l'actigramme étendu (GRAI) et de IEM.

GRAI (Classe)	Relation	IEM (classe)	Description
Processus	Equivalent	Action	Processus et activité étendue ne sont pas bien distinguées dans le méta-modèle GRAI.
Activité étendue	Equivalent	Action	Processus et activité étendue ne sont pas bien distinguées dans le méta-modèle GRAI.
Flux (avec entrée, sortie et nom)	Chevauchement	Produit	Tous les produits ne sont pas des flux et inversement.
Ressource	Sous-ensemble	Ressource (avec ressource successeur)	Des ressources peuvent juste être utilisées pour des flux entrée/sortie.
Flux (avec entrée, sortie et nom)	Sur-ensemble	Ressource (avec successeur)	Flux (avec nom) ne fait pas nécessairement référence à une ressource; cependant tous les ressources utilisées comme entrée ou sortie d'une action sont représentées comme des flux avec entrée, sortie et nom.
Flux (avec entrée, sortie	Chevauchement	Ordre	Il y a des ordres qui sont

[8] **Note sur la syntaxe utilisée :** Le mot-clé **avec** est employé pour indiquer qu'un attribut ou une association doit avoir une valeur.
Ex. **Le flux (avec le nom)** signifie que le flux doit fournir une valeur pour son **attribut** nom ou un lien avec une classe **nom**.
Par ailleurs, dans la colonne description, on trouve de temps en temps la remarque « à vérifier ». Cette remarque indique qu'une étude plus approfondie de l'équivalence des composants devrait être effectuée.

et nom)			représentés comme des flux avec contrôle
Flux (avec contrôle et nom)	Chevauchement	Commande	Il a plus de flux
Opérateur logique	Chevauchement	Elément de connections (séparation, jonction, Décision)	Certains opérateurs logiques qui ne sont pas des éléments de connections et inversement)
Opérateur logique	Chevauchement	Port	
Connecteur	Chevauchement	Port	Certains ports ne sont pas des connecteurs et inversement). Il y a également plus de connecteurs que de ports dans IEM.

Tableau 5.7-1 : Correspondances entre les composants de GRAI et de IEM

5.7.2 EEML et IEM

Dans le Tableau 5.7-2, sont indiquées les relations que l'on peut identifier entre les composants de EEML de IEM.

EEML (Classe)	**Relation**	**IEM (classe)**	**Description**
Port d'entrée	Chevauchement	Port (avec entrée)	
Port de sortie	Chevauchement	Port (avec sortie)	
Ressource/Outil/Objet/ Objet informationnelle /Outil logiciel	Chevauchement	Ressource	A vérifier
Rôle	Sur-ensemble	Objet IEM	A vérifier. Rôle et objet IEM sont utilisés pour représenter le même ensemble d'objets.
Tâche	Sur-ensemble	Action	A vérifier. Toutes les Action state sont représentées comme des tâches dans EEML mais il y a une action state qui n'existe pas dans EEML (action class en IEM n'est qu'un mécanisme de réutilisation).
Point de décision	Equivalent	Elément de connexion	
	Sur-ensemble	Jonction	
	Sur-ensemble	Décision	
	Sur-ensemble	Séparation	
Flux	Sous-ensemble	La relation de succession identifiée entre Action, Jonction, décision, séparation, processus, état d'objet IEM et flux.	

Tableau 5.7-2 : Correspondances entre les composants de EEML et de IEM

5.7.3 GRAI et EEML

Dans le Tableau 5.7-3, sont indiquées les relations que l'on peut identifier entre les composants de EEML et de l'actigramme étendu (GRAI).

EEML (Classe)	Relation	GRAI (classe)	Description
Tâche	Sur-ensemble de	Activité étendue	Une tâche peut-être utilisée comme processus dans GRAI
	Sous-ensemble	Processus	Un processus peut-être utilisé pour représenter le niveau le général du modèle.
Port d'entrée	Chevauchement	Opérateur logique	Parfois, car en GRAI il est possible d'avoir plusieurs flux entrant dans une activité étendue. Un opérateur logique est utilisé au niveau détail supérieur de l'activité pour joindre ces flux.
	Chevauchement	Connecteur	Il y a des connecteurs qui ne sont pas des ports d'entrée et inversement.
Port de sortie	Chevauchement	Opérateur logique	Parfois, car en GRAI il est possible d'avoir plusieurs flux entrant dans une activité étendue. Un opérateur logique est utilisé au niveau détail supérieur de l'activité pour joindre ces flux.
	Chevauchement	Connecteur	Il y a des connecteurs qui ne sont pas des ports de sortie et inversement.
Point de décision	Chevauchement	Opérateur logique	
Rôle	Equivalent	Ressource	Une classe Ressource existe aussi en EEML
Point de décision abstrait	Chevauchement	Opérateur logique	Dérivé car la classe point de décision abstrait est une méta-classe d'autre classe
Flux	Pas de relation	Flux (avec ressource)	Un flux associé avec une ressource en GRAI n'est pas un flux en EEML mais une association entre tâche et rôle.
Flux	Chevauchement	Flux (avec entrée, sortie et nom)	
Rôle (avec flux)	Chevauchement	Opérateur logique	
Flux	Chevauchement	Flux (avec contrôle)	

Tableau 5.7-3 : Correspondances entre les composants de GRAI et de EEML

5.8 Méta-modèle d'UEML 1.0

Ainsi, à l'aide des relations que nous venons d'identifier entre les composants des trois langages de modélisation d'entreprise que nous considérons, nous pouvons construire le méta-modèle du langage UEML. Les composants d'UEML sont donnés par l'équation :

Composants UEML = Composants Communs +
(quelques) Composants Non Communs

En effet, dans ce réseau thématique le langage UEML n'a été défini que pour permettre de traduire un modèle écrit dans un langage dans un autre langage. Ainsi, un composant n'est retenu pour être dans le méta-modèle d'UEML que s'il appartient à au moins **deux des langages considérés** (il est donc soit un composant non commun, soit un composant commun).

Le méta-modèle obtenu pour UEML est illustré sur la Figure 5.8-1.

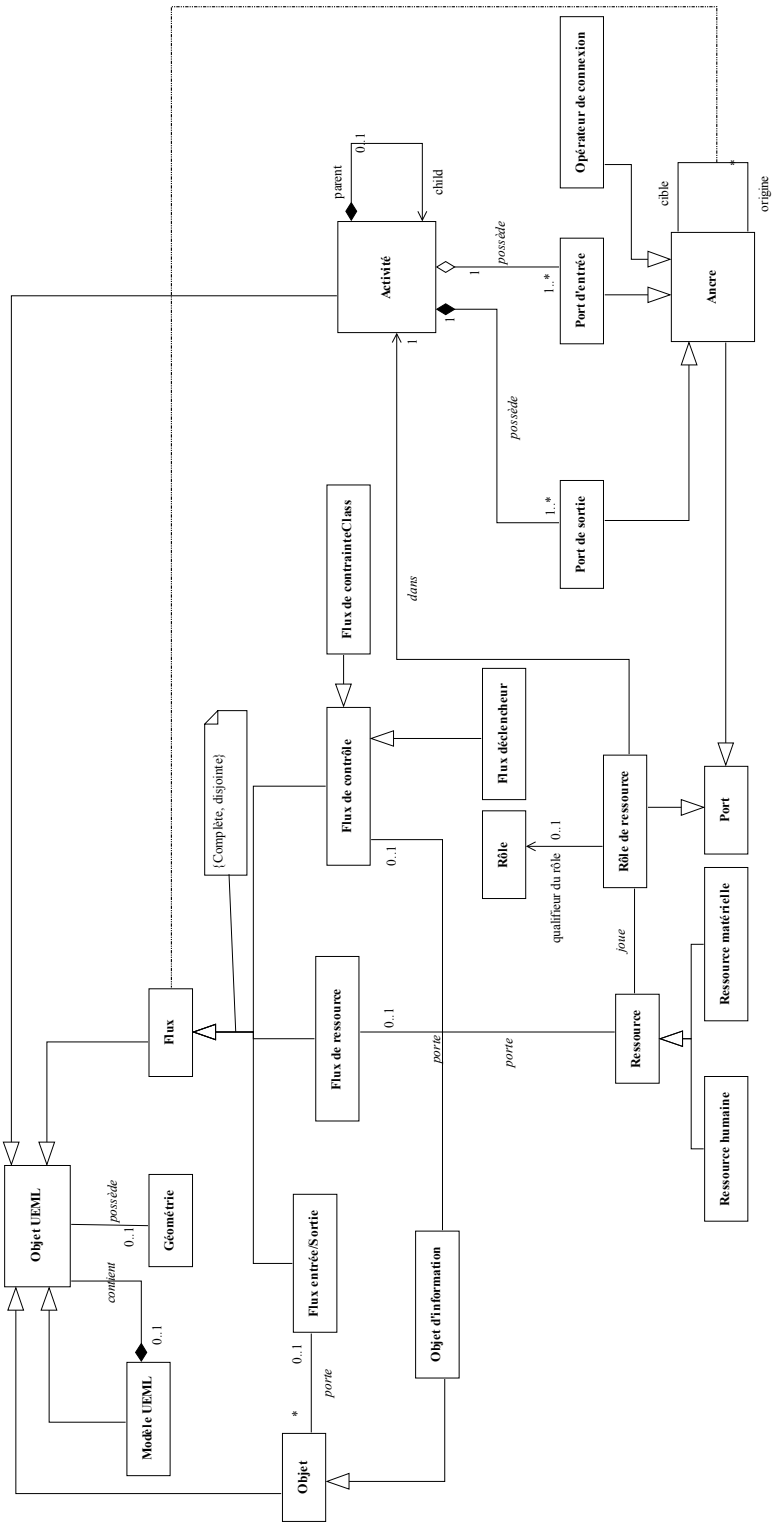


Figure 5.8-1 : Méta-modèle d'UEML 1.0

5.9 Correspondances entre IEM, EEML, GRAI et UEML 1.0

Les règles de correspondances entre les composants du langage UEML 1.0 et les composants des langages IEM, EEML et GRAI utilisés pour modéliser le scénario, sont représentés dans le Tableau 5.9-1.

Composants communs UEML	GRAI	IEM	EEML
Activité	Activité étendue	Action	Tâche
Ressource	Ressource	Ressource	Ressource
Rôle de ressource	Pas explicite	Objet IEM	Rôle
Flux d'entrée/sortie	Flux d'entrée/sortie	Successeur / Elément de processus	Flux (avec est un flux de contrôle = faux)
Flux de contrôle	Contrôle / Flux (avec déclencheur = vrai)	Contrôle successeur / Elément de processus	Flux (avec est un flux de contrôle = vrai)
Flux de ressource	Ressource / Flux (avec déclencheur = faux)	Ressource successeur / Ressource	<.> (Tâche, Rôle)
Opérateur de connexion	Opérateur logique	Elément de connexion	Point de décision (non port d'entrée/sortie)
Port	Connecteur	Port	Point de décision (port d'entrée/sortie)

Tableau 5.9-1 : Correspondances entre IEM, EEML, GRAI et UEML 1.0

5.10 Glossaire des composants

Chaque composant du méta-modèle UEML 1.0 a ensuite été défini à l'aide d'une définition informelle écrite en langage naturel. Dans les paragraphes suivants, nous donnons une partie du glossaire qui a été défini lors du réseau thématique UEML. Uniquement les composants principaux sont présentés. Pour chaque composant nous donnons donc :

- Une définition informelle,
- la signification de certaines associations qui existent avec ce composant,
- éventuellement certaines contraintes qui s'appliquent à ce composant.

5.10.1 Activité

Définition informelle

Une activité représente une description générique d'une partie du comportement d'une entreprise qui produit des sorties à partir d'un ensemble d'entrées. Une activité peut être commandée et peut employer des ressources.

Associations

- Une activité peut être décomposée en activités,
- une activité peut exiger un ou plusieurs rôles de ressource joués par des ressources,
- une activité est au moins lié à un port d'entrée pour lequel les flux représentent les entrées de l'activité,
- une activité est au moins lié à un port de sortie pour lequel les flux représentent les sorties de l'activité.

Contraintes

Une activité ne peut pas se décomposer en elle-même (directement ou indirectement).

5.10.2 Flux

Définition informelle

Un flux représente le flux d'un objet d'une origine à une cible. L'origine et la cible d'un flux sont des « ancres » qui peuvent être un port d'entrée, un port de sortie ou un opérateur de connexion. Un flux est soit :

- un flux d'entrée/sortie,
- un flux de ressource
- un flux de contrôle.

Un flux porte des objets qui sont compatibles avec son type plus spécialisé. Un flux d'entrée/sortie porte soit des objets, soit des objets informationnels ou soit des ressources. Un flux de ressource porte des ressources. Un flux de contrôle porte des objets informationnels.
Un flux d'entrée/sortie représente le flux d'un objet entre deux activités, l'objet étant une entrée et/ou un résultat pour les activités. L'objet peut soit être consommé, modifié ou créé par l'activité. Si le flux d'entrée/sortie est relié à un port d'entrée d'une activité, alors l'objet est porté par ce flux et est fourni à l'activité à exécuter. Si flux d'entrée/sortie sort d'un port de sortie d'une activité, alors l'objet porté par ce flux est produit par l'activité.
Un flux de ressource représente principalement le flux d'une ressource entre deux activités. Le flux de ressource relie alors le port de sortie de l'activité qui le produit et un port d'entrée de l'activité qui l'exige.

Un flux de contrôle reliant deux activités représente un des trois cas suivants :

- la relation de précédence entre les activités (flux de contrôle sans objet porté),

- le déclenchement d'une activité après une autre (le flux déclenchant portant éventuellement un objet informationnel qui représente l'événement déclenchant de l'Activité cible).

- un flux d'un objet informationnel qui est utilisé pour contraindre l'exécution de l'activité (un Flux de contrainte portant une contrainte sous forme d'objet informationnel tel que par exemple la description d'une procédure qui doit être suivie pour exécuter l'Activité).

Association

Un flux d'entrée/sortie contient tout au plus un objet. Un objet peut être porté par plus d'un Flux d'entrée/sortie.

Un flux de ressource contient tout au plus une ressource. Une ressource peut être porté par plus d'un flux ressource.

Un flux de contrôle porte tout au plus un objet informationnel. Un objet informationnel peut être porté par plus d'un Flux de contrôle.

Contraintes

Un flux ne peut pas avoir la même ancre comme origine et destination.
Un flux de contrainte porte au moins un objet informationnel.
Les flux reliés au même opérateur de connexion doivent appartenir à la même sous-classe du flux.

5.10.3 Opérateur de connexion

Définition informelle

Un opérateur de connexion représente le groupement ou la séparation des flux entre deux activités. C'est un type spécial d'ancre.

Contraintes

Un opérateur de connexion du type « joindre » est la cible d'au moins de deux flux et est l'origine d'exactement un flux.

Un opérateur de connexion du type « séparer » est l'origine d'au moins de deux flux et est la cible d'exactement un flux.

5.10.4 Ressource

Définition informelle

Une ressource est un genre spécial d'objet requis pour l'exécution d'une activité Elle peut représenter une ressource simple ou un ensemble de ressources avec des possibilités et des propriétés semblables. Une ressource peut être une ressource matérielle ou une ressource humaine.

Associations

Une ressource peut jouer plusieurs rôles de ressource.
Une ressource peut être impliquée dans plusieurs flux de ressource.

5.10.5 Rôle de ressource

Définition informelle

Un rôle de ressource définie la nécessité d'une ressource pour une activité. Il définit le rôle joué par une ressource pour une activité.

Un rôle de ressource est utilisé uniquement pour une activité. Une activité peut avoir plusieurs rôles de ressource et une ressource peut jouer plusieurs rôles de ressource.

5.11 Conclusion

Dans ce chapitre, nous avons traité un exemple en prenant en compte trois langages de modélisation d'entreprise. Cette exemple, bien que limité à seulement trois langages, a permit de montrer la difficulté d'élaborer le langage UEML. La comparaison des composants est une tâche ardue qui demande beaucoup d'efforts. Une des idées intéressantes que l'on peut retenir de ces travaux est l'utilisation d'un scénario. En effet, pour l'instant il est tout de même illusoire de vouloir procéder à la comparaison des composants de manière formelle. L'utilisation d'un scénario permet donc un gain de temps. Cependant, la définition d'un scénario pertinent pour l'ensemble des langages est nécessaire. En effet, dans le cadre de ce projet, le scénario était pauvre en terme d'information sur le management ou la prise de décision. De ce fait, la grille et les réseaux GRAI n'ont pas pu être utilisés de manière significative et ont donc été abandonnés. Enfin, le problème dégagé au paragraphe 5.6, se pose ici avec l'utilisation du scénario. Afin de généraliser les correspondances au niveau des modèles à celles au niveau des langages, il est nécessaire, pour être formel, qu'elles soient vraies pour une infinité de scénario.

Par ailleurs, dans ce réseau thématique, l'élaboration d'UEML s'est principalement focalisée sur la traduction d'un modèle écrit dans un langage dans un autre. Ainsi, UEML n'est pas une

réunion de tous les composants des langages, idée à laquelle nous adhérons, mais une intersection.

De plus, la comparaison des composants ne tient pas compte ici, des relations composant-composé que nous avons mis en évidence dans le chapitre précédent. Le chevauchement des composants n'est en effet traité que d'un point de vue de la généralisation (spécialisation). De même, la démarche à suivre pour traiter ce type de comparaison n'est pas vraiment clairement explicitée. En effet, on ne sait pas trop comment elle se généralise dans le cas où, par exemple, plusieurs composants présentant des caractéristiques communes doivent être comparés.

Enfin, le langage a été construit en utilisant principalement une approche ascendante. Bien qu'une étude concernant les fonctionnalités d'UEML ait été réalisée, celle-ci n'a pas été liée au développement du langage UEML proprement dit. En effet, les tâches concernant la construction d'UEML et la définition des fonctionnalités ont été menées de manière séparée. Ce n'est que par la suite, que le problème de savoir si le langage UEML construit permettait effectivement de remplir les fonctionnalités identifiées, a été traité.

Conclusions, limites et perspectives

Tout au long de ce mémoire, nous nous sommes attachés à répondre à la problématique concernant la définition d'un langage unifié de modélisation d'entreprise. Nous avons montré qu'un certain nombre de langages de modélisation d'entreprise actuels possèdent des intersections entre eux, mais qu'il n'existe pas de langage suffisamment intégré permettant de représenter l'ensemble de l'information contenu dans ces langages. Nous avons pu cependant souligner, que diverses approches, mêmes si elles ne sont pas assez abouties pour être un langage unifié, apportent des résultats intéressants sur le sujet (entre autre PSL et ENV12204/CEN/ISO 19440). Néanmoins, ces efforts ne fournissent pas, à l'heure actuelle, une solution satisfaisante concernant notre problématique.

Ensuite, nous avons présenté notre point de vue, ainsi que notre démarche concernant l'élaboration du langage UEML. Nous avons exposé un ensemble de principes relatifs à la comparaison des composants. Nous avons, ainsi, distingué le cas de l'inclusion de type ensembliste de celle de type composant-composé. Nous avons utilisé une approche ensembliste, afin de pouvoir écrire un certain nombre d'équations pour permettre de déterminer les composants élémentaires avec les composants des langages, et ceci d'une manière systématique et automatisable dans une certaine mesure. En effet, les premières comparaisons entre les composants ne sont pas automatisables tant que le problème de la représentation de la sémantique des composants n'a pas eu de solution satisfaisante. Nous avons illustré ces principes au travers d'un exemple simple. Nous avons présenté brièvement l'utilisation que l'on peut faire d'un langage comme OCL, afin de lever un certain nombre d'ambiguïtés que véhicule le diagramme de classe UML. Ce langage ne peut cependant pas résoudre toutes les ambiguïtés de représentation. Nous avons également montré que le

développement d'un langage unifié nécessite, d'une part, une représentation formelle des méta-modèles, et d'autre part, que les composants des langages soient définis de manière plus formelle. Enfin, nous avons traité un exemple en prenant en compte trois langages de modélisation d'entreprise. Cet exemple, bien que limité, a permit de montrer la difficulté d'élaborer le langage UEML.

Cependant, les travaux de recherche présentés ici, ne sont qu'une première approche concernant l'élaboration d'UEML. Le but de ces travaux n'est pas le développement d'un langage UEML, mais plutôt de dégager certains principes et éléments de réflexion. Beaucoup d'efforts sont encore nécessaires. Par exemple, nous avons dit quelques mots sur la comparaison des composants d'un point de vue de la composition, mais nous n'avons pas réellement donné de solution afin de traiter ce problème. La non résolution de cette difficulté reste cependant un obstacle à la définition de notre langage unifié. De plus, la comparaison des composants est une tâche difficile qui demande beaucoup d'efforts. Une des idées intéressantes que l'on peut retenir est l'utilisation d'un scénario. En effet, pour l'instant il est illusoire de vouloir procéder à la comparaison des composants de manière formelle. L'utilisation d'un scénario permet donc un gain de temps. La difficulté étant de définir un scénario pertinent pour l'ensemble des langages. De plus, le problème qui se pose avec l'utilisation du scénario est que les correspondances au niveau des modèles ne peuvent être généralisées à celles au niveau des langages que si elles sont vraies pour une infinité de scenarios. Par ailleurs, nous avons également vu que le développement d'UEML nécessite que les langages soient définis de manière plus formelle, afin d'éviter de véhiculer certaines ambiguïtés. Comme nous l'avons déjà dit, les méthodes de modélisation d'entreprise actuelles proposant plusieurs vues (et donc souvent plusieurs langages), souffrent d'un manque de rigueur concernant la définition des interactions entre les composants des différents langages (de chaque vue). Il faut donc préciser et ainsi rendre la définition des langages plus cohérente, afin de ne pas bloquer le développement d'UEML.

Par ailleurs, la démarche d'élaboration d'UEML que nous avons présentée (approche hybride), n'a pas été utilisée. En effet, les exemples que nous avons traités n'étaient qu'uniquement basés sur une approche ascendante. Une étude des fonctionnalités attendues pour UEML est donc nécessaire, afin de choisir les bons langages à intégrer.

Enfin, la problématique d'UEML s'inscrit dans une problématique plus large qui est celle de l'interopérabilité des applications d'entreprises. En effet, la compétitivité des entreprises est partiellement déterminée par leur aptitude à progressivement interopérer avec d'autres organisations, comme dans les pratiques de commerce collaboratif. L'interopérabilité est un des facteurs critiques de succès, afin que les entreprises deviennent plus flexibles et que les efforts dus à l'établissement ainsi qu'à la pérennité de la coopération inter-entreprises soient réduits. Au niveau Européen, le projet IDEAS (IST IST-2001-37368) avait pour but de produire un plan de recherche à long terme concernant l'amélioration de l'interopérabilité des applications d'entreprises. Les travaux de recherche concernant l'élaboration d'un langage unifié de modélisation d'entreprises sont donc une contribution à ces travaux.

Ainsi, ces dernières années, de nombreux travaux concernant l'amélioration de l'interopérabilité (principalement l'interopérabilité de logiciel) on été développés tel que :

- les conférences internationales ICEIMT],
- les travaux des organismes de normalisation (CEN et ISO),
- les travaux plus récents de l'« Interest group on UEML » de la Task Force IFAC-IFIP,
- les travaux du groupe de travail européen UEML (IST-2001-34229)
- les travaux du groupe de travail "Modélisation d'entreprise" du Groupement de Recherche en Productique.

Ces travaux continue actuellement aux travers de deux collaborations européennes : le réseau d'excellence INTEROP (Interoperability Research for Networked Enterprises Applications and Software ; 6ème PCRD, IS 508011) et le projet intégré ATHENA (Advanced Technologies for interoperability of Heterogeneous Enterprise Networks and their Applications ; 6ème PCRD, IST 507849).

Beaucoup de travaux sont encore nécessaires afin d'élaborer de manière correcte notre langage unifié de modélisation d'entreprise. Dans des travaux futurs notre objectif est de clarifier et de proposer une méthode générique permettant de comparer des composants. Cette comparaison doit tenir compte des liens qui existent entre les composants, c'est-à-dire des relations de composition ou de généralisation. De plus, la méthode devra permettre d'inclure d'autres types de relations si nécessaires. Par ailleurs, notre approche hybride concernant

l'élaboration d'UEML doit être définie de manière plus complète et une application de cette méthode sur un cas réel permettra probablement de l'améliorer, ou de faire ressortir de nouvelles difficultés oubliées jusqu'à présent. Par ailleurs, nous avons souligné tout au long de ce mémoire que les langages de modélisation d'entreprise souffrent d'un manque de définition formelle de leur sémantique. Des travaux devront donc être mené dans ce sens.

De plus, dans ce mémoire, nous avons utilisé le diagramme de classe UML afin de représenter les méta-modèles des langages. Cependant, même avec l'utilisation du langage OCL ce type de représentation n'est pas totalement formel. Nous restons partisans de ce type de représentation car elle est facile à comprendre et donne une bonne vision de l'ensemble des composants des langages ainsi que les relations entre eux. Toutefois, l'utilisation de langage tel que le langage Z ou la méthode B qui sont des langages formels, peut être une idée intéressante. De plus, des travaux existent et d'autres sont en cours concernant la traduction de diagrammes de classe dans ce type de langages. Même si, des informations supplémentaires seront forcément nécessaires afin de compléter les modèles dans ces langages, cette manière de procéder peut permettre d'avoir deux niveaux de représentation des méta-modèles. Une représentation plus proche de la modélisation d'entreprise et réalisée par des experts des langages, tel que nous l'avons fait dans ce mémoire en fin de compte. Cette représentation est un diagramme de classe plus ou moins formalisé grâce au langage OCL par exemple. Par la suite, une deuxième représentation peut être extraite de la première. Cette vision est davantage une vision informatique du méta-modèles et a pour but de permettre l'implantation des méta-modèles dans un progiciel en éliminant les ambiguïtés et en rendant les méta-modèles formels. Un ensemble de questions venant des informaticiens vers les experts en modélisation d'entreprise sera nécessaire et éventuellement des travaux supplémentaires concernant la formalisation des langages de modélisation d'entreprise seront à entreprendre.

Cependant, il va sans dire que ce type de travaux nécessitent une collaboration très étroite entre deux communautés scientifique (le génie logiciel et la modélisation d'entreprise) et à notre sens cette collaboration est nécessaire si on veut mener à bien ces travaux concernant la définition d'un langage de modélisation d'entreprise.

Bibliographie

A

[AICOSCOP 1990] AICOSCOP, 1990. Rapport Final du contrat de Recherche MRT FRT N°88.00661.

[AMICE, 1993] AMICE, 1993. CIMOSA : Open Architecture for CIM. Berlin, Springer Verlag.

B

[Braesch *et al.*, 1995] Ch. BRAESCH, A. HAURAT et J.-M. BEVING, 1995. – L'entreprise-système – dans la modélisation systémique en entreprise, Paris : Hermès.

[Braesch, 2002] Braesch Ch., 2002. Le modèle OLYMPIOS. Supports de cours de l'école de printemps « Modélisation d'entreprise », Groupe de travail n°5 du Groupement pour la recherche en productique, Albi-Carmaux, France, 28-30 mai.

[Berio et al., 2004] Berio G., Panetto H, Petit Michael : Résultats et enjeux d'un langage unifié de modélisation d'entreprise – dans *Proc. de* la 5ième *Conférence Francophone de Modélisation et Simulation, MOSIM*, Nantes, France, 1-3 septembre 2004.

C

[CEN ENV 40003, 1990] CEN, 1990. – ENV 40003 - computer integrated manufacturing - CIM systems architecture framework for modelling. – comité européen de normalisation (CEN), Bruxelles, Belgique.

[CEN ENV 40003, 1990] CEN, 1990. – ENV 40003 - computer integrated manufacturing - CIM systems architecture framework for modelling. – comité européen de normalisation (CEN), Bruxelles, Belgique.

[CEN ENV 12204, 1995] CEN, 1996. – ENV 12204 - advanced manufacturing technology - systems architecture - constructs for enterprise modelling. – rapport technique, comité européen de normalisation (CEN), Bruxelles, Belgique, février.

[CEN ENV 12204, 1995] CEN, 1996. – ENV 12204 - advanced manufacturing technology - systems architecture - constructs for enterprise modelling. – rapport technique, comité européen de normalisation (CEN), Bruxelles, Belgique, février.

[CEN ENV 13550, 1999] CEN, 1995. – ENV 13550 - enterprise model execution and integration services (emeis). – rapport technique, comité européen de normalisation (CEN), Bruxelles, Belgique.

[CEN ENV 13550, 1999] CEN, 1995. – ENV 13550 - enterprise model execution and integration services (emeis). – rapport technique, comité européen de normalisation (CEN), Bruxelles, Belgique.

[CEN TC310/WG1, 2002] CEN TC310/WG1, 2002. – CIM system architecture - framework for enterprise integration. – Rapport technique Pre EN ISOV 19439, comité européen de normalisation (CEN), Bruxelles, Belgique.

[Chapurlat *et al.*, 1999] Charpulat V., Larnac M.,Lamine E. et Magnier J., 1999. Definition of a formal analysis framework for existing enterprise modelling approaches. Actes de Annual conf. of ICIMS-NOE, Life cycle approaches to production systems : management, control, supervision (ASI), Louvain, Belgique, 22-24 septembre.

[Chen *et al.*, 1999] D. Chen, B. Vallespir. – Theories of design and enterprise modeling. – in actes de Annual conference of ICIMS-NOE, Life cycle approaches to production systems: management, control, supervision, ASI, Louvain, Belgique, 22-24 septembre 1999.

[Chen *et al.*, 2000] D. CHEN, B. VALLESPIR, G. DOUMEINGTS. – Enterprise modelling and engineering, some complementary requirements on the standardisation. – in actes de CEN TC310 WG1 Workshop on evolution of enterprise engineering and integration, Berlin, Allemagne, 24-26 mai 2000.

[Chen *et al.*, 2001] D. CHEN et F. VERNADAT, 2001. – Standardisation on Enterprise Modelling and Integration : Achievements, on-going Works and Future Perspectives. – in actes du 10ème symposium Information Control in Manufacturing (INCOM'01), Vienne, Autriche, 20-22 septembre.

[Chen *et al.*, 2002] D. Chen, B. Vallespir, G. Doumeingts. – Developing an unified enterprise modelling language (UEML) – Roadmap and requirements. – in actes de *3rd IFIP Working conference on infrastructures for virtual enterprise, PROVE*, Sesimbra, Portugal, 1er-3 mai 2002 – Collaborative Business Ecosystems and Virtual Enterprises, Kluwer Academic Publishers.

[Chen, 2002] D. CHEN, 2002. – Standards on enterprise integration and interoperability. – présentation orale à 1er EI3-IC workshop, EADS, Paris, France, 5-7 décembre.

[Chen et al., 2004] D. Chen et F. Vernadat, Standards on enterprise integration and engineering – A state of the art, dans International Journal of Computer Integrated Manufacturing (IJCIM), Volume 17, n°3, Avril-Mai 2004, pp.235-253.

D

[Deschamps et al., 2004] Deschamps J.C., Roque M., Vallespir B. - Modélisation de processus par réseaux de transformateurs : définition et propriétés – dans Proc. de la 5ième Conférence Francophone de Modélisation et Simulation, MOSIM, Nantes, France, 1-3 septembre 2004.

[Doumeingts *et al.*, 1987] G. Doumeingts, B. Vallespir, D. Darricau, M. Roboam. – La conception des systèmes avancés de production. – in actes du Colloque international La Productique : Environnement international industriel et économique, Toulouse, France, 1-2 octobre 1987.

[Doumeingts *et al.*, 1993] G. Doumeingts, D. Chen, B. Vallespir, P. Fénié. – GIM (GRAI Integrated Methodology) and its evolutions - A methodology to design and specify advanced manufacturing systems. – in actes de *IFIP WG5.3 workshop on the design of information infrastructure systems for manufacturing*, *DIISM,* Tokyo, Japon, 8-10 novembre 1993 – IFIP Transations B14, H. Yoshikawa et J. Goossenaerts ed., Amsterdam : Elsevier, 1993.

[Doumeingts *et al.*, 1995] G. Doumeingts, B. Vallespir, F. Marcotte. – A proposal for an integrated model of manufacturing system : application to the re-engineering of an assembly shop. – in Control Engineering Practice, Special section on models for management and control, vol. 3, n° 1, Oxford : Pergamon Press Ltd, 1995.

[Doumeingts *et al.*, 2001] G.Doumeingts, T.Despoix : documentation interne GRAISOFT 2001

[Doumeingts, 1984] Doumeingts, G.,Méthode GRAI, méthode de conception des systèmes en productique, Thèse de Doctorat, Université Bordeaux I, 1984.

[Doumeingts, 1990] G. DOUMEINGTS « Méthodes pour concevoir et spécifier les systèmes de production » actes du colloque international CIM'90, Bordeaux, pp 89-103, juin 1990.

[Doumeingts, 1994] G.DOUMEINGTS, B. VALLESPIR "Gestion de Production : Principes", Techniques de l'ingénieur, Référence A 8 265 , Novembre 1994, 24p.

[Doumeingts, 1998] G. DOUMEINGTS "Cours de Productique" Formation de DEA APSI, Université Bordeaux 1, 1999/2000.

E

[El Mhamedi *et al.*, 1997] El Mhamedi A., Leerch C., Marier S., Sonntag M. et Vernadat F., 1997. Intégration des ACtivités NOn Structurées dans la Modélisation des Systèmes de Production. Rapport Final, Action Incitative du DSPT8 en Productique, février.

[El Mhamedi, 2002] El Mhamedi A., 2002. La méthode ACNOS. Supports de cours de l'école de printemps « Modélisation d'entreprise », Groupe de travail n°5 du Groupement pour la recherche en productique, GRP, Albi-Carmaux, France, 28-30 mai.

F

[Falkenberg et al., 1998] E.D. Falkenberg, W. Hesse, P. Lindgreen, B.E. Nilsson, J.L.H. Oei, C. Rolland, R.K. Stamper, F.J.M. Van Assche, A.A. Verrijn-Stuart, K. Voss: FRISCO - A Framework of Information System Concepts - The FRISCO Report. IFIP WG 8.1 Task Group FRISCO, 1998.

[Ferdinand de Saussure, 1983] Ferdinand de Saussure, Cours dans General Linguistics, trans. Roy Harris (La Salle, Ill.: Open Court, 1983).

G

[GERAM, 1999] GERAM, 1999. GERAM : Generalised Enterprise Reference Architecture and Methodology. Version 1.6.1, IFIP–IFAC Task Force on Architectures for Enterprise Integration, Mars.

[Giard, 2003] V.Giard, Gestion de la production et des flux, Economica, 2003.

[GRP/GT5, 1999] La Modélisation d'Entreprise : le point de vue productique, Version 1.1, Mai 1999, Document de référence du GT5 "Modélisation d'Entreprise" du GRP.

I

[ISO 10303-1, 1993] Industrial Automation Systems and Integration – Product Data Representation and Exchange – Part 1 : Overview and Fundamental Principles, ISO TC184/SC4.

[ISO 14258, 1999] Industrial automation systems - concepts and rules for enterprise models – ISO TC184/SG5/WG1, 1999.

[ISO 15704, 1998] Requirements for enterprise - reference architectures and methodologies. – ISO TC184/SC5/WH1, N423.

[ISO CD 18629-1, 2001] Industrial Automation Systems and Integration, Process Specification Language (PSL), Part1: Overview and basic Principles, JW8/ISO 184/SC4/SC5.

[ISO DIS 10303-11, 1992] The EXPRESS Language Reference Manual, ISO TC184/SC4/WG5, N35.

[ISO FCD, 2000] ISO FCD, 2000. – Information technology - open distributed processing - reference model - enterprise viewpoint. – 15414.

[ISO TC154/WG1, 1999] ISO TC154/WG1, 1999. – ISO 16668 - basic semantic register (bsr) - rules, guidelines and methodology. – N007, 31 août.

[ISO TC184/SG5, 1999a] ISO TC184/SG5, 1999a. – Industrial automation systems - concepts and rules for enterprise models. – Rapport technique ISO 14258, ISO.

[ISO TC184/SG5, 1999b] ISO TC184/SG5, 1999b. – Industrial automation systems - requirements for enterprise - reference architectures and methodologies. – Rapport technique ISO/FDIS 15704, ISO.

[ISO/CEN 19439, 2002] Enterprise Integration – Framework for enterprise Modelling, Final draft, CEN TC310/WG1 et ISO TC184 SC5/WG1.

[ISO/CEN 19440, 2002] Language Constructs for Enterprise Modelling, Final draft, CEN TC310/WG1 et ISO TC184 SC5/WG1.

[ISO/IEC CD 15288, 1999] ISO IEC, 1999. – Life cycle management system - life cycle processes. – N2184, CD 15288, ISO/JTC1/SC7, 19 juillet.

[ISO/IEC CD 15288, 1999] ISO IEC, 1999. – Life cycle management system - life cycle processes. – N2184, CD 15288, ISO/JTC1/SC7, 19 juillet.

[ISO/IEC CD 15414, 2000] Information technology – Open Distributed Processing – Reference model – Enterprise Language, ITU-T Recommendation X.911, Version 303, ISO/IEC JTC 1/SC7/WG 17.

K

[Knothe *et al.*, 2003] Knotte T., Busselt C. and Böll D. – Rapport sur UEML (needs and requirements). – Réseau thématique UEML-Contrat n°: IST – 2001 – 34229, Rapport du Work Package 1, April 2003.

L

[Le Moigne, 1974] J.L. LE MOIGNE, "Les systèmes de décision dans les organisations", Presses universitaires de France, Paris,1974.

[Le Moigne, 1977] J.L. LE MOIGNE, "La théorie du système général. Théorie de la modélisation", Presses universitaires de France, Paris, 1977.

[Le Moigne, 1990] J.L. LE MOIGNE, "La modélisation des systèmes complexes", Bordas, 1990.

M

[Mayer *et al.*, 1992] Mayer, R.J., Cullinane, T.P., deWitte, P.S., Knappenberger, W.B., Perakath, B., Wells, M.S. (1992) Information Integration for Concurrent Engineering (IICE) IDEF3 Process Description Capture Method Report. Air Force Systems Command, Wright-Patterson Air Force Base, Ohio 45433, AL-TR-1992-0057.

[Mélèse, 1972] J. MELESE "L'analyse modulaire des systèmes de gestion", Editions Hommes et Techniques, 1972.

[Morand, 1995] Morand B., Statut épistémologique des modèles dans la conception des systèmes 'information. Revue Ingénierie des Systèmes d'Information, vol 3, n°5, pp. 665-700, 1995.

O

[OMG, 2003] OMG. – Unified Modeling Language Specification. – Version 1.5, formal / 03-03-0, 2003.

[Oussalah, 1997] C. OUSSALAH « Ingénierie Objet, Concepts et techniques » InterEditions, 1997.

P

[Parent *et al.*, 2000] Parent, C., Spaccapietra, S. Database integration: the key to data interoperability. In Papazouglu M.P. Spaccapietra S. Tari (Eds), Advances in Object-Oriented Data Modeling. MIT Press, 2000.

[Panetto *et al.*, 2001] H. PANETTO, F. MAYER et P. LHOSTE, 2001. – Unified Modelling Language for meta-modelling : towards Constructs definition. – in actes du 10ème symposium Information Control in Manufacturing (INCOM'01), Vienne, Autriche, 20-22 septembre.

[Panetto et al., 2002] UML Semantics Representation of Enterprise Modelling Constructs. In K.Kosanke, R. Jochem, J.G. Nell, A.O. Baz (eds.); Proceedings of International Conference on Enterprise Integration and Modeling Technology (ICEIMT'02), 381-387.

[Panetto et al., 2004] Panetto H., Berio G., Benali K., Boudjlida N., Petit M. (2004). A Unified Enterprise Modelling Language for enhanced interoperability of Enterprise Models. *Proceedings of the 11th IFAC INCOM2004 Symposium*, April 5th-7th, Bahia, Brazil

[Petit *et al.*, 1997] Petit M., Goossenaerts J., Gruninger M.,. Nell J. G. et Vernadat F., 1997. Formal Semantics of Enterprise Models, Actes de Int. Conf. on Enterprise Integration and Modelling Technology (ICEIMT'97), Turin, Italie, 28-30 octobre, Kosanke K. et Nell J. ed., Berlin, Springer.

[Petit *et al.*, 2002] Petit M. UEML Thematic Network - Contract n°: IST – 2001 – 34229 – Work Package 1 Report - Enterprise Modelling State of the Art, October 2002.

[Petit, 2003] Michaël Petit, Some methodological clues for defining a Unified Enterprise Modelling Language, In Enterprise Inter- and Intra-organisational Intergration - Building an International Consensus, Kurt Kosanke, Roland Jochem, James G. Nell and Angel Ortiz Bas (editors), Kluwer Academic Publishers, 2003, ISBN 1-4020-7277-5.

[Pourcel *et al.*, 2002] Pourcel C. et Gourc D., 2002. Modélisation MECI. Supports de cours de l'école de printemps « Modélisation d'entreprise », Groupe de travail n°5 du Groupement pour la recherche en productique, GRP, Albi-Carmaux, France, 28-30 mai.

R

[Roque *et al.*, 2002] M. Roque, B. Vallespir G. Doumeingts. – Contribution au développement d'UEML : Principes et projet. – Séminaire du groupement pour la recherche en productique, GRP, Tarbes, France, 24-25 octobre 2002. Communication orale.

[Roque *et al.*, 2004] Roque M., Vallespir B. and Doumeingts G. - Modélisation d'entreprise traduction et méta-modélisation. 5ème Conférence Francophone de Modélisation et Simulation, MOSIM, Nantes, France, 1-3 September 2004

[Roque *et al.*, 2005] Roque M., Vallespir B. and Doumeingts G. From a models translation case towards identification of some issues about UEML – in Proc. of the workshop on Entreprise Integration, Interoperability and Networking (EI2N), Geneva, Switzerland, February 22, 2005.

S

[Scheer, 2002] W. SCHEER, 2002. – ARIS : des processus de gestion au système intégré d'applications. – M. Flöter-Morrie trad., Berlin : Springer.

[Schlenoff *et al.*, 2000] Schlenoff, C., Gruninger M., Tissot, F., Valois, J., Lubell, J., Lee, J., The Process Specification Language (PSL) : Overview and Version 1.0 Specification, NISTIR 6459, National Institute of Standards and Technology, Gaithersburg, MD(2000).

[Simon, 1960] H.A. SIMON, "The new science of management decision", Harper and Row publisher, New York et Evanston, 1960.

[Simon, 1977] H.A. SIMON, "Administrative behavior. A study of decision making processes in manufacturing organization", MacMillan Ed., New York, Troisième édition, 1977.

[Spur *et al.*, 1996] G. Spur, K. Mertins, and R. Jochem. *Integrated Enterprise Modelling*. Beuth Verlag GmbH, Berlin, 1996. ISBN 3-410-13310-0.

T

[Theroude, 2002] Theroude, 2002. Formalisme et système pour la représentation et la mise en œuvre des processus de pilotage des relations entre donneurs d'ordres et fournisseurs. Thèse de l'Institut National Polytechnique de Grenoble, 17 juillet.

[Tardieu *et al.*, 1983] Tardieu, Rochfeld, Colletti - La méthode MERISE (Tome 1 et 2) - Editions d'organisation, 1983

V

[Vallespir *et al.*, 1992] B. VALLESPIR, C. MERLE, G. DOUMEINGTS. – The GRAI Integrated Method : a technico-economical methodology to design manufacturing systems. – in actes de 1st IFAC workshop on « A cost effective use of computer aided technologies and integration methods in small and medium sized companies », CIM'92, Vienne, Autriche, 7-8 septembre 1992.

[Vallespir *et al.*, 1993] B. VALLESPIR, C. MERLE, G. DOUMEINGTS. – GIM : a technico-economic methodology to design manufacturing systems. – in Control Engineering Practice, vol. 1, n°6, Oxford : Pergamon Press Ltd, 1993.

[Vallespir *et al.*, 1997] B. Vallespir, G. Doumeingts. – La méthode GRAI. – in supports de cours de l'Ecole de printemps « Modélisation d'entreprise », Groupe de travail n°5 du Groupement pour la recherche en productique, GRP, Albi-Carmaux, France, 28-30 mai 2002.

[Vallespir *et al.*, 1999] B. Vallespir, S. Kleinhans, G. Doumeingts. – Enterprise modelling and performance: part 2. benchmarking of industrial business processes. – in International Journal of Business Performance Management, vol. 1, n° 2, Genève : Inderscience Publishers, 1999.

[Vallespir *et al.*, 2002] Vallespir B. et Doumeingts G., 2002. La méthode GRAI. Supports de cours de l'école de printemps « Modélisation d'entreprise », Groupe de travail n°5 du Groupement pour la recherche en productique, GRP, Albi-Carmaux, France, 28-30 mai.

[Vallespir, 2003] B. Vallespir. – Modélisation d'entreprise et architecture de conduite des systèmes de production. – Mémoire d'Habilitation à Diriger des Recherches, Université Bordeaux 1, 19 décembre 2003.

[Vernadat, 1999a] Vernadat F., 1999. Techniques de Modélisation en Entreprise : Application aux Processus Opérationnels. Paris, Economica, Collection Gestion.

[Vernadat, 1999b] Vernadat F., 1999. Unified Enterprise Modelling Language (UEML). Présentation orale à IFAC IFIP Task force Interest group on UEML, Paris, France, 16 décembre.

[Vernadat, 2001] Vernadat F., 2001. UEML : Towards a Unified Enterprise Modelling Language. Actes de la 3ème Conf. Francophone de Modélisation et Simulation (MOSIM'01), Troyes, France, 25-27 avril.

[Volle, 2003]. http://www.volle.com/travaux/ensemble.htm

W

[Warmer et al., 1999] Warmer, J., Kleppe, A. The Object Constraint Language: Precise Modeling with UML Addison-Wesley 1999.

[Williams, 1996] Williams T. J., 1996. An Overview of PERA and the Purdue Methodology. in Architectures for Enterprise Integration, Bernus P., Nemes L. et Williams T. J. ed., Londres, Chapman et Hall.

Z

[Zachman, 1987] J. Zachman, "A Framework for Information Systems Architecture", *IBM Systems Journal*, Vol. 26, No. 3, 1987.

[Zanettin, 1994] M. Zanettin. – Contribution à la conception des Systèmes de Production. – Thèse de Doctorat de l'Université Bordeaux 1, Spécialité Productique, 31 mars 1994.